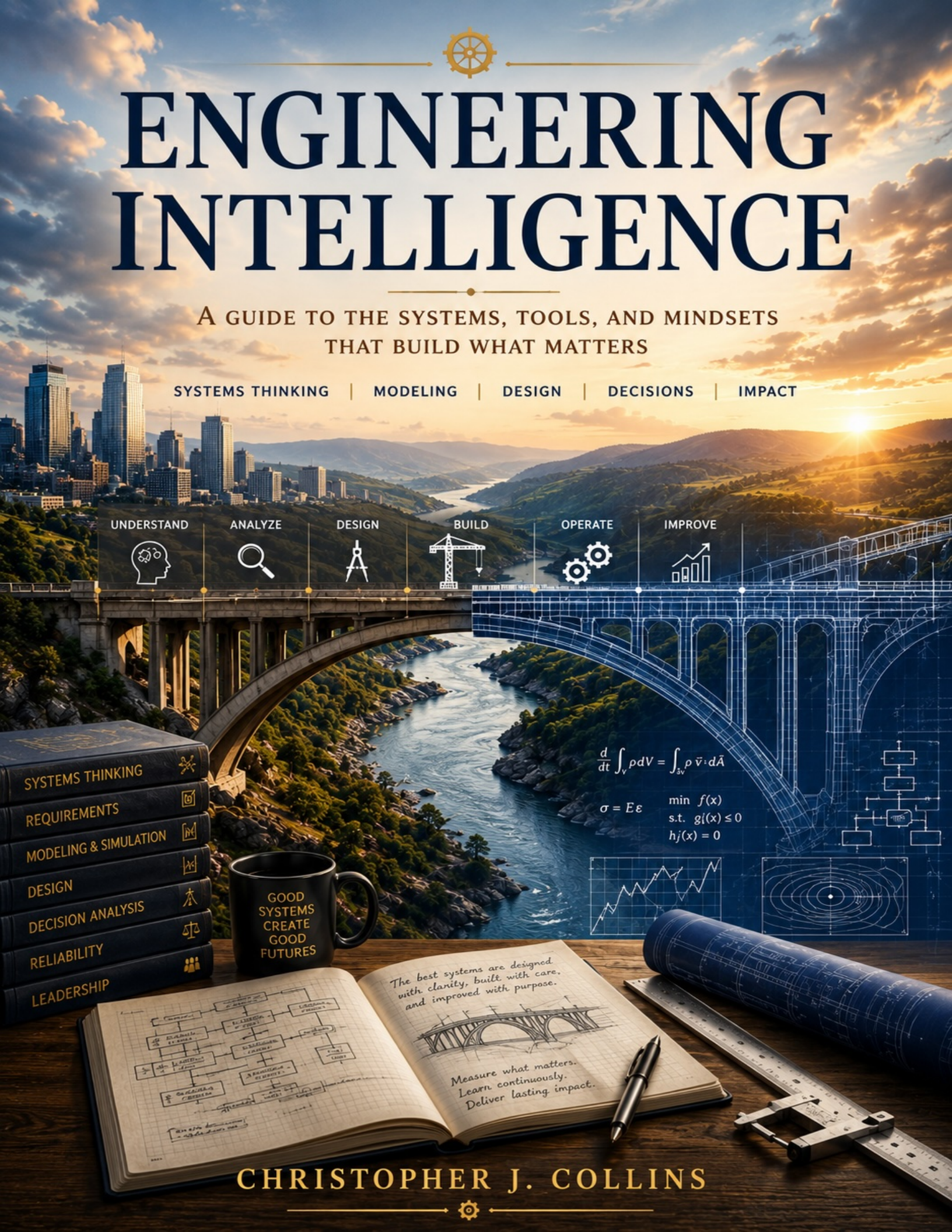
ENGINEERING INTELLIGENCE

A GUIDE TO THE SYSTEMS, TOOLS, AND MINDSETS THAT BUILD WHAT MATTERS

SYSTEMS THINKING | MODELING | DESIGN | DECISIONS | IMPACT



CHRISTOPHER J. COLLINS

Table of Contents

1. Copyright and publication . . . **1**
2. Dedication . . . **3**
Preface . . . **5**
3. Acknowledgments . . . **7**
Executive Summary . . . **9**
The central thesis . . . 9
Engineering is an attention-allocation system . . . 9
Engineering is a queueing system . . . 10
Engineering is an economic system . . . 10
Metrics are decision products . . . 11
Measurement must support causal learning . . . 12
AI changes the production system . . . 12
Architecture is measurable consequence . . . 13
The platform is an evidence system . . . 13
Governance is part of measurement quality . . . 14
The operating model . . . 14
A maturity path . . . 15
Stage 1: trustworthy evidence . . . 15
Stage 2: Engineering Change intelligence . . . 15
Stage 3: governed measurement . . . 15
Stage 4: causal learning . . . 15
Stage 5: economic and portfolio intelligence . . . 15
Stage 6: AI and agent evaluation . . . 15
Stage 7: bounded adaptive policy . . . 15
What success looks like . . . 15
4. Reader's Guide . . . **17**
4.1. For engineering executives . . . 17
4.2. For platform and developer-productivity leaders . . . 17
4.3. For measurement scientists and analysts . . . 17
4.4. For architects and senior engineers . . . 18
4.5. For AI engineering and evaluation teams . . . 18
4.6. For security, privacy, legal, and governance teams . . . 18
4.7. For practitioners and team leaders . . . 18
4.8. How the book is organized . . . 19
4.8.1. Foundations . . . 19
4.8.2. Measurement and causal learning . . . 19
4.8.3. AI, instrumentation, and architecture . . . 19
4.8.4. Operational methods . . . 19
4.8.5. Scientific and organizational maturity . . . 19
4.8.6. Reference standards . . . 19
4.9. Reading conventions . . . 19
4.10. Applying the framework to a real decision . . . 20
4.11. What not to do . . . 20
4.12. A note on context . . . 20
4.13. A note on evidence maturity . . . 21
4.14. A note on terminology . . . 21
4.15. Using the templates . . . 21
4.16. The intended outcome . . . 21

5. Introduction . . . **23**
5.1. Why engineering needs intelligence . . . 23
5.2. From dashboards to a learning system . . . 24
5.3. The limits of familiar productivity measures . . . 25
5.4. The engineering change as the unit of analysis . . . 25
5.4.1. Change reconstruction . . . 27
5.5. Engineering is a socio-technical system . . . 28
5.6. Human attention as an economic resource . . . 29
5.7. AI as a first-class engineering participant . . . 29
5.8. Measurement for decisions . . . 30
5.9. The scientific method and engineering experimentation . . . 31
5.10. A reference architecture for Engineering Intelligence . . . 31
5.11. Scope of this book . . . 32
5.12. Key takeaways . . . 32
5.13. Open questions . . . 33
5.14. Implementation checklist . . . 33
5.14.1. Research foundations and source notes . . . 33
6. Engineering Economics . . . **37**
6.1. Why cost is harder than it appears . . . 37
6.2. Cost, price, value, and investment . . . 38
6.3. The total cost of an Engineering Change . . . 38
6.4. Human effort cost . . . 39
6.4.1. Active time versus elapsed time . . . 39
6.4.2. Attention-weighted effort . . . 40
6.4.3. Context recovery and interruption cost . . . 40
6.5. Tool and AI cost . . . 40
6.6. Infrastructure cost . . . 41
6.7. Coordination cost . . . 42
6.8. Delay and cost of delay . . . 43
6.8.1. Queueing and inventory cost . . . 44
6.9. Quality cost . . . 44
6.9.1. Prevention, appraisal, and failure . . . 45
6.10. Risk and tail events . . . 45
6.11. Value and return . . . 46
6.12. Worked example: evaluating an AI review tool . . . 47
6.13. Worked example: a required specialist approval . . . 48
6.14. Estimation methods . . . 48
6.14.1. Calibration samples . . . 49
6.14.2. Hierarchical models . . . 49
6.15. Marginal versus average economics . . . 49
6.16. Uncertainty and sensitivity analysis . . . 50
6.17. Common failure modes . . . 50
6.17.1. Monetizing everything . . . 50
6.17.2. Treating salary as value . . . 50
6.17.3. Ignoring displaced work . . . 50
6.17.4. Double counting . . . 51
6.17.5. Optimizing component cost . . . 51
6.17.6. Using cost metrics for individual ranking . . . 51
6.18. Designing an economic scorecard . . . 51
6.19. Key takeaways . . . 51

6.20. Open questions . . . 52
6.21. Implementation checklist . . . 52
6.21.1. Research foundations and source notes . . . 52
7. The Engineering Change . . . 57
7.1. Why the pull request is insufficient . . . 57
7.2. Formal definition . . . 58
7.2.1. Engineering intent . . . 58
7.2.2. Artifacts and activities . . . 58
7.2.3. Relationships . . . 59
7.2.4. Outcomes . . . 59
7.3. The Engineering Change graph . . . 59
7.4. Boundaries and identity . . . 59
7.4.1. Parent and child changes . . . 60
7.4.2. Merge and split operations . . . 60
7.5. Lifecycle states . . . 60
7.5.1. Event time and processing time . . . 60
7.6. Deterministic reconstruction . . . 60
7.6.1. Adoption strategy . . . 61
7.7. Probabilistic reconstruction . . . 61
7.7.1. Negative evidence . . . 61
7.7.2. Confidence . . . 61
7.8. Provenance and explainability . . . 62
7.9. Multi-repository changes . . . 62
7.10. Repository topology as context . . . 63
7.11. Relationship to initiatives, tickets, and releases . . . 63
7.12. Membership over time . . . 64
7.13. Cost and value allocation . . . 65
7.14. Instrumentation architecture . . . 65
7.15. Near-real-time and batch responsibilities . . . 65
7.16. Data-quality metrics . . . 66
7.17. Validation of reconstruction . . . 66
7.18. Human correction and active learning . . . 66
7.19. Privacy and governance . . . 66
7.20. Common failure modes . . . 66
7.20.1. Treating ticket links as ground truth . . . 67
7.20.2. Clustering by author and time alone . . . 67
7.20.3. Ignoring deployment and production evidence . . . 67
7.20.4. Hiding uncertainty . . . 67
7.20.5. Building one giant initiative node . . . 67
7.20.6. Recomputing history without versioning . . . 67
7.21. Worked example: reconstructing an Engineering Change under confidence . . . 67
7.22. Key takeaways . . . 68
7.23. Open questions . . . 68
7.24. Implementation checklist . . . 68
7.24.1. Research foundations and source notes . . . 68
8. Human Attention and Cognitive Load . . . 75
8.1. Why attention is the scarce resource . . . 75
8.2. Cognitive burden is multidimensional . . . 76
8.3. A cognitive work model . . . 76
8.4. Measuring latent burden through proxies . . . 77

8.5. Context switching and attention residue . . . 77
8.5.1. Concurrent work in progress . . . 78
8.6. Review as cognitive work . . . 78
8.7. Comment counts are weak measures . . . 79
8.8. AI as a cognitive participant . . . 79
8.9. Why acceptance rate is insufficient . . . 80
8.10. A cognitive offload index . . . 80
8.11. Instrumenting AI review interactions . . . 81
8.12. Validating cognitive proxies . . . 81
8.12.1. Predictive validity . . . 81
8.12.2. Convergent validity . . . 81
8.12.3. Discriminant validity . . . 81
8.12.4. Experimental responsiveness . . . 82
8.12.5. Calibration studies . . . 82
8.13. Experiment designs . . . 82
8.14. Behavioral adaptation . . . 82
8.15. Ethical constraints . . . 83
8.16. Common failure modes . . . 83
8.16.1. Equating speed with lower cognitive burden . . . 83
8.16.2. Treating AI output volume as offload . . . 83
8.16.3. Using one universal cognitive score . . . 83
8.16.4. Ignoring learning . . . 83
8.16.5. Measuring individuals rather than systems . . . 83
8.17. Worked example: quantifying context-switch cost . . . 83
8.18. Key takeaways . . . 84
8.19. Open questions . . . 84
8.20. Implementation checklist . . . 84
8.20.1. Research foundations and source notes . . . 85
9. Measuring Engineering Systems . . . 91
9.1. Measurement as a decision discipline . . . 91
9.2. The measurement hierarchy . . . 92
9.3. Properties of a good metric . . . 94
9.3.1. Decision relevance . . . 94
9.3.2. Construct validity . . . 94
9.3.3. Reliability . . . 95
9.3.4. Sensitivity . . . 95
9.3.5. Robustness . . . 95
9.3.6. Explainability . . . 95
9.3.7. Stability . . . 95
9.3.8. Collection cost . . . 95
9.4. Metric taxonomy . . . 95
9.5. Direct and derived metrics . . . 96
9.6. Leading and lagging measures . . . 96
9.7. Local and system-level measures . . . 96
9.8. Composite metrics . . . 97
9.9. Guardrail metrics . . . 97
9.10. Statistical treatment . . . 97
9.10.1. Control charts . . . 97
9.10.2. Time-series effects . . . 98
9.10.3. Bayesian updating . . . 98
9.11. Metric versioning . . . 98

9.12. Comparison and normalization . . . 98
9.13. Measurement anti-patterns . . . 98
9.13.1. Activity as productivity . . . 98
9.13.2. Individual ranking . . . 99
9.13.3. One universal score . . . 99
9.13.4. Silent definition changes . . . 99
9.13.5. Correlation as causation . . . 99
9.13.6. Dashboard accumulation . . . 99
9.14. Designing a measurement catalogue . . . 99
9.15. Example: measuring review latency . . . 100
9.16. Example: measuring AI review quality . . . 100
9.17. Worked example: the cost of an unversioned metric change . . . 100
9.18. Key takeaways . . . 101
9.19. Open questions . . . 101
9.20. Implementation checklist . . . 101
9.20.1. Research foundations and source notes . . . 101
10. Experimentation and Causal Inference . . . 109
10.1. Why observation is not enough . . . 109
10.2. Define the intervention precisely . . . 110
10.3. Unit of assignment and unit of analysis . . . 111
10.4. Randomized controlled experiments . . . 111
10.5. Cluster-randomized experiments . . . 111
10.6. Switchback experiments . . . 112
10.7. Staggered rollout . . . 112
10.8. Difference-in-differences . . . 112
10.9. Interrupted time series . . . 112
10.10. Synthetic controls and matched comparisons . . . 112
10.11. Regression discontinuity . . . 113
10.12. Primary outcomes and guardrails . . . 113
10.13. Effect size and practical significance . . . 113
10.14. Power and sample size . . . 113
10.15. Sequential testing . . . 114
10.16. Multiple comparisons . . . 114
10.17. Treatment exposure and compliance . . . 114
10.18. Interference and network effects . . . 115
10.19. Heterogeneous treatment effects . . . 115
10.20. External factors . . . 115
10.21. Experiment registry . . . 116
10.22. Experiment infrastructure . . . 116
10.23. Example experiment: AI comment prioritization . . . 117
10.24. Example experiment: specialist review routing . . . 117
10.25. Common failure modes . . . 117
10.25.1. Before-and-after attribution . . . 117
10.25.2. Self-selection bias . . . 117
10.25.3. Underpowered rare outcomes . . . 118
10.25.4. Metric substitution . . . 118
10.25.5. Peeking and optional stopping . . . 118
10.25.6. Ignoring noncompliance . . . 118
10.25.7. Global averages . . . 118
10.25.8. Unrecorded concurrent interventions . . . 118
10.26. Worked example: did the AI reviewer actually reduce review latency? . . . 118

10.27. Key takeaways . . . 119
10.28. Open questions . . . 119
10.29. Implementation checklist . . . 119
10.29.1. Research foundations and source notes . . . 119
11. AI-Assisted Engineering . . . 129
11.1. From tool to participant . . . 129
11.2. Levels of AI participation . . . 130
11.3. AI contribution records . . . 130
11.4. Attribution is a graph problem . . . 131
11.5. Evaluating AI code generation . . . 131
11.6. Evaluating AI review . . . 131
11.7. Ground truth and golden data . . . 132
11.8. Human validation burden . . . 132
11.9. Trust calibration . . . 132
11.10. Cognitive offload and reload . . . 133
11.11. Quality-adjusted AI leverage . . . 133
11.12. Agents and multi-step tasks . . . 133
11.13. Autonomy and risk tiers . . . 136
11.14. Model, prompt, and policy drift . . . 136
11.15. Human learning and skill . . . 136
11.16. Governance and security . . . 136
11.17. Experiment patterns . . . 137
11.18. Common failure modes . . . 137
11.18.1. Counting generated code as productivity . . . 137
11.18.2. Treating acceptance as correctness . . . 137
11.18.3. Ignoring human validation . . . 137
11.18.4. Evaluating only synthetic benchmarks . . . 137
11.18.5. Global average performance . . . 137
11.18.6. Silent model changes . . . 138
11.18.7. Automating beyond evidence . . . 138
11.19. Worked example: measuring net cognitive offload from an AI reviewer . . . 138
11.20. Key takeaways . . . 138
11.21. Open questions . . . 139
11.22. Implementation checklist . . . 139
11.22.1. Research foundations and source notes . . . 139
12. Enterprise Instrumentation . . . 149
12.1. Instrument the system, not the dashboard . . . 149
12.2. Source domains . . . 150
12.3. Canonical event envelope . . . 150
12.4. Event time, ingestion time, and processing time . . . 151
12.5. Delivery semantics and idempotency . . . 151
12.6. Immutable raw evidence . . . 151
12.7. Schema management . . . 152
12.8. Identity resolution . . . 152
12.9. Human identity and privacy . . . 153
12.10. Stream processing responsibilities . . . 153
12.11. Batch processing responsibilities . . . 153
12.12. Lambda, kappa, and hybrid designs . . . 154
12.13. Engineering Change reconstruction service . . . 154
12.14. Engineering knowledge graph . . . 154

12.15. Feature store . . . 154
12.16. Metric engine . . . 155
12.17. Experiment data plane . . . 155
12.18. Data quality . . . 155
12.19. Lineage and reproducibility . . . 156
12.20. Storage layers . . . 156
12.21. Retention and deletion . . . 157
12.22. Security model . . . 157
12.23. Operational reliability . . . 157
12.24. Build versus buy . . . 157
12.25. Incremental adoption . . . 157
12.26. Common failure modes . . . 158
12.26.1. Dashboard-first architecture . . . 158
12.26.2. Centralizing transformed data without raw evidence . . . 158
12.26.3. Ignoring temporal identity . . . 158
12.26.4. Mixing event and processing time . . . 158
12.26.5. Unversioned schemas and metrics . . . 158
12.26.6. Collecting excessive behavioral telemetry . . . 158
12.26.7. Treating one source as universal truth . . . 158
12.27. Worked example: measuring identity-resolution accuracy . . . 158
12.28. Key takeaways . . . 159
12.29. Open questions . . . 159
12.30. Implementation checklist . . . 159
12.30.1. Research foundations and source notes . . . 160
13. Complexity, Blast Radius, and Repository Topology . . . 171
13.1. Why size is not complexity . . . 171
13.2. Complexity dimensions . . . 172
13.3. Implementation complexity . . . 172
13.4. Domain complexity . . . 172
13.5. Architectural complexity . . . 173
13.6. Verification complexity . . . 173
13.7. Organizational complexity . . . 173
13.8. Novelty . . . 174
13.9. Repository topology . . . 174
13.10. Multi-repository change graphs . . . 175
13.11. Deployment topology . . . 175
13.12. Blast radius . . . 176
13.13. A blast-radius score . . . 176
13.14. Reversibility . . . 177
13.15. Change surface versus impact surface . . . 177
13.16. Measuring dependency reach . . . 177
13.17. Complexity-adjusted comparisons . . . 178
13.18. Complexity and review routing . . . 178
13.19. AI and complexity . . . 178
13.20. Estimation methods . . . 179
13.21. Validation . . . 179
13.22. Common failure modes . . . 179
13.22.1. Lines changed as complexity . . . 179
13.22.2. Repository count as work count . . . 179
13.22.3. One universal complexity score . . . 179
13.22.4. Static dependencies as complete reach . . . 179

13.22.5. Realized incidents as blast radius . . . 179
13.22.6. Ignoring reversibility . . . 180
13.23. Worked example: scoring the blast radius of a shared-auth library change . . . 180
13.24. Key takeaways . . . 180
13.25. Open questions . . . 180
13.26. Implementation checklist . . . 181
13.26.1. Research foundations and source notes . . . 181
14. The Engineering Intelligence Platform . . . 191
14.1. Platform objectives . . . 191
14.2. Core architectural principles . . . 192
14.3. Logical architecture . . . 192
14.4. Service boundaries . . . 194
14.4.1. Source connector service . . . 194
14.4.2. Identity service . . . 194
14.4.3. Engineering Change service . . . 194
14.4.4. Topology service . . . 194
14.4.5. Feature service . . . 194
14.4.6. Metric service . . . 194
14.4.7. Experiment service . . . 195
14.4.8. Model service . . . 195
14.4.9. Decision product service . . . 195
14.5. Canonical APIs . . . 195
14.6. Storage architecture . . . 195
14.7. Stream processing . . . 196
14.8. Batch and graph computation . . . 196
14.9. Metric specification language . . . 196
14.10. Policy and routing engine . . . 197
14.11. Decision products . . . 197
14.11.1. Engineer view . . . 197
14.11.2. Team view . . . 197
14.11.3. Platform view . . . 197
14.11.4. Executive view . . . 197
14.12. Data contracts and ownership . . . 198
14.13. Platform SLOs . . . 198
14.14. Security and privacy architecture . . . 198
14.15. Reference implementation roadmap . . . 199
14.15.1. Phase 1: evidence foundation . . . 199
14.15.2. Phase 2: Engineering Change graph . . . 199
14.15.3. Phase 3: governed metrics . . . 199
14.15.4. Phase 4: experimentation . . . 199
14.15.5. Phase 5: AI and cognitive instrumentation . . . 199
14.15.6. Phase 6: predictive and prescriptive intelligence . . . 199
14.16. Suggested technology-neutral components . . . 199
14.17. Build-versus-buy decisions . . . 200
14.18. Operating model . . . 200
14.19. Common failure modes . . . 200
14.19.1. Building a universal data lake without decisions . . . 200
14.19.2. Coupling metrics to dashboards . . . 200
14.19.3. Hiding provenance . . . 200
14.19.4. Centralizing all ownership . . . 200
14.19.5. Automating policy too early . . . 200

14.19.6. Ignoring operating model 201
14.20. Worked example: sequencing platform delivery for first-year value 201
14.21. Key takeaways 201
14.22. Open questions 201
14.23. Implementation checklist 202
14.23.1. Research foundations and source notes 202
15. The Engineering Metrics Catalogue 209
15.1. Why a catalogue matters 209
15.2. Metric specification template 210
15.3. Flow metrics 211
15.3.1. Engineering Change lead time 211
15.3.2. Active work time 211
15.3.3. Queue time 211
15.3.4. Flow efficiency 211
15.3.5. Work in process 211
15.3.6. Aging work 211
15.4. Review metrics 212
15.4.1. Time to first substantive review 212
15.4.2. Active review effort 212
15.4.3. Review response cycles 212
15.4.4. Review queue age 212
15.4.5. Review Friction Index 212
15.5. Quality metrics 212
15.5.1. Change failure rate 212
15.5.2. Escaped defect rate 212
15.5.3. Severity-weighted defect cost 212
15.5.4. Rework ratio 213
15.5.5. Detection stage distribution 213
15.6. Economic metrics 213
15.6.1. Total Engineering Change cost 213
15.6.2. Cost per accepted outcome 213
15.6.3. Cost of delay 213
15.6.4. Scarce-attention consumption 213
15.6.5. Tool net value 213
15.7. Human and cognitive metrics 214
15.7.1. Context recovery time 214
15.7.2. Concurrent cognitive inventory 214
15.7.3. Cognitive Offload Index 214
15.7.4. Cognitive reload 214
15.7.5. Attention fragmentation 214
15.8. AI metrics 214
15.8.1. Finding precision and recall 214
15.8.2. Precision at k 214
15.8.3. NDCG 214
15.8.4. Duplicate finding rate 215
15.8.5. Human validation burden 215
15.8.6. Action rate 215
15.8.7. AI contribution latency 215
15.8.8. AI Leverage Factor 215
15.9. Architecture and topology metrics 215
15.9.1. Repository spread 215

15.9.2. Dependency depth 215
15.9.3. Critical-path duration 215
15.9.4. Ownership boundary count 215
15.9.5. Blast Radius Score 215
15.9.6. Change surface to impact surface ratio 215
15.10. Experiment metrics 216
15.10.1. Treatment exposure rate 216
15.10.2. Compliance rate 216
15.10.3. Average treatment effect 216
15.10.4. Minimum practical effect attainment 216
15.10.5. Guardrail breach rate 216
15.10.6. Experiment decision latency 216
15.11. Data quality metrics 216
15.11.1. Event completeness 216
15.11.2. Linkage coverage 216
15.11.3. Identity ambiguity rate 216
15.11.4. Freshness 217
15.11.5. Replay divergence 217
15.12. Metric bundles 217
15.12.1. AI review bundle 217
15.12.2. Flow bundle 217
15.12.3. Platform bundle 217
15.13. Metric lifecycle 218
15.14. Review and retirement 218
15.15. Common failure modes 218
15.15.1. Name without definition 218
15.15.2. Formula without construct 218
15.15.3. Metric without decision 218
15.15.4. Score without components 218
15.15.5. Dollar value without assumptions 219
15.16. Worked example: reading the flow bundle without misreading it 219
15.17. Key takeaways 219
15.18. Open questions 219
15.19. Implementation checklist 220
15.19.1. Research foundations and source notes 220
16. Experiment Playbooks 229
16.1. How to use a playbook 229
16.2. Playbook 1: AI pull-request reviewer 230
16.2.1. Decision 230
16.2.2. Mechanism 230
16.2.3. Treatment 230
16.2.4. Assignment 230
16.2.5. Primary outcomes 230
16.2.6. Guardrails 230
16.2.7. Instrumentation 231
16.2.8. Analysis 231
16.2.9. Decision rule 231
16.3. Playbook 2: ranked versus unranked AI findings 231
16.3.1. Hypothesis 231
16.3.2. Treatment 231
16.3.3. Control 231

16.3.4. Primary outcomes . . . 231
16.3.5. Risks . . . 231
16.4. Playbook 3: merge queue introduction . . . 232
16.4.1. Decision . . . 232
16.4.2. Mechanism . . . 232
16.4.3. Assignment . . . 232
16.4.4. Primary outcome . . . 232
16.4.5. Guardrails . . . 232
16.4.6. External factors . . . 232
16.4.7. Analysis . . . 232
16.5. Playbook 4: CI acceleration . . . 232
16.5.1. Decision . . . 232
16.5.2. Treatments . . . 232
16.5.3. Primary outcome . . . 233
16.5.4. Economic model . . . 233
16.5.5. Guardrails . . . 233
16.5.6. Assignment . . . 233
16.5.7. Interpretation risk . . . 233
16.6. Playbook 5: risk-based specialist review . . . 233
16.6.1. Decision . . . 233
16.6.2. Treatment . . . 233
16.6.3. Control . . . 233
16.6.4. Primary outcome . . . 233
16.6.5. Guardrails . . . 233
16.6.6. Design . . . 234
16.7. Playbook 6: pull-request size guidance . . . 234
16.7.1. Decision . . . 234
16.7.2. Mechanism . . . 234
16.7.3. Treatment . . . 234
16.7.4. Unit of analysis . . . 234
16.7.5. Primary outcomes . . . 234
16.7.6. Heterogeneous effects . . . 234
16.7.7. Decision rule . . . 234
16.8. Playbook 7: monorepo or repository consolidation . . . 235
16.8.1. Decision . . . 235
16.8.2. Intervention . . . 235
16.8.3. Design . . . 235
16.8.4. Primary outcomes . . . 235
16.8.5. Guardrails . . . 235
16.8.6. External factors . . . 235
16.9. Playbook 8: automated context restoration . . . 235
16.9.1. Hypothesis . . . 235
16.9.2. Treatment . . . 236
16.9.3. Control . . . 236
16.9.4. Eligibility . . . 236
16.9.5. Primary outcome . . . 236
16.9.6. Guardrails . . . 236
16.9.7. Instrumentation . . . 236
16.10. Playbook 9: human-first versus AI-first review . . . 236
16.10.1. Decision . . . 236
16.10.2. Mechanisms . . . 236

16.10.3. Treatment arms 236
16.10.4. Outcomes 237
16.10.5. Design 237
16.11. Playbook 10: autonomous maintenance agent 237
16.11.1. Decision 237
16.11.2. Eligible tasks 237
16.11.3. Treatment 237
16.11.4. Control 237
16.11.5. Primary outcome 237
16.11.6. Guardrails 237
16.11.7. Attribution 238
16.12. Playbook 11: branch-protection policy 238
16.12.1. Decision 238
16.12.2. Intervention 238
16.12.3. Primary outcome 238
16.12.4. Design 238
16.12.5. Guardrails 238
16.13. Playbook 12: review-queue prioritization 238
16.13.1. Hypothesis 238
16.13.2. Treatment 238
16.13.3. Control 238
16.13.4. Outcomes 239
16.13.5. Design 239
16.14. External-factor control checklist 239
16.15. Pre-experiment readiness 239
16.16. Post-experiment decision record 240
16.17. Common failure modes 240
16.17.1. Evaluating the wrong unit 240
16.17.2. Changing several mechanisms at once 240
16.17.3. Missing exposure 240
16.17.4. Short observation windows 240
16.17.5. Ignoring negative or subgroup effects 240
16.17.6. No decision threshold 240
16.18. Worked example: applying the AI-review playbook 240
16.19. Key takeaways 241
16.20. Open questions 241
16.21. Implementation checklist 241
16.21.1. Research foundations and source notes 242
17. Governance, Ethics, and Organizational Adoption 247
17.1. Measurement changes behavior 247
17.2. Goodhart’s Law and metric corruption 248
17.3. Foundational governance principles 248
17.3.1. Purpose limitation 248
17.3.2. Data minimization 248
17.3.3. Proportionality 248
17.3.4. Transparency 248
17.3.5. Contestability 249
17.3.6. Separation of purposes 249
17.3.7. Human accountability 249
17.4. Acceptable and prohibited uses 249
17.5. Individual-level measurement 249

17.6. Metric rights . . . 250
17.7. Privacy by architecture . . . 250
17.8. Data retention . . . 250
17.9. Model governance . . . 251
17.10. Fairness and disparate impact . . . 251
17.11. Incentive-aware metric design . . . 251
17.12. Governance bodies . . . 252
17.12.1. Metric council . . . 252
17.12.2. Experiment review . . . 252
17.12.3. Data governance . . . 252
17.12.4. AI and model governance . . . 252
17.12.5. Employee or practitioner representation . . . 252
17.13. Organizational adoption . . . 252
17.14. Trust as a system dependency . . . 253
17.15. Communication patterns . . . 253
17.16. Governance for experiments . . . 253
17.17. Governance for AI agents . . . 254
17.18. Handling metric disputes . . . 254
17.19. Change management . . . 254
17.20. Auditability . . . 254
17.21. Adoption maturity model . . . 255
17.22. Common failure modes . . . 255
17.22.1. Starting with executive ranking dashboards . . . 255
17.22.2. Policy without architecture . . . 255
17.22.3. Consent theater . . . 256
17.22.4. Treating fairness as equal averages . . . 256
17.22.5. Permanent experimental telemetry . . . 256
17.22.6. Governance only after harm . . . 256
17.23. Worked example: responding to a metric-misuse incident . . . 256
17.24. Key takeaways . . . 256
17.25. Open questions . . . 257
17.26. Implementation checklist . . . 257
17.26.1. Research foundations and source notes . . . 257
18. Portfolio and Executive Decision-Making . . . 265
18.1. From measurement to allocation . . . 265
18.2. Portfolio value model . . . 266
18.3. Capacity is multidimensional . . . 266
18.4. Opportunity cost . . . 266
18.5. Mandatory and strategic work . . . 267
18.6. Option value . . . 267
18.7. Value of information . . . 267
18.8. Dependency-aware portfolios . . . 267
18.9. Cost of delay at portfolio level . . . 268
18.10. Uncertainty and distributions . . . 268
18.11. Scenario planning . . . 269
18.12. Monte Carlo portfolio simulation . . . 269
18.13. Risk appetite and constraints . . . 269
18.14. Portfolio balance . . . 270
18.15. Platform investment economics . . . 270
18.16. Tool portfolio decisions . . . 270
18.17. Executive scorecards . . . 271

18.18. Avoiding the one-score trap . . . 271
18.19. Decision records . . . 271
18.20. Forecast calibration . . . 272
18.21. Portfolio review cadence . . . 272
18.22. Common failure modes . . . 272
18.22.1. Ranking by ROI alone . . . 272
18.22.2. Treating capacity as fungible . . . 272
18.22.3. Double counting value . . . 272
18.22.4. Ignoring uncertainty . . . 273
18.22.5. Portfolio theater . . . 273
18.22.6. Punishing forecast error . . . 273
18.23. Worked example: allocating capacity across three initiatives . . . 273
18.24. Key takeaways . . . 273
18.25. Open questions . . . 274
18.26. Implementation checklist . . . 274
18.26.1. Research foundations and source notes . . . 274
19. Implementation and Rollout . . . 283
19.1. Start with decisions, not completeness . . . 283
19.2. Pilot selection criteria . . . 284
19.3. The minimum viable evidence path . . . 284
19.4. Phase 0: charter and safeguards . . . 284
19.5. Phase 1: evidence foundation . . . 285
19.6. Phase 2: Engineering Change reconstruction . . . 285
19.7. Phase 3: governed metrics . . . 285
19.8. Phase 4: experimentation . . . 286
19.9. Phase 5: operational decision products . . . 286
19.10. Phase 6: scale and federation . . . 286
19.11. Team structure . . . 287
19.12. Product management for measurement . . . 287
19.13. Proving value . . . 287
19.14. Baseline before intervention . . . 288
19.15. Data quality gates . . . 288
19.16. Onboarding new domains . . . 288
19.17. Shadow mode . . . 289
19.18. Rollout patterns . . . 289
19.19. Change communication . . . 289
19.20. Operating cadence . . . 289
19.20.1. Weekly operational review . . . 290
19.20.2. Monthly metric review . . . 290
19.20.3. Quarterly governance review . . . 290
19.20.4. Experiment decision reviews . . . 290
19.21. Scaling source integrations . . . 290
19.22. Build economics . . . 290
19.23. Failure recovery . . . 291
19.24. Anti-patterns . . . 291
19.24.1. Enterprise-wide launch first . . . 291
19.24.2. Data platform as the product . . . 291
19.24.3. Dashboard proliferation . . . 291
19.24.4. Hidden quality problems . . . 291
19.24.5. No local ownership . . . 291
19.24.6. Scaling before trust . . . 291

19.25. A ninety-day pilot outline . . . 291
19.25.1. Days 1–30 . . . 291
19.25.2. Days 31–60 . . . 292
19.25.3. Days 61–90 . . . 292
19.26. Worked example: selecting the first pilot . . . 292
19.27. Key takeaways . . . 293
19.28. Open questions . . . 293
19.29. Implementation checklist . . . 293
19.29.1. Research foundations and source notes . . . 293
20. The Science of Engineering Intelligence . . . 299
20.1. From analytics practice to scientific discipline . . . 299
20.2. The object of study . . . 300
20.3. Levels of explanation . . . 300
20.3.1. Artifact level . . . 300
20.3.2. Change level . . . 300
20.3.3. Queue and workflow level . . . 300
20.3.4. Team level . . . 300
20.3.5. Architecture level . . . 301
20.3.6. Portfolio level . . . 301
20.3.7. Enterprise level . . . 301
20.4. Constructs and observables . . . 301
20.5. A theory-to-evidence chain . . . 301
20.6. Falsifiable hypotheses . . . 302
20.7. Descriptive, predictive, causal, and prescriptive claims . . . 302
20.7.1. Descriptive . . . 302
20.7.2. Predictive . . . 303
20.7.3. Causal . . . 303
20.7.4. Prescriptive . . . 303
20.8. Measurement validity . . . 303
20.8.1. Construct validity . . . 303
20.8.2. Content validity . . . 303
20.8.3. Criterion validity . . . 303
20.8.4. Convergent validity . . . 303
20.8.5. Discriminant validity . . . 303
20.8.6. Ecological validity . . . 303
20.9. Reliability and repeatability . . . 304
20.10. Measurement invariance . . . 304
20.11. Calibration . . . 304
20.12. Ground truth is constructed . . . 305
20.13. Annotation science . . . 305
20.14. Benchmark design . . . 305
20.15. Retrieval and ranking evaluation . . . 306
20.16. Economic evaluation of models . . . 306
20.17. Internal replication . . . 306
20.18. External replication . . . 307
20.19. Reproducibility packages . . . 307
20.20. Pre-registration . . . 307
20.21. Registered reports . . . 308
20.22. Negative results . . . 308
20.23. Meta-analysis . . . 308
20.24. Transportability . . . 309

20.25. Natural experiments 309
20.26. Longitudinal research 309
20.27. System adaptation 310
20.28. Multi-method research 310
20.29. Shared research infrastructure 310
20.30. Privacy-preserving collaboration 311
20.31. Evidence maturity levels 311
20.31.1. Level 0: anecdote 311
20.31.2. Level 1: descriptive evidence 311
20.31.3. Level 2: replicated association 311
20.31.4. Level 3: credible causal evidence 311
20.31.5. Level 4: replicated causal mechanism 311
20.31.6. Level 5: operationally validated policy 311
20.32. Scientific reporting standard 312
20.33. An open evidence graph 312
20.34. Research agenda 313
20.34.1. Engineering Change reconstruction 313
20.34.2. Cognitive load 313
20.34.3. AI collaboration 313
20.34.4. Review science 313
20.34.5. Architecture and organizational structure 313
20.34.6. Value attribution 313
20.34.7. Metric governance 313
20.34.8. Long-run capability 313
20.35. Institutional design for research 313
20.36. Anti-patterns 314
20.36.1. Dashboard empiricism 314
20.36.2. Metric essentialism 314
20.36.3. One-company universalism 314
20.36.4. Benchmark overfitting 314
20.36.5. Causal language without causal design 314
20.36.6. Hidden researcher degrees of freedom 314
20.36.7. Publication of winners only 314
20.36.8. Permanent validity 314
20.37. Key takeaways 314
20.38. Open questions 315
20.39. Scientific practice checklist 315
20.39.1. Research foundations and source notes 315
21. The Learning Engineering Organization 323
21.1. Measurement is not the destination 323
21.2. The organizational learning loop 324
21.3. Observation 324
21.4. Explanation 324
21.5. Intervention 325
21.6. Evaluation 325
21.7. Decision 326
21.8. Institutional memory 326
21.9. The five capabilities 326
21.9.1. Evidence capability 327
21.9.2. Experimental capability 327
21.9.3. Decision capability 327

21.9.4. Adaptation capability . . . 327
21.9.5. Memory capability . . . 327
21.10. Local autonomy and shared foundations . . . 327
21.11. Leadership behavior . . . 328
21.12. Practitioner participation . . . 328
21.13. Learning from incidents . . . 328
21.14. Learning from successful changes . . . 329
21.15. Learning across teams . . . 329
21.16. Learning across time horizons . . . 329
21.16.1. Immediate . . . 329
21.16.2. Tactical . . . 330
21.16.3. Strategic . . . 330
21.16.4. Institutional . . . 330
21.17. AI as participant and subject . . . 330
21.18. The changing role of the engineer . . . 330
21.19. Capability preservation . . . 331
21.20. Incentives and truth . . . 331
21.21. Economic discipline . . . 331
21.22. Architecture as accumulated decision history . . . 332
21.23. Learning portfolio . . . 332
21.24. Learning velocity . . . 333
21.25. Signs of maturity . . . 333
21.26. Signs of failure . . . 333
21.27. A practical operating cadence . . . 334
21.27.1. Weekly system review . . . 334
21.27.2. Monthly learning review . . . 334
21.27.3. Quarterly portfolio review . . . 334
21.27.4. Semiannual evidence synthesis . . . 334
21.27.5. Annual capability review . . . 334
21.28. The role of an Engineering Intelligence function . . . 334
21.29. A north-star architecture . . . 335
21.30. A north-star culture . . . 335
21.31. The durable advantage . . . 335
21.32. Key takeaways . . . 336
21.33. Open questions . . . 336
21.34. Learning organization checklist . . . 336
21.34.1. Research foundations and source notes . . . 337
22. The Future of Engineering Intelligence . . . 345
22.1. A changing production system . . . 345
22.2. Intent becomes a first-class artifact . . . 346
22.3. From Engineering Change to change lineage . . . 346
22.4. AI agents as economic actors . . . 347
22.5. Agent identity and accountability . . . 347
22.6. Delegation graphs . . . 348
22.7. Machine-generated activity inflation . . . 348
22.8. Scarcity moves to validation . . . 348
22.9. Review becomes orchestration . . . 349
22.10. Competing and cooperating models . . . 349
22.11. Model monoculture risk . . . 350
22.12. Continuous evaluation . . . 350
22.13. Dynamic treatment systems . . . 350

22.14. Personalized engineering assistance . . . 351
22.15. Skill development and skill atrophy . . . 351
22.16. Software architecture generated at scale . . . 352
22.17. Architecture policy as executable evidence . . . 352
22.18. Simulation before deployment . . . 352
22.19. Digital twins of engineering systems . . . 353
22.20. Prescriptive systems . . . 353
22.21. Closed-loop automation . . . 353
22.22. Human control is not a binary . . . 354
22.23. Governance at machine speed . . . 354
22.24. Privacy under richer instrumentation . . . 355
22.25. Security of the evidence system . . . 355
22.26. Adversarial behavior . . . 355
22.27. Synthetic evidence risk . . . 356
22.28. Open standards . . . 356
22.29. Vendor-neutral evidence . . . 356
22.30. Engineering Intelligence as infrastructure . . . 357
22.31. What should remain stable . . . 357
22.31.1. Measure outcomes, not artifact volume . . . 357
22.31.2. Treat attention as scarce . . . 357
22.31.3. Preserve the Engineering Change as a unit of consequence . . . 357
22.31.4. Separate observation from inference . . . 357
22.31.5. Require lineage . . . 358
22.31.6. Evaluate interventions causally . . . 358
22.31.7. Include economics . . . 358
22.31.8. Govern use, not only collection . . . 358
22.31.9. Expect adaptation . . . 358
22.31.10. Preserve human challenge . . . 358
22.32. Near-term priorities . . . 358
22.33. A staged future architecture . . . 358
22.33.1. Stage 1: observable workflows . . . 359
22.33.2. Stage 2: reconstructed changes . . . 359
22.33.3. Stage 3: experimental organization . . . 359
22.33.4. Stage 4: human-machine production system . . . 359
22.33.5. Stage 5: prescriptive intelligence . . . 359
22.33.6. Stage 6: bounded autonomous optimization . . . 359
22.34. Failure futures . . . 359
22.34.1. The surveillance enterprise . . . 359
22.34.2. The automation illusion . . . 359
22.34.3. The model monoculture . . . 359
22.34.4. The unaccountable agent mesh . . . 359
22.34.5. The synthetic management system . . . 360
22.34.6. The permanently experimental organization . . . 360
22.35. A desirable future . . . 360
22.36. Final synthesis . . . 360
22.37. Key takeaways . . . 361
22.38. Open questions . . . 361
22.39. Future-readiness checklist . . . 361
22.39.1. Research foundations and source notes . . . 362
Appendix A: Engineering Change Reference Model . . . 369
A.1. Purpose . . . 369

A.2. Design principles . . . 369
A.2.1. Tool independence . . . 369
A.2.2. Event-derived state . . . 369
A.2.3. Temporal correctness . . . 369
A.2.4. Explicit uncertainty . . . 370
A.2.5. Many-to-many reality . . . 370
A.2.6. Human and machine participation . . . 370
A.2.7. Outcome orientation . . . 370
A.3. Core entity model . . . 370
A.3.1. EngineeringChange . . . 370
A.3.2. Intent . . . 371
A.3.3. Actor . . . 371
A.3.4. Artifact . . . 372
A.3.5. Activity . . . 372
A.3.6. Review . . . 373
A.3.7. Finding . . . 373
A.3.8. ValidationRun . . . 374
A.3.9. Build . . . 374
A.3.10. Deployment . . . 374
A.3.11. Release . . . 375
A.3.12. Outcome . . . 375
A.3.13. Incident . . . 375
A.3.14. Experiment . . . 376
A.3.15. Exposure . . . 376
A.3.16. Decision . . . 376
A.4. Relationship vocabulary . . . 377
A.5. Lifecycle model . . . 377
A.5.1. Proposed . . . 377
A.5.2. Active . . . 377
A.5.3. In validation . . . 378
A.5.4. Accepted . . . 378
A.5.5. Integrated . . . 378
A.5.6. Deployed . . . 378
A.5.7. Released . . . 378
A.5.8. Operating . . . 378
A.5.9. Completed . . . 378
A.5.10. Reverted . . . 378
A.5.11. Abandoned . . . 378
A.5.12. Superseded . . . 378
A.6. State transition rules . . . 378
A.7. Identity strategy . . . 379
A.7.1. Explicit identifiers . . . 379
A.7.2. Deterministic reconstruction . . . 379
A.7.3. Probabilistic reconstruction . . . 379
A.8. Link confidence . . . 380
A.9. Provenance model . . . 380
A.10. Temporal ownership . . . 381
A.11. Work decomposition and aggregation . . . 381
A.12. Change boundaries . . . 381
A.13. AI and agent provenance . . . 382
A.14. Correction model . . . 382

A.15. Deletion and retention . . . 383
A.16. Canonical event envelope . . . 383
A.17. Data quality dimensions . . . 383
A.17.1. Completeness . . . 384
A.17.2. Timeliness . . . 384
A.17.3. Uniqueness . . . 384
A.17.4. Consistency . . . 384
A.17.5. Validity . . . 384
A.17.6. Linkage coverage . . . 384
A.17.7. Linkage confidence . . . 384
A.17.8. Temporal correctness . . . 384
A.17.9. Provenance completeness . . . 384
A.18. Minimum viable implementation . . . 384
A.19. Storage patterns . . . 385
A.20. Query examples . . . 385
A.20.1. Changes released without confirmed validation . . . 385
A.20.2. Human validation burden by AI model version . . . 385
A.20.3. Deployment linkage quality . . . 386
A.21. Conformance levels . . . 386
A.21.1. Level 1: identifiable . . . 386
A.21.2. Level 2: traceable . . . 386
A.21.3. Level 3: measurable . . . 386
A.21.4. Level 4: experimental . . . 386
A.21.5. Level 5: adaptive . . . 386
A.22. Reference-model anti-patterns . . . 386
A.22.1. Pull request as the universal change . . . 386
A.22.2. Mutable current-state tables only . . . 386
A.22.3. One change, one deployment . . . 386
A.22.4. Current ownership applied historically . . . 387
A.22.5. Inferred links without confidence . . . 387
A.22.6. Machine actors collapsed into bot . . . 387
A.22.7. Outcome equals deployment success . . . 387
A.22.8. Silent correction . . . 387
A.23. Reference-model checklist . . . 387
Appendix B: Metric Specification Standard . . . 389
B.1. Why a metric standard is necessary . . . 389
B.2. Governing principle . . . 389
B.3. Metric specification lifecycle . . . 389
B.4. Required specification fields . . . 390
B.4.1. Metric identity . . . 390
B.4.2. Purpose . . . 390
B.4.3. Construct . . . 391
B.4.4. Claim type . . . 391
B.4.5. Unit of analysis . . . 391
B.4.6. Population . . . 391
B.4.7. Measurement interval . . . 392
B.4.8. Formula . . . 392
B.4.9. Numerator and denominator . . . 392
B.4.10. Aggregation . . . 393
B.4.11. Dimensions . . . 393
B.4.12. Source evidence . . . 394

B.4.13. Transformation lineage 394
B.4.14. Data-quality requirements 394
B.4.15. Missing-data policy 394
B.4.16. Bot and AI treatment 395
B.4.17. Complexity and work-mix adjustment 395
B.4.18. Uncertainty 395
B.4.19. Validity evidence 396
B.4.20. Reliability evidence 396
B.4.21. Interpretation 396
B.4.22. Prohibited interpretations 397
B.4.23. Guardrails 397
B.4.24. Privacy classification 397
B.4.25. Access policy 397
B.4.26. Behavioral risk 398
B.4.27. Decision thresholds 398
B.4.28. Versioning rules 398
B.4.29. Monitoring 398
B.4.30. Review and retirement 399
B.5. Complete metric template 399
B.6. Metric dependency graph 401
B.7. Metric bundles 401
B.8. Metric maturity levels 402
B.8.1. Level 0: draft 402
B.8.2. Level 1: defined 402
B.8.3. Level 2: validated 402
B.8.4. Level 3: governed 402
B.8.5. Level 4: decision-grade 402
B.8.6. Level 5: policy-grade 402
B.9. Validation process 402
B.10. Metric tests 403
B.11. Publication contract 403
B.12. Change management 404
B.13. Metric incident management 404
B.14. Example: deployment frequency 404
B.15. Example: AI acceptance rate 405
B.16. Example: review precision 405
B.17. Anti-patterns 405
B.17.1. Formula-only definition 405
B.17.2. Metric without population 405
B.17.3. Missing equals zero 406
B.17.4. Versionless dashboard 406
B.17.5. Universal threshold 406
B.17.6. Bare point estimate 406
B.17.7. Individual ranking by system telemetry 406
B.17.8. Metric without retirement 406
B.18. Metric specification checklist 406
Appendix C: Experiment and Causal Study Protocol 407
C.1. Scope 407
C.2. The experiment lifecycle 407
C.3. Start with a decision 407
C.4. Study charter 408

C.5. Causal question . . . 408
C.6. Hypothesis specification . . . 409
C.7. Theory of change . . . 409
C.8. Eligibility . . . 410
C.9. Unit of assignment . . . 410
C.10. Unit of analysis . . . 410
C.11. Treatment specification . . . 411
C.12. Comparator . . . 411
C.13. Randomization . . . 411
C.14. Stratification and blocking . . . 412
C.15. Cluster randomization . . . 412
C.16. Switchback experiments . . . 412
C.17. Stepped-wedge rollout . . . 413
C.18. Exposure . . . 413
C.19. Intention-to-treat and treatment-on-treated . . . 413
C.20. Noncompliance . . . 413
C.21. Interference . . . 414
C.22. Spillover measurement . . . 414
C.23. Primary outcome . . . 414
C.24. Secondary outcomes . . . 415
C.25. Guardrails . . . 415
C.26. Outcome windows . . . 415
C.27. Baseline period . . . 416
C.28. Sample-size and power planning . . . 416
C.29. Rare outcomes . . . 416
C.30. Noninferiority . . . 416
C.31. Economic outcomes . . . 417
C.32. External-factor registry . . . 417
C.33. Readiness review . . . 417
C.34. Pre-analysis plan . . . 418
C.35. Estimand . . . 418
C.36. Covariate adjustment . . . 418
C.37. Missing outcomes . . . 419
C.38. Sequential monitoring . . . 419
C.39. Stop rules . . . 419
C.40. Analysis principles . . . 420
C.41. Distributional effects . . . 420
C.42. Heterogeneous treatment effects . . . 420
C.43. Multiple comparisons . . . 421
C.44. Robustness checks . . . 421
C.45. Quasi-experimental studies . . . 421
C.45.1. Difference-in-differences . . . 421
C.45.2. Interrupted time series . . . 421
C.45.3. Regression discontinuity . . . 421
C.45.4. Instrumental variables . . . 421
C.45.5. Synthetic control . . . 422
C.45.6. Matching and weighting . . . 422
C.46. Decision framework . . . 422
C.47. Decision record . . . 422
C.48. Reporting package . . . 423
C.49. Reproducibility . . . 423

C.50. Follow-up . . . 423
C.51. Replication . . . 424
C.52. Study maturity levels . . . 424
C.52.1. Level 0: observational signal . . . 424
C.52.2. Level 1: designed observational study . . . 424
C.52.3. Level 2: credible quasi-experiment . . . 424
C.52.4. Level 3: randomized evaluation . . . 424
C.52.5. Level 4: replicated causal evidence . . . 424
C.52.6. Level 5: governed operational policy . . . 424
C.53. Anti-patterns . . . 424
C.53.1. Adoption as treatment . . . 425
C.53.2. Assignment without exposure . . . 425
C.53.3. Moving outcome definitions . . . 425
C.53.4. Success metric without guardrails . . . 425
C.53.5. Unit mismatch . . . 425
C.53.6. Post-treatment filtering . . . 425
C.53.7. Dashboard significance testing . . . 425
C.53.8. Rare-event safety claim . . . 425
C.53.9. Experiment without decision . . . 425
C.54. Protocol checklist . . . 425
Appendix D: Governance, Privacy, and Acceptable Use Standard . . . 427
D.1. Purpose . . . 427
D.2. Core principles . . . 427
D.2.1. Purpose limitation . . . 428
D.2.2. Data minimization . . . 428
D.2.3. Proportionality . . . 428
D.2.4. Transparency . . . 428
D.2.5. Contestability . . . 428
D.2.6. Human accountability . . . 428
D.2.7. Evidence quality . . . 428
D.2.8. Temporal relevance . . . 428
D.2.9. Separation of learning and punishment . . . 428
D.3. Governance operating model . . . 428
D.4. Roles and responsibilities . . . 431
D.4.1. Executive sponsor . . . 431
D.4.2. Engineering Intelligence owner . . . 431
D.4.3. Data owner . . . 431
D.4.4. Metric owner . . . 431
D.4.5. Study owner . . . 431
D.4.6. Domain representative . . . 431
D.4.7. Privacy reviewer . . . 431
D.4.8. Security reviewer . . . 431
D.4.9. Legal or compliance reviewer . . . 431
D.4.10. Practitioner representative . . . 431
D.4.11. Governance board . . . 431
D.5. Use classification . . . 432
D.5.1. Class 0: platform operations . . . 432
D.5.2. Class 1: descriptive team learning . . . 432
D.5.3. Class 2: organizational decision support . . . 432
D.5.4. Class 3: policy influence . . . 432
D.5.5. Class 4: individual or employment consequence . . . 433

D.6. Permitted uses . . . 433
D.7. Prohibited uses . . . 433
D.8. Restricted uses . . . 434
D.9. Purpose registration . . . 434
D.10. Approval criteria . . . 435
D.11. Data inventory . . . 435
D.12. Data classification . . . 435
D.12.1. Public . . . 436
D.12.2. Internal . . . 436
D.12.3. Confidential . . . 436
D.12.4. Restricted . . . 436
D.12.5. Highly restricted . . . 436
D.13. Sensitive data . . . 436
D.14. Data minimization review . . . 436
D.15. Identity handling . . . 437
D.16. Aggregation policy . . . 437
D.17. Re-identification risk . . . 437
D.18. Access-control model . . . 438
D.18.1. Raw source evidence . . . 438
D.18.2. Canonical row-level evidence . . . 438
D.18.3. Restricted analytical datasets . . . 438
D.18.4. Team aggregates . . . 438
D.18.5. Enterprise aggregates . . . 438
D.19. Purpose-based access . . . 438
D.20. Least privilege . . . 439
D.21. Separation of duties . . . 439
D.22. Data retention . . . 439
D.23. Deletion . . . 440
D.24. Export controls . . . 440
D.25. Vendor governance . . . 440
D.26. AI model governance . . . 441
D.27. Generated explanations . . . 441
D.28. Automated decisions . . . 442
D.28.1. Advisory . . . 442
D.28.2. Confirm-before-action . . . 442
D.28.3. Bounded autonomous . . . 442
D.28.4. High-impact autonomous . . . 442
D.29. Automation control requirements . . . 442
D.30. Individual-level views . . . 443
D.31. Practitioner notice . . . 443
D.32. Consent . . . 443
D.33. Correction rights . . . 444
D.34. Appeal process . . . 444
D.35. Fairness and subgroup review . . . 444
D.36. Distributional effects . . . 445
D.37. Metric-use review . . . 445
D.38. Experiment governance . . . 445
D.39. Research publication . . . 446
D.40. Monitoring approved uses . . . 446
D.41. Audit logging . . . 446
D.42. Periodic access review . . . 447

D.43. Governance incident . . . 447
D.44. Incident response . . . 447
D.45. Metric and model rollback . . . 448
D.46. Misuse detection . . . 448
D.47. Policy exception process . . . 448
D.48. Governance review cadence . . . 449
D.48.1. Weekly operational review . . . 449
D.48.2. Monthly use review . . . 449
D.48.3. Quarterly governance review . . . 449
D.48.4. Annual program review . . . 449
D.49. Governance maturity levels . . . 449
D.49.1. Level 0: implicit . . . 449
D.49.2. Level 1: documented . . . 449
D.49.3. Level 2: operational . . . 449
D.49.4. Level 3: auditable . . . 449
D.49.5. Level 4: adaptive . . . 450
D.49.6. Level 5: trustworthy autonomy . . . 450
D.50. Governance anti-patterns . . . 450
D.50.1. Collect first, justify later . . . 450
D.50.2. Access by seniority . . . 450
D.50.3. Aggregate means anonymous . . . 450
D.50.4. Metric governance without use governance . . . 450
D.50.5. Consent theater . . . 450
D.50.6. AI explanation as due process . . . 450
D.50.7. Permanent pilot . . . 450
D.50.8. Policy without enforcement . . . 450
D.50.9. Governance as delay only . . . 450
D.51. Standard implementation checklist . . . 451
Appendix E: Economic Evaluation and Business Case Standard . . . 453
E.1. Purpose . . . 453
E.2. Core economic model . . . 453
E.3. Economic evaluation principles . . . 454
E.3.1. Compare alternatives . . . 454
E.3.2. Count net effects . . . 454
E.3.3. Distinguish capacity from cash . . . 454
E.3.4. Value outcomes, not activity . . . 454
E.3.5. Include uncertainty . . . 454
E.3.6. Track realization . . . 454
E.3.7. Include distribution . . . 454
E.4. Business-case identity . . . 455
E.5. Define the counterfactual . . . 455
E.6. Cost taxonomy . . . 456
E.6.1. Acquisition cost . . . 456
E.6.2. Implementation cost . . . 456
E.6.3. Operating cost . . . 456
E.6.4. Human attention cost . . . 456
E.6.5. Switching cost . . . 456
E.6.6. Risk cost . . . 456
E.6.7. Opportunity cost . . . 456
E.6.8. Residual and terminal cost . . . 456
E.7. Benefit taxonomy . . . 456

E.7.1. Direct cash savings 456
E.7.2. Avoided cost 457
E.7.3. Released capacity 457
E.7.4. Throughput value 457
E.7.5. Cost-of-delay reduction 457
E.7.6. Quality and reliability value 457
E.7.7. Risk reduction 457
E.7.8. Option value 457
E.7.9. Capability value 457
E.8. Labor cost model 458
E.9. Capacity realization 458
E.10. Fragmentation discount 458
E.11. Bottleneck adjustment 459
E.12. Queueing effects 459
E.13. Cost of delay 459
E.14. Quality economics 460
E.15. Risk-adjusted value 460
E.16. AI tool economics 460
E.17. Validation burden 461
E.18. Vendor pricing models 461
E.19. Build versus buy 462
E.20. Architecture investment economics 462
E.21. Platform economics 463
E.22. Allocation of shared costs 463
E.23. Sunk costs 464
E.24. Evaluation horizon 464
E.25. Discounting 464
E.26. Standard financial outputs 464
E.27. Break-even analysis 465
E.28. Scenario analysis 465
E.29. Sensitivity analysis 465
E.30. Monte Carlo analysis 466
E.31. Evidence strength 466
E.32. Attribution 466
E.33. Double-counting controls 467
E.34. Capacity, savings, and value labels 467
E.34.1. Realized cash impact 467
E.34.2. Contracted or committed impact 467
E.34.3. Avoided planned cost 467
E.34.4. Released capacity 467
E.34.5. Modeled outcome value 467
E.34.6. Strategic or option value 467
E.35. Value-realization plan 467
E.36. Realization ledger 468
E.37. Post-implementation review 468
E.38. Renewal economics 469
E.39. Portfolio allocation 469
E.40. Decision thresholds 470
E.41. Decision record 470
E.42. Business-case maturity levels 470
E.42.1. Level 0: narrative 470

E.42.2. Level 1: estimated . . . 470
E.42.3. Level 2: evidence-linked . . . 470
E.42.4. Level 3: decision-grade . . . 471
E.42.5. Level 4: realized . . . 471
E.42.6. Level 5: adaptive portfolio evidence . . . 471
E.43. Anti-patterns . . . 471
E.43.1. Salary multiplication as savings . . . 471
E.43.2. Vendor usage as value . . . 471
E.43.3. Gross benefit without validation cost . . . 471
E.43.4. Static counterfactual . . . 471
E.43.5. Best-case-only forecast . . . 471
E.43.6. Double-counted benefit . . . 471
E.43.7. Ignored bottleneck . . . 471
E.43.8. Ignored migration and exit . . . 471
E.43.9. Modeled value reported as realized savings . . . 472
E.43.10. No post-implementation review . . . 472
E.44. Economic evaluation checklist . . . 472
Appendix F: Technical Reference Architecture . . . 473
F.1. Architectural objective . . . 473
F.2. Architectural principles . . . 473
F.2.1. Immutable evidence first . . . 473
F.2.2. Canonical semantics . . . 473
F.2.3. Event time over ingestion time . . . 473
F.2.4. Temporal identity and ownership . . . 474
F.2.5. Confidence-aware linkage . . . 474
F.2.6. Version everything consequential . . . 474
F.2.7. Separate evidence, inference, and decision . . . 474
F.2.8. Purpose-bound access . . . 474
F.2.9. Progressive capability . . . 474
F.3. Logical architecture . . . 474
F.4. Source systems . . . 476
F.4.1. Planning and intent . . . 478
F.4.2. Source control and review . . . 478
F.4.3. Build and validation . . . 478
F.4.4. Delivery and operations . . . 478
F.4.5. Organization and economics . . . 478
F.4.6. AI and agent systems . . . 479
F.5. Collection patterns . . . 479
F.5.1. Webhooks . . . 479
F.5.2. Event streams . . . 479
F.5.3. Change-data capture . . . 479
F.5.4. Polling . . . 479
F.5.5. Batch export . . . 479
F.5.6. Client-side instrumentation . . . 479
F.6. Connector contract . . . 480
F.7. Canonical event envelope . . . 480
F.8. Ingestion guarantees . . . 481
F.9. Immutable raw evidence layer . . . 481
F.10. Schema registry . . . 481
F.11. Canonicalization . . . 482
F.12. Data contracts . . . 482

F.13. Identity resolution service . . . 482
F.14. Temporal organizational model . . . 484
F.15. Service and repository catalogue . . . 484
F.16. Engineering Change reconstruction . . . 485
F.17. Reconstruction pipeline . . . 485
F.18. Candidate generation . . . 485
F.19. Confidence service . . . 486
F.20. Lifecycle state projection . . . 486
F.21. Analytical storage . . . 486
F.21.1. Lakehouse or warehouse . . . 486
F.21.2. Graph projection . . . 487
F.21.3. Search index . . . 487
F.21.4. Time-series store . . . 487
F.21.5. Key-value or operational database . . . 487
F.22. Fact and dimension design . . . 487
F.23. Metric computation layer . . . 487
F.24. Metric registry . . . 488
F.25. Feature and model layer . . . 488
F.26. Experiment assignment service . . . 489
F.27. Exposure service . . . 489
F.28. Guardrail monitoring . . . 489
F.29. Decision registry . . . 490
F.30. Evidence registry . . . 490
F.31. Serving layer . . . 491
F.31.1. Metric API . . . 493
F.31.2. Engineering Change API . . . 493
F.31.3. Evidence explorer . . . 493
F.31.4. Experiment console . . . 493
F.31.5. Decision workbench . . . 493
F.31.6. Practitioner views . . . 493
F.31.7. Executive products . . . 493
F.32. API design . . . 493
F.33. Data lineage . . . 494
F.34. Reproducibility . . . 494
F.35. Privacy architecture . . . 494
F.36. Security architecture . . . 495
F.37. Multi-tenancy . . . 495
F.38. Regional and regulatory architecture . . . 496
F.39. Reliability objectives . . . 496
F.40. Failure modes . . . 496
F.41. Backfill and replay . . . 497
F.42. Correction workflow . . . 497
F.43. Observability of the intelligence platform . . . 497
F.44. Quality SLOs . . . 498
F.45. Cost architecture . . . 498
F.46. Technology selection criteria . . . 499
F.47. Deployment topology . . . 499
F.48. Environment strategy . . . 499
F.49. Testing strategy . . . 500
F.49.1. Connector tests . . . 500
F.49.2. Contract tests . . . 500

F.49.3. Reconstruction tests 500
F.49.4. Metric tests 500
F.49.5. Experiment tests 501
F.49.6. Security and privacy tests 501
F.50. Release management 501
F.51. Disaster recovery 501
F.52. Staged implementation 502
F.52.1. Stage 1: evidence foundation 502
F.52.2. Stage 2: Engineering Change graph 502
F.52.3. Stage 3: governed metric products 502
F.52.4. Stage 4: experimentation 502
F.52.5. Stage 5: economics and portfolio 502
F.52.6. Stage 6: AI and agent intelligence 502
F.52.7. Stage 7: bounded adaptive policy 502
F.53. Minimum viable platform 502
F.54. Build sequence 503
F.55. Team topology 503
F.56. Ownership boundaries 504
F.57. Architecture decision records 504
F.58. Technical anti-patterns 504
F.58.1. Dashboard-first architecture 504
F.58.2. One giant enterprise table 504
F.58.3. Current-state-only ingestion 504
F.58.4. Tool object as universal unit 504
F.58.5. Identity everywhere 505
F.58.6. Graph database as semantics 505
F.58.7. Metric SQL in dashboards 505
F.58.8. Experiment flag without exposure 505
F.58.9. Generated narrative without evidence links 505
F.58.10. Real-time everything 505
F.58.11. Raw data access as self-service 505
F.59. Technical reference checklist 505
Appendix G: Operating Templates and Records 507
G.1. How to use these templates 507
G.2. Decision question template 507
G.3. Evidence request template 508
G.4. Metric proposal template 508
G.5. Metric validation record 509
G.6. Data-quality exception record 510
G.7. Engineering Change correction request 510
G.8. Study charter template 510
G.9. Experiment readiness review 511
G.10. Pre-analysis plan template 511
G.11. Study result record 512
G.12. Decision record template 512
G.13. Business-case summary template 513
G.14. Value-realization record 514
G.15. Governance use registration 514
G.16. Access request template 515
G.17. Governance exception template 515
G.18. Governance incident record 515

G.19. Architecture decision record 516
G.20. Source onboarding record 516
G.21. Evidence review agenda 517
G.22. Monthly learning review template 517
G.23. Quarterly portfolio record 518
G.24. Record status vocabulary 518
G.25. Review-date discipline 519
G.26. Evidence references 519
G.27. Correction history 519
G.28. Minimum operating record set 520
G.29. Template anti-patterns 520
G.29.1. Form without ownership 520
G.29.2. Required fields with meaningless placeholders 520
G.29.3. Record after the decision 520
G.29.4. URLs instead of durable references 520
G.29.5. No review date 520
G.29.6. Templates as bureaucracy 521
G.30. Operating-template checklist 521
Appendix H: Glossary and Controlled Vocabulary 523
H.1. Vocabulary principles 523
H.1.1. Prefer semantic meaning over tool terminology 523
H.1.2. Distinguish observation from interpretation 523
H.1.3. Distinguish people, automation, and agents 523
H.1.4. Distinguish assignment from exposure 523
H.1.5. Distinguish capacity from savings 523
H.1.6. Version important semantics 523
H.2. A 523
H.2.1. Acceptance 523
H.2.2. Acceptance rate 523
H.2.3. Accountable owner 524
H.2.4. Active time 524
H.2.5. Actor 524
H.2.6. Agent 524
H.2.7. Agentic workflow 524
H.2.8. Aggregation 524
H.2.9. Analysis unit 524
H.2.10. Annotation 524
H.2.11. Artifact 524
H.2.12. Assignment 524
H.2.13. Assignment unit 525
H.2.14. Attention 525
H.2.15. Attention cost 525
H.2.16. Attribution 525
H.2.17. Audit trail 525
H.2.18. Automation 525
H.2.19. Autonomous action 525
H.2.20. Availability 525
H.3. B 525
H.3.1. Backfill 525
H.3.2. Baseline 526
H.3.3. Benchmark 526

H.3.4. Benefit . . . 526
H.3.5. Blast radius . . . 526
H.3.6. Blocking time . . . 526
H.3.7. Bot . . . 526
H.3.8. Bottleneck . . . 526
H.3.9. Build . . . 526
H.3.10. Business case . . . 526
H.4. C . . . 526
H.4.1. Calibration . . . 526
H.4.2. Canonical event . . . 527
H.4.3. Canonical identity . . . 527
H.4.4. Capacity . . . 527
H.4.5. Capacity realization . . . 527
H.4.6. Carryover . . . 527
H.4.7. Causal claim . . . 527
H.4.8. Causal effect . . . 527
H.4.9. Change failure . . . 527
H.4.10. Change type . . . 527
H.4.11. Closed-loop automation . . . 527
H.4.12. Cluster . . . 527
H.4.13. Cohort . . . 528
H.4.14. Comparator . . . 528
H.4.15. Completeness . . . 528
H.4.16. Complexity . . . 528
H.4.17. Confidence . . . 528
H.4.18. Confidence band . . . 528
H.4.19. Confidence interval . . . 528
H.4.20. Confounder . . . 528
H.4.21. Consequence . . . 528
H.4.22. Construct . . . 528
H.4.23. Construct validity . . . 528
H.4.24. Contamination . . . 529
H.4.25. Continuity factor . . . 529
H.4.26. Control group . . . 529
H.4.27. Correction . . . 529
H.4.28. Cost of delay . . . 529
H.4.29. Counterfactual . . . 529
H.4.30. Credible interval . . . 529
H.4.31. Cycle time . . . 529
H.5. D . . . 529
H.5.1. Data contract . . . 529
H.5.2. Data lineage . . . 529
H.5.3. Data minimization . . . 530
H.5.4. Data-quality gate . . . 530
H.5.5. Decision-grade . . . 530
H.5.6. Decision owner . . . 530
H.5.7. Decision record . . . 530
H.5.8. Delegation . . . 530
H.5.9. Delegation graph . . . 530
H.5.10. Deployment . . . 530
H.5.11. Deployment frequency . . . 530

H.5.12. Descriptive claim 530
H.5.13. Digital twin 530
H.5.14. Direct cost 531
H.5.15. Distributional effect 531
H.5.16. Drift 531
H.6. E 531
H.6.1. Economic value 531
H.6.2. Effect size 531
H.6.3. Eligibility 531
H.6.4. Engineering Change 531
H.6.5. Engineering Change graph 531
H.6.6. Engineering Intelligence 531
H.6.7. Estimand 531
H.6.8. Event time 532
H.6.9. Evidence 532
H.6.10. Evidence graph 532
H.6.11. Evidence maturity 532
H.6.12. Exposure 532
H.6.13. External validity 532
H.7. F 532
H.7.1. Fairness 532
H.7.2. False negative 532
H.7.3. False positive 532
H.7.4. Feature 532
H.7.5. Finding 532
H.7.6. Finding acceptance 533
H.7.7. Finding recall 533
H.7.8. Finding precision 533
H.7.9. Flow 533
H.7.10. Fragmentation 533
H.7.11. Freshness 533
H.8. G 533
H.8.1. Gaming 533
H.8.2. Generated artifact 533
H.8.3. Generated explanation 533
H.8.4. Governance 533
H.8.5. Governance incident 533
H.8.6. Ground truth 534
H.8.7. Guardrail 534
H.9. H 534
H.9.1. Hazard 534
H.9.2. Human-in-the-loop 534
H.9.3. Human-on-the-loop 534
H.9.4. Human validation burden 534
H.9.5. Hypothesis 534
H.10. I 534
H.10.1. Identification strategy 534
H.10.2. Identity resolution 534
H.10.3. Immutable evidence 534
H.10.4. Implementation fidelity 535
H.10.5. Incident 535

H.10.6. Incremental cost 535
H.10.7. Incremental value 535
H.10.8. Inference 535
H.10.9. Ingestion time 535
H.10.10. Instrumentation 535
H.10.11. Instrumental variable 535
H.10.12. Intention-to-treat effect 535
H.10.13. Interference 535
H.10.14. Internal validity 535
H.10.15. Intervention 535
H.11. J 536
H.11.1. Join coverage 536
H.12. L 536
H.12.1. Latency 536
H.12.2. Learning organization 536
H.12.3. Learning velocity 536
H.12.4. Lifecycle state 536
H.12.5. Linkage confidence 536
H.12.6. Loaded labor cost 536
H.12.7. Longitudinal study 536
H.13. M 536
H.13.1. Maintainability 536
H.13.2. Mechanism 537
H.13.3. Measurement invariance 537
H.13.4. Measurement model 537
H.13.5. Metric 537
H.13.6. Metric bundle 537
H.13.7. Metric incident 537
H.13.8. Metric owner 537
H.13.9. Missingness 537
H.13.10. Model 537
H.13.11. Model monoculture 537
H.13.12. Model provenance 537
H.13.13. Multi-tenancy 538
H.14. N 538
H.14.1. Natural experiment 538
H.14.2. Net attention change 538
H.14.3. Net present value 538
H.14.4. Noncompliance 538
H.14.5. Noninferiority 538
H.14.6. Novelty 538
H.15. O 538
H.15.1. Observable 538
H.15.2. Observability 538
H.15.3. Observational study 539
H.15.4. Opportunity cost 539
H.15.5. Option value 539
H.15.6. Outcome 539
H.15.7. Outcome maturity 539
H.15.8. Ownership 539
H.16. P 539

H.16.1. Policy 539
H.16.2. Policy-grade metric 539
H.16.3. Population 539
H.16.4. Population drift 539
H.16.5. Precision 539
H.16.6. Prediction 540
H.16.7. Predictive claim 540
H.16.8. Prescriptive claim 540
H.16.9. Primary outcome 540
H.16.10. Privacy 540
H.16.11. Probabilistic linkage 540
H.16.12. Production function 540
H.16.13. Prompt provenance 540
H.16.14. Propensity score 540
H.16.15. Provenance 540
H.16.16. Proxy 540
H.16.17. Purpose limitation 540
H.17. Q 541
H.17.1. Quality 541
H.17.2. Quality gate 541
H.17.3. Queue 541
H.17.4. Queue delay 541
H.17.5. Quasi-experiment 541
H.18. R 541
H.18.1. Randomization 541
H.18.2. Realized value 541
H.18.3. Recall 541
H.18.4. Reconstruction 541
H.18.5. Registered use 541
H.18.6. Regression discontinuity 542
H.18.7. Reliability 542
H.18.8. Release 542
H.18.9. Released capacity 542
H.18.10. Replay 542
H.18.11. Replication 542
H.18.12. Reproducibility 542
H.18.13. Retention 542
H.18.14. Reversibility 542
H.18.15. Review 542
H.18.16. Review effort 542
H.18.17. Risk 542
H.18.18. Risk class 543
H.18.19. Rollback 543
H.19. S 543
H.19.1. Sample 543
H.19.2. Schema registry 543
H.19.3. Sensitivity analysis 543
H.19.4. Service time 543
H.19.5. Shadow mode 543
H.19.6. Source evidence 543
H.19.7. Spillover 543

H.19.8. Standard error . . . 543
H.19.9. Statistical significance . . . 543
H.19.10. Stratification . . . 544
H.19.11. Study . . . 544
H.19.12. Study charter . . . 544
H.19.13. Subgroup . . . 544
H.19.14. Surveillance . . . 544
H.19.15. Synthetic control . . . 544
H.19.16. Synthetic evidence . . . 544
H.19.17. System of record . . . 544
H.20. T . . . 544
H.20.1. Technical debt . . . 544
H.20.2. Temporal join . . . 544
H.20.3. Temporal ownership . . . 544
H.20.4. Treatment . . . 545
H.20.5. Treatment fidelity . . . 545
H.20.6. Treatment-on-treated effect . . . 545
H.20.7. Throughput . . . 545
H.20.8. Tool usage . . . 545
H.20.9. Traceability . . . 545
H.20.10. Transportability . . . 545
H.21. U . . . 545
H.21.1. Uncertainty . . . 545
H.21.2. Unit of analysis . . . 545
H.21.3. Unit of consequence . . . 545
H.21.4. Utilization . . . 545
H.22. V . . . 546
H.22.1. Validation . . . 546
H.22.2. Validation burden . . . 546
H.22.3. Validation run . . . 546
H.22.4. Validity . . . 546
H.22.5. Value realization . . . 546
H.22.6. Vendor-neutral evidence . . . 546
H.22.7. Version . . . 546
H.23. W . . . 546
H.23.1. Wait time . . . 546
H.23.2. Work in progress . . . 546
H.23.3. Work mix . . . 546
H.23.4. Work-mix adjustment . . . 547
H.23.5. Workflow . . . 547
H.24. Controlled distinctions . . . 547
H.24.1. Engineering Change versus pull request . . . 547
H.24.2. Deployment versus release . . . 547
H.24.3. Activity versus outcome . . . 547
H.24.4. Elapsed time versus active effort . . . 547
H.24.5. Usage versus effectiveness . . . 547
H.24.6. Adoption versus exposure . . . 547
H.24.7. Correlation versus causation . . . 547
H.24.8. Prediction versus causal estimate . . . 547
H.24.9. Confidence versus probability . . . 547
H.24.10. Precision versus accuracy . . . 548

H.24.11. Capacity value versus cash savings 548
H.24.12. Avoided cost versus savings 548
H.24.13. Generated explanation versus observed evidence 548
H.24.14. Automation versus agent 548
H.24.15. Review finding versus defect 548
H.24.16. Quality signal versus quality outcome 548
H.24.17. Metric quality versus engineering quality 548
H.24.18. Data freshness versus event latency 548
H.24.19. Governance approval versus scientific validity 548
H.24.20. Reproducibility versus replication 548
H.24.21. Internal validity versus external validity 549
H.24.22. Policy versus treatment 549
H.24.23. Guardrail versus target 549
H.24.24. Correction versus deletion 549
H.25. Vocabulary governance 549
H.26. Vocabulary checklist 550
Appendix I: Evidence Base and Bibliography 551
I.1. How to use this evidence base 551
Software delivery and organizational performance 551
Socio-technical systems and software architecture 552
Human attention, cognition, and interruption 552
Measurement theory and metrics 553
Queueing, flow, and operations 553
Causal inference and experimentation 554
Ranking and information-retrieval evaluation 554
Organizational learning, safety, and resilience 555
Economics and productivity 555
AI-assisted software engineering 556
AI risk, governance, and standards 556
Data systems, provenance, and observability 557
Research design, reproducibility, and reporting 557
I.2. Source-evidence classification 558
I.2.1. Foundational theory 558
I.2.2. Peer-reviewed empirical evidence 558
I.2.3. Standards and official frameworks 558
I.2.4. Practitioner synthesis 558
I.2.5. Emerging preprint 558
I.2.6. Organizational case evidence 558
I.3. Citation-quality rules 558
I.4. Evidence gaps and active research areas 559
I.5. Bibliography maintenance checklist 559
Appendix J: Maturity Assessment and Adoption Roadmap 561
J.1. Assessment philosophy 561
J.2. Maturity levels 561
J.2.1. Level 0: fragmented telemetry 563
J.2.2. Level 1: descriptive visibility 563
J.2.3. Level 2: connected Engineering Change evidence 563
J.2.4. Level 3: governed decision intelligence 564
J.2.5. Level 4: experimental learning system 564
J.2.6. Level 5: adaptive engineering organization 565

J.3. Assessment dimensions . . . 565
J.4. Decision orientation . . . 565
J.4.1. Diagnostic questions . . . 565
J.4.2. Evidence of maturity . . . 565
J.4.3. Common failure . . . 565
J.5. Source evidence and data contracts . . . 566
J.5.1. Diagnostic questions . . . 566
J.5.2. Evidence of maturity . . . 566
J.5.3. Common failure . . . 566
J.6. Engineering Change semantics . . . 566
J.6.1. Diagnostic questions . . . 566
J.6.2. Evidence of maturity . . . 566
J.6.3. Common failure . . . 567
J.7. Identity, ownership, and organizational context . . . 567
J.7.1. Diagnostic questions . . . 567
J.7.2. Evidence of maturity . . . 567
J.7.3. Common failure . . . 567
J.8. Metric quality . . . 567
J.8.1. Diagnostic questions . . . 567
J.8.2. Evidence of maturity . . . 567
J.8.3. Common failure . . . 568
J.9. Human attention and work-system measurement . . . 568
J.9.1. Diagnostic questions . . . 568
J.9.2. Evidence of maturity . . . 568
J.9.3. Common failure . . . 568
J.10. Experimentation and causal inference . . . 568
J.10.1. Diagnostic questions . . . 568
J.10.2. Evidence of maturity . . . 568
J.10.3. Common failure . . . 569
J.11. Economic evaluation . . . 569
J.11.1. Diagnostic questions . . . 569
J.11.2. Evidence of maturity . . . 569
J.11.3. Common failure . . . 569
J.12. Architecture and complexity intelligence . . . 569
J.12.1. Diagnostic questions . . . 569
J.12.2. Evidence of maturity . . . 570
J.12.3. Common failure . . . 570
J.13. Governance, privacy, and trust . . . 570
J.13.1. Diagnostic questions . . . 570
J.13.2. Evidence of maturity . . . 570
J.13.3. Common failure . . . 570
J.14. AI and agent readiness . . . 570
J.14.1. Diagnostic questions . . . 570
J.14.2. Evidence of maturity . . . 571
J.14.3. Common failure . . . 571
J.15. Platform and operational reliability . . . 571
J.15.1. Diagnostic questions . . . 571
J.15.2. Evidence of maturity . . . 571
J.15.3. Common failure . . . 571
J.16. Organizational learning . . . 572
J.16.1. Diagnostic questions . . . 572

J.16.2. Evidence of maturity 572
J.16.3. Common failure 572
J.17. Scoring method 572
J.18. Weighting by decision risk 573
J.19. Critical-capability floor 573
J.20. Assessment process 574
J.21. Assessment outputs 574
J.22. Selecting the first decision 574
J.23. Poor first decisions 575
J.24. First 30 days: define and bound 575
J.24.1. Outcomes 575
J.24.2. Activities 575
J.24.3. Exit criteria 576
J.25. Days 31 to 60: build the evidence path 576
J.25.1. Outcomes 576
J.25.2. Activities 576
J.25.3. Exit criteria 577
J.26. Days 61 to 90: evaluate and decide 577
J.26.1. Outcomes 577
J.26.2. Activities 577
J.26.3. Exit criteria 577
J.27. Months 4 to 6: operationalize 578
J.28. Months 7 to 12: expand causal and economic capability 578
J.29. Year 2: enterprise learning system 578
J.30. Capability dependency map 579
J.31. Team and sponsorship model 579
J.32. Funding model 579
J.33. Adoption signals 580
J.34. Warning signals 580
J.35. Maturity reassessment cadence 580
J.36. Executive maturity summary template 581
J.37. Roadmap decision rules 581
J.38. Assessment anti-patterns 582
J.38.1. Maturity theater 582
J.38.2. Average-score optimism 582
J.38.3. Tool-based maturity 582
J.38.4. Roadmap by connector 582
J.38.5. Enterprise-wide first step 582
J.38.6. Permanent pilot team 582
J.38.7. Governance last 582
J.38.8. No realization review 582
J.39. Maturity and adoption checklist 582
Appendix K: Chapter Source Notes and Citation Map 585
K.1. Citation conventions 585
K.2. Chapter 1 — Introduction 585
K.2.1. Established foundations 585
K.2.2. Original synthesis 585
K.3. Chapter 2 — Engineering Economics 586
K.3.1. Established foundations 586
K.3.2. Original synthesis 586
K.4. Chapter 3 — Engineering Change 586

K.4.1. Established foundations . . . 586
K.4.2. Original synthesis . . . 586
K.5. Chapter 4 — Human Attention . . . 586
K.5.1. Established foundations . . . 586
K.5.2. Original synthesis . . . 587
K.6. Chapter 5 — Measuring Engineering Systems . . . 587
K.6.1. Established foundations . . . 587
K.6.2. Original synthesis . . . 587
K.7. Chapter 6 — Experimentation and Causal Inference . . . 587
K.7.1. Established foundations . . . 587
K.7.2. Original synthesis . . . 587
K.8. Chapter 7 — AI-Assisted Engineering . . . 587
K.8.1. Established and emerging evidence . . . 588
K.8.2. Caution . . . 588
K.8.3. Original synthesis . . . 588
K.9. Chapter 8 — Enterprise Instrumentation . . . 588
K.9.1. Established foundations . . . 588
K.9.2. Original synthesis . . . 588
K.10. Chapter 9 — Complexity and Blast Radius . . . 588
K.10.1. Established foundations . . . 588
K.10.2. Original synthesis . . . 589
K.11. Chapter 10 — The Engineering Intelligence Platform . . . 589
K.11.1. Established foundations . . . 589
K.11.2. Original synthesis . . . 589
K.12. Chapter 11 — Engineering Metrics Catalogue . . . 589
K.12.1. Established foundations . . . 589
K.12.2. Original synthesis . . . 589
K.13. Chapter 12 — Experiment Playbooks . . . 590
K.13.1. Established foundations . . . 590
K.13.2. Original synthesis . . . 590
K.14. Chapter 13 — Governance, Ethics, and Adoption . . . 590
K.14.1. Established foundations . . . 590
K.14.2. Original synthesis . . . 590
K.15. Chapter 14 — Portfolio and Executive Decision-Making . . . 590
K.15.1. Established foundations . . . 590
K.15.2. Original synthesis . . . 590
K.16. Chapter 15 — Implementation and Rollout . . . 591
K.16.1. Established foundations . . . 591
K.16.2. Original synthesis . . . 591
K.17. Chapter 16 — The Science of Engineering Intelligence . . . 591
K.17.1. Established foundations . . . 591
K.17.2. Original synthesis . . . 591
K.18. Chapter 17 — The Learning Engineering Organization . . . 591
K.18.1. Established foundations . . . 591
K.18.2. Original synthesis . . . 591
K.19. Chapter 18 — The Future of Engineering Intelligence . . . 592
K.19.1. Established foundations . . . 592
K.19.2. Emerging and original material . . . 592
K.20. Appendix source relationships . . . 592
K.20.1. Appendix A — Engineering Change Reference Model . . . 592
K.20.2. Appendix B — Metric Specification Standard . . . 592

K.20.3. Appendix C — Experiment and Causal Study Protocol . . . 592
K.20.4. Appendix D — Governance, Privacy, and Acceptable Use Standard . . . 592
K.20.5. Appendix E — Economic Evaluation and Business Case Standard . . . 592
K.20.6. Appendix F — Technical Reference Architecture . . . 592
K.20.7. Appendix G — Operating Templates and Records . . . 592
K.20.8. Appendix H — Glossary and Controlled Vocabulary . . . 593
K.20.9. Appendix I — Evidence Base and Bibliography . . . 593
K.20.10. Appendix J — Maturity Assessment and Adoption Roadmap . . . 593
K.21. Citation integration roadmap . . . 593
K.22. Citation map checklist . . . 593
Index . . . **595**

Chapter 1. Copyright and publication

Engineering Intelligence: A Quantitative Framework for Measuring, Understanding, and Optimizing Modern Software Engineering

Copyright © 2026 Christopher J. Collins. All rights reserved.

No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means — electronic, mechanical, photocopying, recording, or otherwise — without the prior written permission of the copyright holder, except brief quotations embodied in critical reviews and certain other non-commercial uses permitted by copyright law.

Every reasonable effort has been made to acknowledge the ideas, evidence, and prior work of others through the citations and source notes accompanying each chapter. Errors of omission are unintentional; corrections may be directed to the publisher and will be acknowledged in subsequent editions.

The methods, frameworks, and worked examples in this book are provided for informational purposes. Application to any specific organization should be tempered by local context, legal and privacy requirements, and the judgement of accountable practitioners.

Published by OrionWorks. Cover art and interior illustration plates by the publisher.

Set in Times Roman and Georgia. Diagrams rendered with Mermaid; mathematical typography with MathJax.

First edition — 2026.

Build 2026-10-19 · commit `8d5a18f8`

Chapter 2. Dedication

For the engineers whose careful work is rarely counted, and for the leaders willing to ask what their measurements are actually measuring.

Preface

Software engineering organizations are increasingly instrumented, yet they often remain poorly understood. They can report pull-request counts, deployment frequency, incident volume, and tool usage, but still struggle to explain where expert attention is spent, which process changes create value, or whether an AI-assisted workflow genuinely improves engineering outcomes.

Engineering Intelligence develops a framework for answering those questions. It treats engineering as a measurable socio-technical system and integrates engineering economics, human cognition, software architecture, queueing theory, causal inference, and AI-assisted development.

Chapter 3. Acknowledgments

This book stands on decades of work by others. The intellectual foundations of engineering economics, human cognition, queueing theory, causal inference, software architecture, and organizational learning are not new; the contribution of this book is a synthesis of those foundations into a single measurable framework for modern software engineering.

The bibliography and per-chapter source notes name the specific researchers, practitioners, and organizations whose work shaped each argument. Any errors of interpretation, emphasis, or omission are the author's alone.

The framework was refined in conversation with engineers, engineering leaders, platform architects, data scientists, and social scientists across many organizations. Their willingness to discuss what they measure — and what they wish they could measure — improved every chapter. Individual acknowledgment is withheld to preserve the confidentiality of specific operating contexts, but the debt is real.

Several early readers offered corrections that materially improved the manuscript. Their patience with drafts that were more ambitious than clear is gratefully recorded here.

Finally, this book was written in the presence of AI-assisted writing tools, applied under the same evaluation discipline it advocates for AI-assisted engineering. Wherever a tool contributed to structure, phrasing, or verification, the author reviewed and rewrote the output before adoption. The book's arguments, judgements, and errors remain the author's.

Executive Summary

Software engineering organizations generate extraordinary amounts of operational evidence, yet often remain unable to answer their most consequential questions with confidence.

They can count commits, pull requests, deployments, incidents, code suggestions, review comments, and tool invocations. They can rarely explain, with the same precision:

- where scarce expert attention is consumed;
- which queues and architectural dependencies limit delivery;
- whether a tool or policy actually caused an improvement;
- whether faster activity produced better outcomes;
- whether released capacity became economic value;
- whether an AI-assisted workflow improved the system after validation, rework, quality, and risk are included;
- whether a metric is sufficiently valid, complete, and governed for the decision being made.

Engineering Intelligence presents a unified framework for solving this problem. It treats software engineering as a measurable socio-technical and economic system: a system in which people, AI participants, repositories, services, dependencies, review processes, build infrastructure, deployment mechanisms, production outcomes, organizational structures, and financial constraints interact.

The objective is not to create more dashboards. It is to improve the quality, speed, reproducibility, and accountability of engineering decisions.

The central thesis

The fundamental unit of analysis should not be the developer, commit, pull request, ticket, deployment, or AI suggestion. Each is only a partial artifact of a larger process.

The central unit is the **Engineering Change**: a vendor-independent representation of an intended modification and its implementation, validation, delivery, operation, and observed consequences.

An Engineering Change connects:

- the intent or problem being addressed;
- the human and machine actors involved;
- source, configuration, infrastructure, data, and documentation artifacts;
- review and validation activity;
- builds and deployable artifacts;
- deployments and releases;
- incidents, defects, rollbacks, and operational outcomes;
- user, business, economic, and learning outcomes.

This unit allows organizations to measure the complete change system rather than optimizing isolated tools or stages.

Engineering is an attention-allocation system

Engineering capacity is not adequately represented by headcount or hours. The scarcest resource is often expert attention: the ability to understand, decide, validate, coordinate, diagnose, and accept responsibility.

Attention is consumed by:

- implementation;
- code and architecture review;
- waiting and context reconstruction;
- build and test failures;
- unclear ownership;
- cross-team dependencies;
- incident response;
- validating AI-generated output;
- interpreting weak or contradictory evidence;
- navigating organizational policy.

The most valuable interventions do not merely increase visible activity. They release or redirect scarce attention toward higher-value decisions.

This leads to a critical economic distinction:

> Released time is not automatically financial savings, and additional output is not automatically value.

Capacity becomes valuable only when it appears in usable form, reaches constrained valuable work, and is absorbed by downstream systems capable of converting it into outcomes.

Engineering is a queueing system

Engineering work moves through queues for implementation, review, specialist approval, CI, test environments, deployment, incident response, and organizational decisions.

Queue behavior is nonlinear. Near high utilization, small changes in arrival rate, variability, or service capacity can produce disproportionately large changes in waiting time.

This means:

- increasing upstream output can worsen end-to-end flow;
- optimizing a non-bottleneck may create little value;
- reducing variability may be as important as increasing nominal capacity;
- scarce reviewers and environments should be treated as constrained service systems;
- cycle-time averages can conceal severe tail behavior.

Engineering Intelligence therefore measures arrival, service, wait, work in progress, variability, utilization, and lifecycle state together.

Engineering is an economic system

Engineering investments should be evaluated against credible alternatives, not against zero.

A complete business case includes:

- acquisition and implementation cost;
- infrastructure and inference cost;
- support, administration, security, and governance;
- human validation and rework;
- migration, switching, and exit cost;
- quality and operational risk;
- opportunity cost;
- released capacity;
- avoided cost;
- cost-of-delay reduction;
- risk reduction;
- option and capability value.

The framework explicitly separates:

- booked financial savings;
- committed financial impact;
- avoided planned cost;
- released capacity;
- modeled outcome value;
- strategic option value.

This prevents modeled engineering time from being reported as realized cash savings.

Metrics are decision products

A metric is not a number or SQL query. It is a versioned analytical product with:

- a decision purpose;
- a defined construct;
- a unit and eligible population;
- exact formula and aggregation behavior;
- source and transformation lineage;
- data-quality requirements;
- missing-data semantics;
- uncertainty;
- validity and reliability evidence;
- interpretation and prohibited uses;
- privacy and governance controls;
- an owner, review cadence, and retirement condition.

A metric that is useful for team workflow improvement may be invalid for cross-team comparison. A metric valid for tool evaluation may be prohibited for individual performance decisions.

The question is never simply, “Is this metric valid?” It is:

> Is this metric sufficiently valid for this population, decision, consequence, and use?

Measurement must support causal learning

Descriptive telemetry can show that two things moved together. It cannot, by itself, establish that an intervention caused the outcome.

Engineering Intelligence supports a continuum of evidence:

- descriptive observation;
- predictive modeling;
- designed observational studies;
- natural and quasi-experiments;
- randomized experiments;
- replicated causal evidence;
- monitored operational policy.

Experiments require more than a feature flag. They require:

- executable eligibility;
- an appropriate assignment unit;
- deterministic assignment;
- treatment and comparator specifications;
- observed treatment exposure;
- primary outcomes and guardrails;
- power and readiness review;
- interference and noncompliance analysis;
- a predeclared analysis plan;
- decision and stopping rules;
- long-term follow-up and replication.

Assignment must remain distinct from exposure. Adoption must remain distinct from effectiveness.

AI changes the production system

AI-assisted engineering should not be evaluated through suggestion volume, acceptance rate, generated code, or anecdotal enthusiasm alone.

AI may reduce authoring effort while increasing:

- review and validation burden;
- false-positive handling;
- rework;
- maintenance complexity;
- security review;
- model and infrastructure cost;

- correlated failure risk;
- skill atrophy;
- dependence on a vendor or model family.

The correct unit is the net effect on the Engineering Change and the surrounding system.

A useful economic expression is:

$$NetAttentionChange = AuthoringAttentionReduced - ValidationAttentionAdded - ReworkAttentionAdded$$

AI systems also require provenance: provider, model, configuration, prompt, retrieval context, tool permissions, initiating actor, delegation chain, latency, and cost.

As agents become more autonomous, the Engineering Change graph must include machine identity, delegated authority, validation responsibility, and acceptance accountability.

Architecture is measurable consequence

Architecture influences the cost and risk of change through:

- dependency reach;
- ownership fragmentation;
- cognitive load;
- validation scope;
- deployment coupling;
- blast radius;
- specialist requirements;
- operational recovery.

Architecture should not be reduced to static complexity counts. It should be evaluated by how structure changes the effort, coordination, safety, and reversibility of Engineering Changes.

The framework combines repository, service, runtime, deployment, ownership, and incident graphs to evaluate architectural constraints over time.

The platform is an evidence system

A credible Engineering Intelligence platform preserves a traceable path from source evidence to decision.

The reference architecture includes:

- connectors and event ingestion;
- immutable raw evidence;
- canonical events and schema contracts;
- temporal identity and ownership;
- service and repository catalogues;
- Engineering Change reconstruction;
- confidence-aware relationships;
- analytical, graph, search, and operational projections;

- metric, model, experiment, evidence, and decision registries;
- purpose-bound serving products;
- privacy, security, audit, correction, replay, and quality controls.

The architecture separates:

- observed evidence;
- reconstructed relationships;
- model inference;
- generated explanation;
- causal estimate;
- organizational decision.

This separation is essential because each carries different uncertainty and authority.

Governance is part of measurement quality

An unrestricted evidence system will damage trust and eventually damage the evidence itself.

Engineering Intelligence therefore includes controls for:

- purpose limitation;
- data minimization;
- use classification;
- least-privilege and purpose-based access;
- identity isolation and aggregation thresholds;
- correction and appeal;
- prohibited individual ranking;
- vendor and model governance;
- automated-action boundaries;
- retention, deletion, and export;
- governance incidents and rollback.

The framework presumes that ordinary engineering telemetry is unsuitable for individual compensation, promotion, discipline, or termination decisions unless a separate formal process establishes exceptional validity, necessity, fairness, legal authority, transparency, and due process. In most organizations, such uses should remain prohibited.

The operating model

Engineering Intelligence becomes useful when embedded in an operating cadence.

A complete loop is:

1. identify a consequential decision;
2. define the construct and evidence required;
3. instrument and validate the evidence path;
4. specify governed metrics;

5. run descriptive or causal analysis;
6. evaluate economics and risk;
7. make and record a bounded decision;
8. monitor implementation and guardrails;
9. measure realized outcomes;
10. preserve learning, corrections, and negative results;
11. revisit the decision when assumptions or conditions change.

The book provides standard templates for each part of this loop.

A maturity path

Organizations can develop the capability incrementally.

Stage 1: trustworthy evidence

Establish immutable source evidence, canonical events, identity, ownership, lineage, and data-quality visibility.

Stage 2: Engineering Change intelligence

Reconstruct end-to-end changes and connect artifacts, activities, validation, deployment, release, incidents, and outcomes.

Stage 3: governed measurement

Create metric specifications, registries, quality gates, uncertainty, and interpretation controls.

Stage 4: causal learning

Add assignment, exposure, guardrails, experiments, quasi-experiments, and reproducibility.

Stage 5: economic and portfolio intelligence

Connect engineering evidence to cost, capacity realization, delay, risk, business outcomes, and portfolio decisions.

Stage 6: AI and agent evaluation

Measure model versions, prompts, delegation, validation burden, correlated risk, and long-term capability effects.

Stage 7: bounded adaptive policy

Allow reversible, monitored, purpose-limited automation only where evidence, governance, and rollback are sufficiently mature.

What success looks like

A successful Engineering Intelligence program is not defined by the number of connectors, dashboards, metrics, or AI features.

It is visible when the organization can repeatedly:

- identify the real constraint;
- estimate uncertainty honestly;
- test interventions rather than debate anecdotes;
- distinguish capacity from savings;
- detect harmful tradeoffs;
- make bounded and reversible decisions;
- reproduce the evidence behind those decisions;
- learn across teams and over time;
- preserve practitioner trust.

The ultimate product is not telemetry. It is an organization that becomes better at learning how it engineers.

Chapter 4. Reader's Guide

Engineering Intelligence can be read sequentially, but it is also designed as a working reference. Different readers can enter through different paths depending on the decision they face.

4.1. For engineering executives

Begin with:

- the Executive Summary;
- Chapter 2, *Engineering Economics*;
- Chapter 14, *Portfolio and Executive Decision-Making*;
- Chapter 17, *The Learning Engineering Organization*;
- Appendix E, *Economic Evaluation and Business Case Standard*;
- Appendix G, *Operating Templates and Records*.

This path focuses on investment, capacity, risk, causal evidence, portfolio choices, and value realization.

4.2. For platform and developer-productivity leaders

Begin with:

- Chapter 3, *Engineering Change*;
- Chapter 4, *Human Attention*;
- Chapter 5, *Measuring Engineering Systems*;
- Chapter 10, *The Engineering Intelligence Platform*;
- Chapter 15, *Implementation and Rollout*;
- Appendix F, *Technical Reference Architecture*.

This path focuses on building an evidence system that connects workflow instrumentation to useful decisions.

4.3. For measurement scientists and analysts

Begin with:

- Chapter 5, *Measuring Engineering Systems*;
- Chapter 6, *Experimentation and Causal Inference*;
- Chapter 11, *Engineering Metrics Catalogue*;
- Chapter 12, *Experiment Playbooks*;
- Chapter 16, *The Science of Engineering Intelligence*;
- Appendix B, *Metric Specification Standard*;
- Appendix C, *Experiment and Causal Study Protocol*;
- Appendix I, *Evidence Base and Bibliography*.

This path focuses on constructs, validity, reliability, uncertainty, causal design, replication, and reporting.

4.4. For architects and senior engineers

Begin with:

- Chapter 3, *Engineering Change*;
- Chapter 4, *Human Attention*;
- Chapter 9, *Complexity and Blast Radius*;
- Chapter 17, *The Learning Engineering Organization*;
- Appendix A, *Engineering Change Reference Model*;
- Appendix F, *Technical Reference Architecture*.

This path treats architecture as a determinant of coordination cost, validation burden, change risk, and reversibility.

4.5. For AI engineering and evaluation teams

Begin with:

- Chapter 6, *Experimentation and Causal Inference*;
- Chapter 7, *AI-Assisted Engineering*;
- Chapter 12, *Experiment Playbooks*;
- Chapter 18, *The Future of Engineering Intelligence*;
- Appendix C, *Experiment and Causal Study Protocol*;
- Appendix D, *Governance, Privacy, and Acceptable Use Standard*;
- Appendix E, *Economic Evaluation and Business Case Standard*.

This path focuses on net attention, validation burden, model and agent provenance, quality guardrails, causal evaluation, economics, and governance.

4.6. For security, privacy, legal, and governance teams

Begin with:

- Chapter 8, *Enterprise Instrumentation*;
- Chapter 13, *Governance, Ethics, and Adoption*;
- Chapter 18, *The Future of Engineering Intelligence*;
- Appendix D, *Governance, Privacy, and Acceptable Use Standard*;
- Appendix F, *Technical Reference Architecture*;
- Appendix G, *Operating Templates and Records*.

This path focuses on purpose limitation, access, identity, retention, automated decisions, correction, appeal, and audit.

4.7. For practitioners and team leaders

Begin with:

- Chapter 1, *Introduction*;
- Chapter 3, *Engineering Change*;

- Chapter 4, *Human Attention*;
- Chapter 11, *Engineering Metrics Catalogue*;
- Chapter 13, *Governance, Ethics, and Adoption*;
- Appendix H, *Glossary and Controlled Vocabulary*.

This path explains how measurement can improve the work system without reducing people to activity counts.

4.8. How the book is organized

4.8.1. Foundations

Chapters 1 through 4 define the discipline, economic framing, Engineering Change, and human attention.

4.8.2. Measurement and causal learning

Chapters 5 and 6 establish measurement and experimentation principles.

4.8.3. AI, instrumentation, and architecture

Chapters 7 through 10 apply the framework to AI-assisted engineering, enterprise data, complexity, blast radius, and platform design.

4.8.4. Operational methods

Chapters 11 through 15 provide metrics, experiment playbooks, governance, executive decision methods, and rollout guidance.

4.8.5. Scientific and organizational maturity

Chapters 16 through 18 address research quality, organizational learning, and future human-agent engineering systems.

4.8.6. Reference standards

The appendices contain implementable reference models, metric and experiment standards, governance, economics, architecture, operating records, vocabulary, and bibliography.

4.9. Reading conventions

The book uses several recurring distinctions.

Observed evidence is directly recorded from a source system.

Reconstruction links evidence into entities, relationships, or lifecycle state.

Inference estimates something not directly observed.

Prediction estimates an unknown or future result.

Causal evidence estimates how an intervention changed an outcome relative to a counterfactual.

Decision is the organizational action selected from alternatives.

These categories should not be collapsed.

4.10. Applying the framework to a real decision

A practical reading and application sequence is:

1. Write the decision question.
2. Identify the unit of consequence and affected population.
3. Define the construct rather than beginning with available telemetry.
4. Map the Engineering Change lifecycle and actors.
5. Identify attention, queue, architecture, quality, and economic mechanisms.
6. Specify metrics and guardrails.
7. Assess source, linkage, validity, and privacy quality.
8. Choose descriptive, predictive, experimental, or quasi-experimental analysis.
9. Compare credible alternatives economically.
10. Make a bounded, reversible decision.
11. Record the decision, assumptions, and review date.
12. Measure adoption, exposure, guardrails, and realized outcomes.
13. Preserve corrections, limitations, and negative results.

Appendix G provides reusable records for this process.

4.11. What not to do

The book should not be used as justification for:

- ranking engineers by visible activity;
- treating one metric as universal productivity;
- reporting released capacity as booked savings;
- claiming causation from a before-and-after chart;
- assuming adoption proves value;
- deploying AI without measuring validation burden;
- collecting all available data before defining purpose;
- allowing generated narratives to replace evidence;
- automating consequential policy without rollback and appeal.

4.12. A note on context

No metric, model, benchmark, or organizational practice is context-free.

The definitions and standards in this book are designed to make context explicit: population, workflow, architecture, risk, work mix, time, ownership, source quality, treatment version, and decision consequence.

Readers should adapt implementations while preserving these semantic and scientific constraints.

4.13. A note on evidence maturity

Some foundations in this book draw on mature research in measurement, queueing, causal inference, organizational learning, and software delivery. Other areas—particularly long-term AI effects, autonomous engineering agents, skill formation, digital twins, and prescriptive engineering policy—remain active research questions.

The book distinguishes durable principles from emerging possibilities and treats uncertainty as information rather than an editorial weakness.

4.14. A note on terminology

Capitalized terms such as *Engineering Change* refer to controlled concepts defined in Appendix H. Tool-specific terminology should be mapped to these concepts rather than used as a substitute for them.

4.15. Using the templates

The reference templates are deliberately technology-neutral. They may become:

- versioned files in a repository;
- forms in a developer portal;
- registry records;
- workflow objects;
- governed documents;
- API resources.

Their value comes from preserving purpose, ownership, evidence, versions, uncertainty, conditions, and review—not from the storage format.

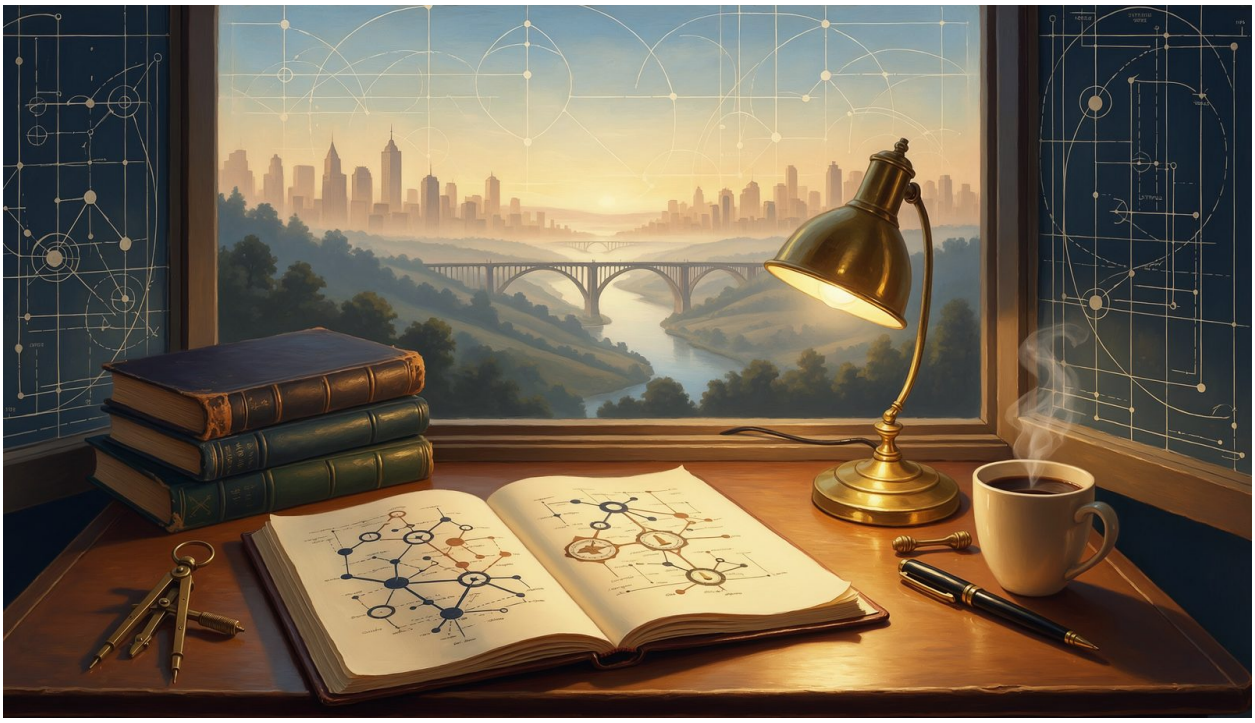
4.16. The intended outcome

The book succeeds when it helps an organization replace vague claims such as “engineering is faster” or “the AI tool is productive” with more useful statements:

> For this population and intervention version, under these evidence-quality conditions, we observed this effect and uncertainty, these guardrails remained within bounds, this amount of capacity was released, this portion became realized value, and this bounded decision will be reviewed on this date.

That is the language of an engineering organization capable of learning.

Chapter 5. Introduction



Engineering Intelligence is the discipline of measuring, modeling, understanding, and *improving software engineering as a complete socio-technical system. It extends beyond pull-request analytics and developer dashboards to connect engineering activity with human attention, AI participation, architectural complexity, operational risk, economic cost, and business outcomes. This chapter establishes the motivation, terminology, and scientific principles used throughout the book.*

5.1. Why engineering needs intelligence

Modern software organizations generate extraordinary amounts of operational data. Source-control platforms record every commit, branch, review, and merge. Continuous-integration systems retain build results and test histories. Issue trackers capture work planning and defect reports. Deployment systems know what reached production, when, and through which path. Observability platforms reveal how software behaved after release. AI coding and review tools add another stream of comments, suggestions, decisions, and generated artifacts.

Despite this abundance, many organizations still cannot answer basic questions with confidence:

- What does an engineering change actually cost?
- Where is expert human attention consumed?
- Which review activities prevent defects, and which merely add delay?

- Does an AI reviewer reduce cognitive burden, or does it create a new validation burden?
- Why do two apparently similar pull requests require radically different effort?
- How much cost is created by repository boundaries, team boundaries, approval policies, and deployment topology?
- Which interventions improve outcomes, and which merely move work elsewhere?

The obstacle is not a lack of metrics. It is a lack of models.

A metric describes an observation. A model explains how observations relate, identifies mechanisms, supports prediction, and allows competing interventions to be evaluated. Pull-request lead time, for example, is useful as a descriptive measure. It becomes operationally valuable only when an organization can distinguish waiting from active work, control for change complexity, identify the constrained stage, and estimate whether an intervention would improve quality-adjusted delivery rather than simply accelerate merging. This decision-oriented view is consistent with the broader measurement principle that validity attaches to a specific interpretation and use, not merely to a number's reproducibility. See [messick1995] and [hand2004].

Engineering Intelligence therefore begins with a change in objective:

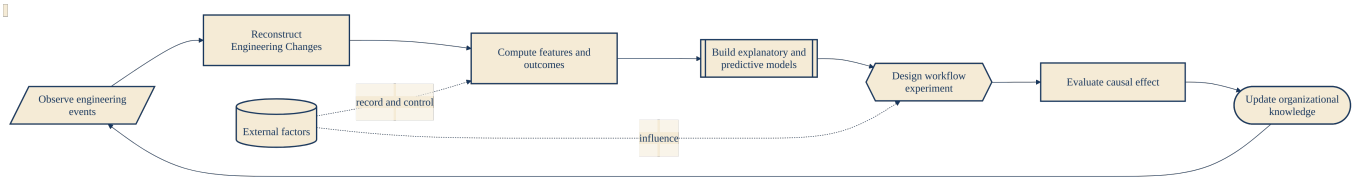
> The purpose of engineering measurement is not to count activity. It is to improve decisions about how scarce engineering resources are used.

Those resources include money and compute, but the most constrained resource in most engineering organizations is expert human attention. Build capacity can be purchased. AI inference can be scaled. Experienced judgment, architectural context, security expertise, and sustained cognitive focus remain comparatively scarce. Cognitive-load and interruption research provide part of the theoretical basis for treating attention as constrained and for distinguishing uninterrupted reasoning from fragmented work. See [sweller1988] and [mark2008].

5.2. From dashboards to a learning system

A dashboard can reveal that review latency increased. An Engineering Intelligence system should help determine why it increased, whether the increase is harmful, which populations are affected, what intervention is likely to work, and whether the intervention caused an improvement.

This requires a closed learning loop:



The loop contains six capabilities:

1. **Observation** — capture events from development, review, build, deployment, production, and planning systems.
2. **Reconstruction** — associate those events with coherent units of engineering work.
3. **Modeling** — estimate cost, latency, quality, risk, cognitive burden, and value while controlling for context.
4. **Explanation** — identify bottlenecks, relationships, and plausible causal mechanisms.
5. **Experimentation** — test changes to tools, policies, prompts, workflows, and organizational structures.
6. **Learning** — update organizational knowledge and feed evidence back into future decisions.

The final step is essential. Measurement that does not change a decision is, at best, descriptive. At worst, it creates

surveillance, metric gaming, and administrative burden without improving the engineering system. The risk that measures become distorted when converted into targets is well established in the work associated with Campbell's Law and Goodhart's Law. See [campbell1979] and [goodhart1984].

5.3. The limits of familiar productivity measures

Software engineering has repeatedly adopted measures that are easy to compute but dangerous to optimize directly.

Lines of code can reflect implementation volume, generated code, verbosity, duplication, or unnecessary complexity. Commit count depends on personal workflow and repository policy. Pull-request count is sensitive to slicing conventions. Story points are locally defined and frequently distorted when aggregated. Utilization rewards visible busyness while penalizing the slack required for learning, incident response, mentoring, and creative problem solving.

Even stronger frameworks must be used carefully. Delivery-performance measures such as deployment frequency, lead time for changes, change-failure rate, and restoration time are valuable system-level outcomes and are central to the software-delivery research synthesized in [forsgren2018]. The SPACE framework emphasizes that productivity cannot be represented by a single dimension and includes satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow. See [forsgren2021-space]. Developer Experience research further emphasizes feedback loops, cognitive load, and flow state. See [noda2023-devex] and [fagerholm2012].

Engineering Intelligence builds on these foundations. It does not attempt to replace them with one universal score. Instead, it adds four capabilities:

- a richer unit of analysis than the individual pull request;
- explicit economic and cognitive models;
- first-class treatment of AI participants;
- an experimentation and causal-inference layer.

The result is less like a scorecard and more like an observability and decision system for the engineering organization.

5.4. The engineering change as the unit of analysis

A pull request is a convenient platform object, but it is rarely the complete unit of engineering work.

A single product change may require:

- a user-interface pull request;
- an API change in another repository;
- a shared-library release;
- a database schema migration;
- infrastructure-as-code updates;
- a feature-flag definition;
- security and privacy approvals;
- documentation and operational runbook changes;
- a coordinated deployment sequence;
- production verification and cleanup work.

Conversely, one pull request may combine several unrelated changes. Pull-request boundaries are shaped by repository topology, team conventions, code ownership, and tooling limitations. Treating every pull request as an equivalent unit therefore introduces substantial measurement error. The broader relationship between organizational coordination and

technical structure is established in socio-technical research, including [conway1968], [cataldo2006], and [herbsleb2007].

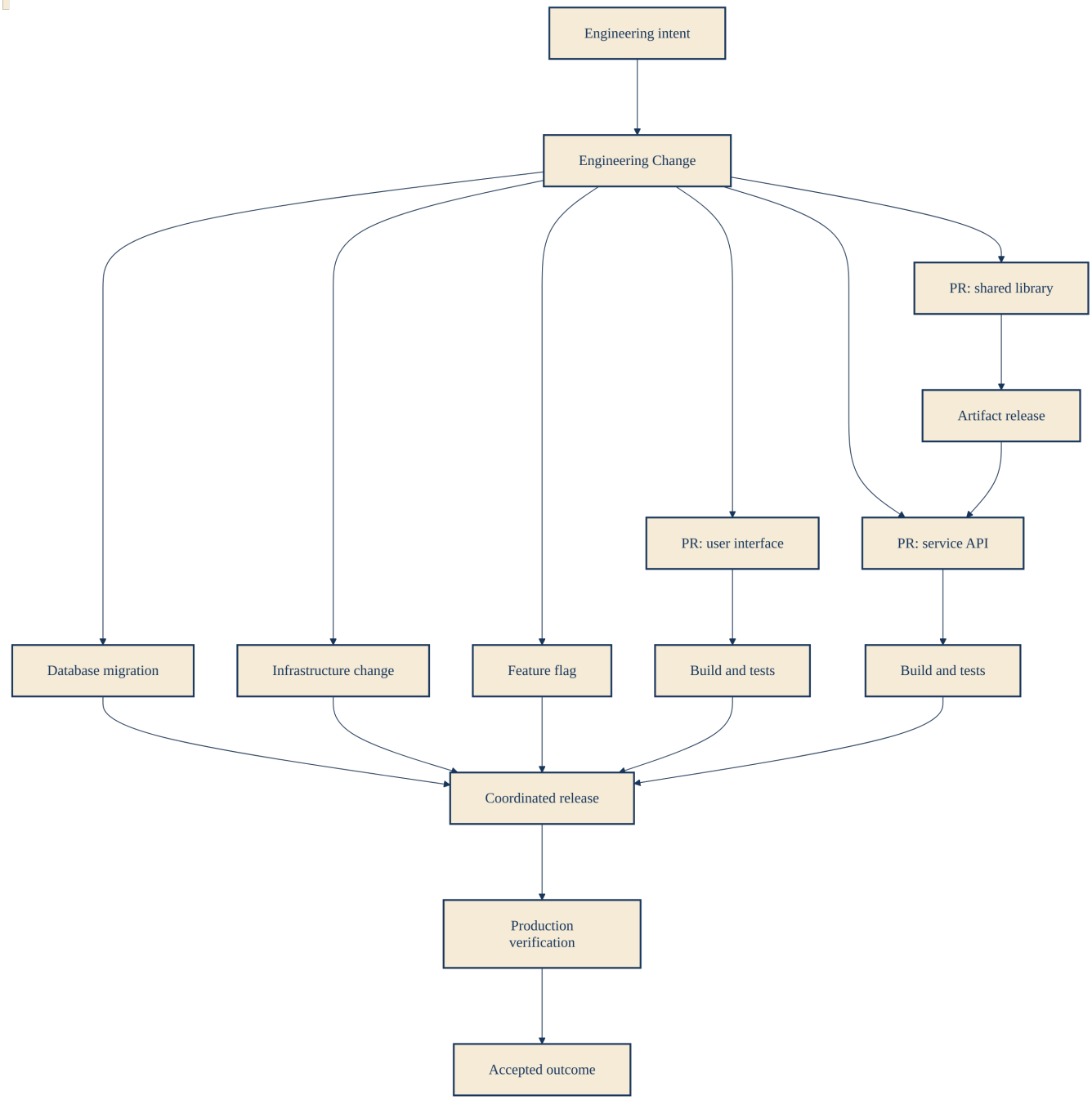
This book defines an **Engineering Change** as follows.



> An **Engineering Change** is the set of human, AI, code, configuration, data, infrastructure, review, verification, deployment, and operational activities required to move one coherent engineering intent from conception to an accepted outcome.

The accepted outcome may be production deployment, a completed migration, a security remediation, an internal platform capability, a retired system, or another explicitly defined terminal state. Not every Engineering Change produces user-visible code.

The relationship between intent and implementation is many-to-many:



This graph-oriented representation matters for multi-repository and multi-team work. Cost and latency should be attributed to the complete change graph, while local measures can still be retained for individual pull requests, builds, approvals, and deployments. The entity-activity-agent structure is compatible with general provenance models such as W3C PROV, while the Engineering Change model adds domain-specific semantics. See [moreau2013].

5.4.1. Change reconstruction

Engineering Changes may be declared explicitly through a change identifier, issue link, delivery plan, or release manifest. In many organizations, however, they must be reconstructed probabilistically from incomplete evidence. Useful linking signals include:

- issue and ticket identifiers in branches, commits, and pull requests;
- shared authors, reviewers, and time windows;
- dependency and build relationships;
- references in pull-request descriptions and comments;
- coordinated feature flags or release identifiers;
- service and repository ownership;
- deployment ordering;
- semantic similarity of titles and code changes;
- traces emitted by developer tools and AI agents.

The system should retain both the inferred relationship and its confidence. Silent certainty is dangerous: a falsely joined change can inflate apparent cost and blast radius, while a missed relationship fragments a costly coordinated effort into deceptively simple pull requests. Preserving provenance, versions, and replayability follows established data-system principles described in [moreau2013], [kleppmann2017], and [akidau2018].

5.5. Engineering is a socio-technical system

Software delivery is neither purely technical nor purely organizational. Outcomes emerge from interactions among people, tools, architecture, incentives, policies, and external constraints. This framing is consistent with the organizational and socio-technical foundations in [conway1968], [cataldo2006], and [herbsleb2007].

Consider a two-line configuration change. Its implementation complexity may be trivial. Yet the change could modify authentication behavior for every product, require review from three security teams, trigger a coordinated global deployment, and create a severe regulatory risk if incorrect. Code size is low; potential blast radius and coordination complexity are high.

A different change may add thousands of lines of generated client code. Its raw size is large, but its reasoning burden, novelty, and operational risk may be low.

Engineering Intelligence therefore represents a change as a multidimensional vector rather than a single scalar complexity score. Relevant dimensions include:

Dimension	Examples
Implementation complexity	Control flow, algorithmic novelty, concurrency, state management, language and framework difficulty
Domain complexity	Accounting rules, security semantics, distributed systems, machine learning, advanced user interaction, regulatory interpretation
Architectural complexity	Services touched, dependency depth, compatibility requirements, data contracts, shared libraries
Verification complexity	Test environments, nondeterminism, test-data requirements, manual validation, production-only behavior
Deployment complexity	Ordering constraints, migrations, flags, rollback paths, regional sequencing, long-running transitions

Dimension	Examples
Organizational complexity	Teams, ownership boundaries, approvals, external vendors, time zones, decision rights
Potential blast radius	Users, services, data, money, compliance obligations, availability, security boundaries
Novelty	Distance from patterns already understood by the author, reviewers, team, and organization

These dimensions influence both expected effort and uncertainty. They also help prevent invalid comparisons. A workflow intervention should not be judged by comparing raw cycle time across fundamentally different change populations. The modularity principles in [parnas1972] and the team cognitive-load perspective in [skelton2019] provide important foundations for understanding why boundaries and ownership affect change difficulty.

5.6. Human attention as an economic resource

Traditional cost accounting often starts with labor hours multiplied by loaded compensation. That is necessary but incomplete.

An hour of uninterrupted architectural work is not equivalent to twelve five-minute interruptions distributed across a day. A review request that arrives after the author has mentally left the task may require expensive context reconstruction. A senior security engineer's attention may be more constrained than the monetary cost assigned to the same amount of general engineering time. A queue of unresolved reviews can create attention residue even when no one is actively reviewing. These distinctions are consistent with cognitive-load theory, interruption research, and developer-productivity studies. See [sweller1988], [mark2008], and [meyer2014].

Engineering Intelligence therefore distinguishes at least four forms of human effort:

1. **Production effort** — creating code, designs, tests, configurations, and documentation.
2. **Verification effort** — reviewing, testing, validating, and challenging proposed work.
3. **Coordination effort** — aligning people, teams, schedules, ownership, and decisions.
4. **Recovery effort** — reconstructing context after delay, interruption, handoff, or failure.

The fourth category is frequently invisible. It appears as longer completion time, repeated reading, reopened files, duplicated analysis, forgotten assumptions, and hesitant decisions. It is one reason why elapsed time cannot be naively converted into labor cost.

The cost model developed later in this book will therefore include direct effort, queueing delay, context-recovery cost, scarce-expertise consumption, tool cost, infrastructure cost, expected quality loss, and cost of delay. The distinction between elapsed flow time, service time, utilization, and queue delay draws on queueing and operations research, including [little1961], [kingman1961], and [hopp2011].

5.7. AI as a first-class engineering participant

AI coding and review systems change both engineering work and the observability of that work.

An AI reviewer may identify a defect, generate a patch, explain a security issue, suggest a test, summarize a large change, or route a concern to a human expert. The human may accept the suggestion, modify it, reject it, ignore it, create a follow-up issue, or independently discover the same problem.

Treating the AI merely as a software license obscures its role. The AI participates in the engineering system and produces measurable work products and decision prompts. Engineering Intelligence therefore models AI contributions explicitly. Empirical work on AI coding assistants shows that interaction, usability, comprehension, debugging, and outcomes vary materially by task and context. See [peng2023], [vaithilingam2022], and [barke2023].

This does not mean equating model output with human judgment. It means recording the distribution of work:

- What did the AI inspect?
- What did it claim?
- How novel or redundant was the claim?
- What action did the human take?
- How much validation was required?
- Did the contribution alter the final change?
- Did it improve downstream quality?
- What did it cost in inference, latency, and human attention?

This framing creates the possibility of measuring **cognitive offload**: human reasoning or production work avoided because a reliable AI contribution performed it first. It also reveals **cognitive reload**: additional human work created by noisy, ambiguous, or untrustworthy AI output.

A high suggestion-acceptance rate is not sufficient evidence of offload. Developers may accept trivial suggestions, mechanically apply changes they do not understand, or ignore valuable comments because they lack time. Conversely, a rejected comment may still create value by prompting inspection. The correct object of study is the causal effect of AI participation on human effort and quality-adjusted outcomes. Repository-level agent benchmarks such as [jimenez2024] are useful for capability evaluation, but they do not replace organization-specific causal and production-outcome studies.

5.8. Measurement for decisions

Every operational metric in this book is evaluated through six questions:

1. **Purpose** — what decision should this metric inform?
2. **Definition** — what exactly is counted, and at what unit of analysis?
3. **Instrumentation** — which events and entities are required?
4. **Interpretation** — what mechanisms could cause it to move?
5. **Failure modes** — how can it be biased, gamed, or misunderstood?
6. **Actionability** — what intervention becomes possible when the metric changes?

This discipline prevents metric proliferation. Large enterprises can generate thousands of measures; only a subset deserve to become operational objectives. Measurement theory and validity research emphasize that a measure must be evaluated against its intended interpretation and use. See [stevens1946], [messick1995], and [hand2004].

Metrics will be classified along several axes:

- leading versus lagging;
- direct versus inferred;
- local versus system-level;
- diagnostic versus outcome;
- high-frequency versus slow-moving;
- individual, team, repository, service, portfolio, or enterprise scope;

- observational versus experiment-specific.

No individual-level metric should automatically be treated as a performance score. Measurements designed to understand system constraints can become harmful when repurposed to rank people. This changes behavior, damages trust, and invalidates the measurement process itself. See [campbell1979], [goodhart1984], and [messick1995].

5.9. The scientific method and engineering experimentation

Engineering organizations frequently introduce tools and policies globally, compare before-and-after averages, and attribute any movement to the intervention. This is rarely sufficient.

Tool adoption may coincide with seasonal release cycles, reorganizations, staffing changes, product launches, incidents, changes in repository mix, or broader adoption of generative AI. Teams choose tools non-randomly. Early adopters differ from late adopters. Measurement definitions change. Novelty effects decay. Work may be shifted outside the observed boundary. These are standard threats to causal identification and validity described in [shadish2002], [imbens2015], and [hernan2020].

Engineering Intelligence treats experimentation as infrastructure, not an occasional analytics project.

A strong evaluation specifies:

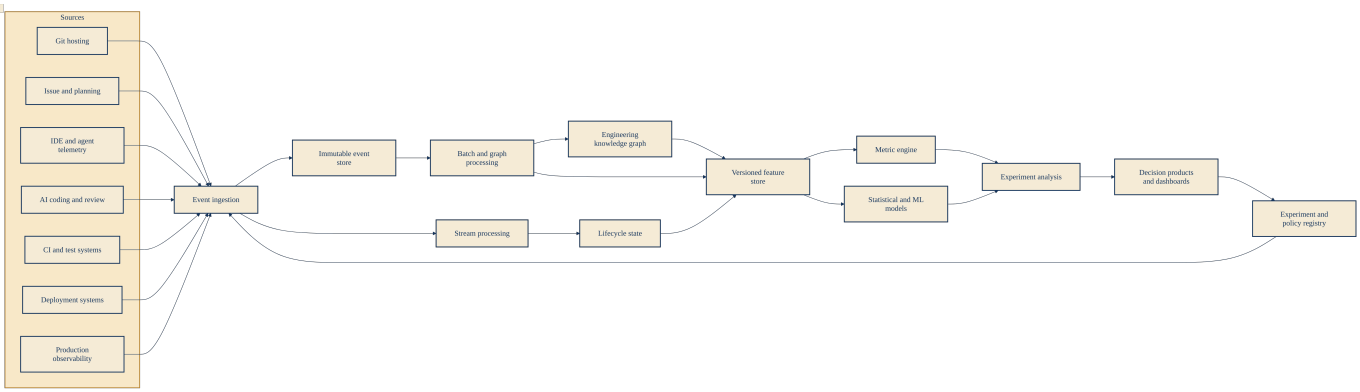
- the intervention and its mechanism;
- the unit of assignment;
- the unit of analysis;
- primary and guardrail outcomes;
- treatment exposure and compliance;
- pre-treatment covariates;
- expected effect size and statistical power;
- interference between treated and untreated units;
- stopping rules;
- heterogeneous effects;
- external events and concurrent changes.

Randomized controlled experiments are valuable when feasible, but software organizations often require alternatives: staggered rollouts, switchback experiments, difference-in-differences, matched controls, regression discontinuity, interrupted time series, synthetic controls, hierarchical models, and Bayesian sequential analysis. The methodological foundations are covered in [rubin1974], [holland1986], [shadish2002], [angrist2009], [imbens2015], [hernan2020], and [kohavi2020].

The objective is not statistical sophistication for its own sake. It is to avoid confident but incorrect conclusions about expensive organizational changes.

5.10. A reference architecture for Engineering Intelligence

A mature implementation separates event capture, identity resolution, change reconstruction, feature computation, causal analysis, and presentation.



Near-real-time events are appropriate for lifecycle state, queue depth, tool interactions, CI results, and deployment transitions. Batch computation is appropriate for repository topology, dependency graphs, historical baselines, code-complexity features, ownership networks, and model retraining. Some derived outcomes, such as realized defects or incident impact, arrive days or weeks after a merge.

The architecture must therefore support event time, late-arriving data, changing entity relationships, historical recomputation, and versioned metric definitions. A number displayed without the definition and model version that produced it is not reproducible evidence. These architectural requirements are grounded in event-log, stream-processing, provenance, and reliability principles described in [kleppmann2017], [akidau2018], [moreau2013], and [beyer2016].

5.11. Scope of this book

The remainder of the book develops five connected layers.

Foundations define the Engineering Change, engineering economics, value, cognition, AI participation, complexity, and blast radius.

Measurement provides a catalogue of direct and derived measures for flow, quality, cost, attention, coordination, architecture, and AI contribution.

Instrumentation describes event schemas, identity resolution, stream and batch architecture, knowledge graphs, feature stores, data quality, privacy, and governance.

Experimentation covers randomized and quasi-experimental designs, causal inference, power, sequential testing, heterogeneous effects, and external confounding factors.

Applications demonstrates how the framework can evaluate AI review tools, merge queues, branch protection, stacked changes, repository strategies, CI optimizations, autonomous agents, and portfolio-level engineering investments.

The goal is practical. Every major construct must be measurable, useful for a decision, and capable of empirical validation.

5.12. Key takeaways

- Engineering organizations do not primarily suffer from a lack of metrics; they suffer from weak models connecting activity, attention, quality, cost, and value.
- The Engineering Change—not the individual pull request—is the most useful primary unit for multi-repository, multi-team, and deployment-aware analysis.
- Human attention must be modeled as a scarce economic resource, including context recovery and coordination costs.

- AI should be observed as a participant whose contributions can create both cognitive offload and cognitive reload.
- Metrics should exist to support decisions and experiments, not to maximize dashboard coverage.
- Engineering Intelligence requires a closed loop from observation through causal evaluation to organizational learning.

5.13. Open questions

- How accurately can Engineering Changes be reconstructed when explicit identifiers are absent?
- Which behavioral signals best predict cognitive burden without invasive surveillance?
- How should scarce expert attention be valued when market compensation is a poor proxy for organizational constraint?
- How can AI-generated work be attributed when humans and multiple agents iteratively transform the same artifact?
- Which experiment designs best handle interference across teams sharing repositories, reviewers, and platforms?

5.14. Implementation checklist

1. Establish explicit principles for ethical use of engineering telemetry.
2. Define the first version of the Engineering Change entity and its identifiers.
3. Inventory event sources across source control, CI, issue tracking, AI tools, deployment, and production.
4. Separate raw observations from derived metrics and version every definition.
5. Select a small set of decisions the measurement system must improve.
6. Define baseline outcome and guardrail metrics before introducing new workflow tools.
7. Instrument AI suggestions and human responses as linked events rather than aggregate acceptance counts.
8. Create an experimentation registry that records interventions, exposure, timing, and concurrent external changes.

5.14.1. Research foundations and source notes

This introductory chapter establishes the scope and decision principles of Engineering Intelligence. It combines established measurement, software-delivery, developer-experience, cognitive-load, organizational, and causal-learning research with an original synthesis that treats engineering as an observable socio-technical and economic system.

Measurement validity and intended use

The distinction between a reproducible number and a valid interpretation is grounded in measurement theory. Messick's unified account of construct validity emphasizes that validity concerns the evidential and consequential basis for interpreting and using measurements. Hand similarly cautions that measurement is a model-based act rather than a neutral reading of reality. See [messick1995] and [hand2004].

These foundations support the chapter's central claim that a metric is not inherently useful or valid. Its meaning depends on the construct, population, boundary, context, and decision for which it is used.

Software delivery performance

The chapter's discussion of deployment frequency, lead time for changes, change-failure rate, and restoration time builds on the software-delivery research synthesized in [forsgren2018].

That body of work supports using delivery outcomes as system-level indicators, but it does not justify converting them into universal individual targets. Engineering Intelligence therefore treats them as contextual outcomes that must be interpreted alongside architecture, work type, risk, quality, and organizational constraints.

Multidimensional developer productivity

The rejection of a single universal productivity score is consistent with the SPACE framework. SPACE organizes developer productivity across satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow. See [forsgren2021-space].

The chapter extends this multidimensional view by explicitly adding economic cost, human attention, AI participation, architectural complexity, operational risk, and causal evaluation. These additions are an Engineering Intelligence synthesis rather than a replacement for SPACE.

Developer experience, flow, and feedback

The emphasis on feedback loops, cognitive load, progress, and flow is informed by developer-experience and software-engineering research. See [noda2023-devex], [fagerholm2012], and [meyer2014].

This literature supports treating delays, interruptions, unclear ownership, fragmented tooling, and slow feedback as substantive properties of the engineering system rather than as individual shortcomings.

Cognitive load and interruption

The chapter treats expert attention as a constrained organizational resource. Cognitive-load theory provides a foundation for distinguishing the intrinsic complexity of a task from additional burden imposed by presentation, process, or tooling. See [sweller1988].

Research on interrupted work shows that resumption, context reconstruction, and fragmented attention can impose costs that are not visible in simple activity counts. See [mark2008]. These findings support the later distinction between elapsed time, active effort, waiting, and cognitive burden.

Metric gaming and target distortion

The warning against turning descriptive measures directly into targets is grounded in Campbell's Law and the family of ideas commonly associated with Goodhart's Law. See [campbell1979] and [goodhart1984].

These principles do not imply that targets are always inappropriate. They imply that measures used for control or reward should be treated as interventions that can change behavior, data quality, and the system being observed. Guardrails, counter-metrics, qualitative review, and periodic revalidation are therefore necessary.

Socio-technical systems and organizational structure

Engineering Intelligence treats software delivery as a socio-technical system rather than a sequence of isolated developer actions. Conway's observation about the relationship between communication structures and system design provides one foundation for this view. See [conway1968].

Later chapters expand this foundation through socio-technical congruence, coordination, modularity, team cognitive load,

and architecture. The introductory claim is deliberately broad: engineering outcomes emerge from interactions among people, tools, code, architecture, process, incentives, and organizational boundaries.

Learning systems and causal evaluation

The chapter's observe–reconstruct–model–explain–experiment–learn loop is an original Engineering Intelligence operating model. Its scientific orientation is consistent with established causal-inference and experimentation practice developed later in the book.

The model distinguishes descriptive monitoring from organizational learning. A dashboard can report a change; a learning system must formulate explanations, test interventions, estimate effects, retain uncertainty, and update future decisions.

Original synthesis introduced in this chapter

The following concepts are original organizing constructs of this book rather than established external standards:

- **Engineering Intelligence** as an integrated discipline spanning engineering economics, attention, architecture, delivery, quality, AI participation, governance, and experimentation;
- the **Engineering Change** as a cross-system unit of analysis broader than a pull request;
- the closed Engineering Intelligence learning loop;
- the explicit treatment of expert attention as a scarce operational and economic resource;
- the requirement that engineering measurement be tied to a decision, intervention, or learning objective;
- the separation of observation, reconstruction, modeling, explanation, experimentation, and organizational learning.

These constructs are intended to be testable and adaptable. They should not be interpreted as claims that one universal data model, metric set, or maturity path fits every organization.

Limits of the evidence

The research cited in this chapter spans different methods, populations, organizations, technologies, and time periods. Several limitations follow:

- delivery-performance associations do not establish that optimizing one metric in isolation will cause improvement;
- developer-experience findings may depend on team, task, toolchain, and organizational context;
- cognitive-load and interruption findings do not provide a universal conversion from interruption counts to economic loss;
- Goodhart- and Campbell-style effects identify risks, not deterministic outcomes;
- broad socio-technical relationships do not determine the best organizational design for a particular company;
- the integrated Engineering Intelligence framework remains a synthesis whose value must be evaluated through use, replication, and revision.

The chapter therefore uses these sources as foundations for measurement design and cautious inference, not as justification for deterministic scoring or universal benchmarks.

Chapter 6. Engineering Economics



***Engineering organizations make economic decisions continuously: whether to automate a** review step, split a repository, add a required approver, buy an AI tool, tolerate a slower build, defer a migration, or invest in platform work. Yet these decisions are often made without a shared cost model. This chapter defines the economic foundations of Engineering Intelligence. It separates direct expenditure from human attention, delay, coordination, risk, and quality costs; introduces a formal cost model for an Engineering Change; explains why averages are often misleading; and provides practical methods for estimation, calibration, and decision support.*

6.1. Why cost is harder than it appears

At first glance, the cost of a pull request seems straightforward. Count the time spent by the author and reviewers, multiply by loaded compensation, add the software and infrastructure costs, and report the total. This calculation is useful, but it is not a complete economic model.

Several important costs remain hidden:

- The author may stop active work while waiting for review, then pay a context-reconstruction cost when comments arrive.
- A review request may consume scarce security expertise whose organizational constraint is much greater than its nominal hourly cost.

- A required approval may add little direct labor but create days of queueing delay.
- A noisy AI reviewer may reduce authoring effort while increasing verification effort.
- A change may be cheap to implement but expensive to coordinate across repositories and deployment boundaries.
- A defect avoided before merge creates expected value even though the avoided incident never appears in operational data.
- A delayed change may postpone revenue, risk reduction, learning, or customer benefit.

The correct economic question is not simply, "How much labor was recorded?" It is:

> What resources did this Engineering Change consume, what risks did it create or retire, what value did it enable, and what alternative uses of those resources were displaced?

This is an activity-based view of engineering economics. Costs are attributed to the activities and constraints that created them rather than being inferred from a single duration or output count.

6.2. Cost, price, value, and investment

Four terms must be kept separate.

Cost is the consumption of resources associated with an Engineering Change or process.

Price is the amount paid to acquire a tool, service, or labor input. Price may be easy to observe while cost remains hidden. A low-priced tool can create expensive human validation work.

Value is the benefit created or protected by the outcome. Value may include revenue, retention, reduced operational risk, compliance, faster learning, strategic option value, or avoided future effort.

Investment is current cost incurred to improve future capability. Platform engineering, test infrastructure, codebase simplification, and knowledge transfer are often investments rather than immediate feature-delivery costs.

Confusing these terms leads to poor optimization. A process can become cheaper while destroying value. A change can be expensive while producing exceptional returns. A platform investment can reduce current throughput while increasing future capacity.

6.3. The total cost of an Engineering Change

Let an Engineering Change be denoted by e. Its total economic cost can be represented as:

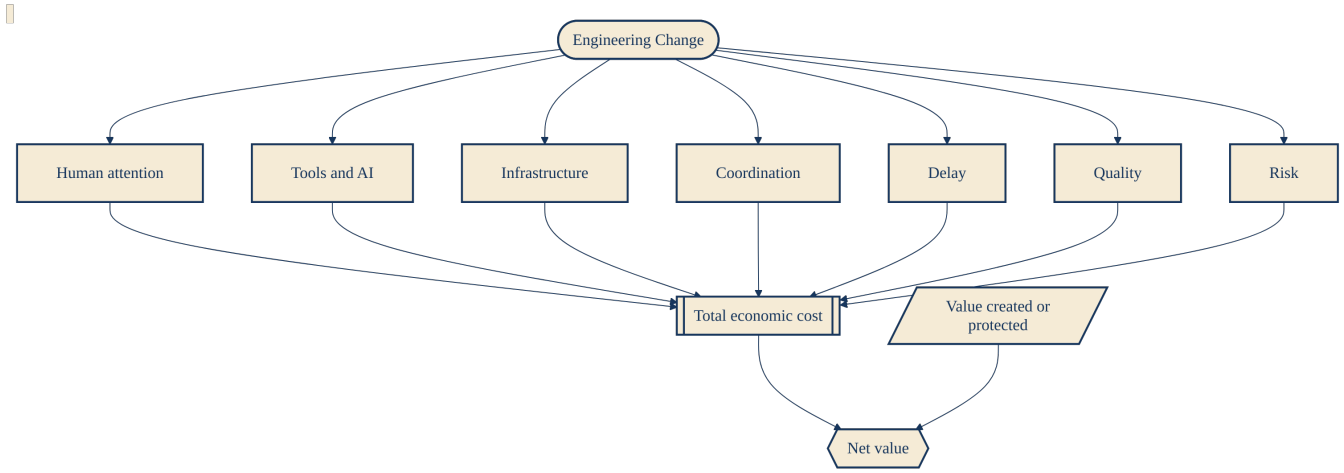
$$C_{total}(e) = C_{human}(e) + C_{tool}(e) + C_{infra}(e) + C_{coord}(e) + C_{delay}(e) + C_{quality}(e) + C_{risk}(e)$$

where:

- C_{human} is the cost of human attention and effort;
- C_{tool} is the marginal and allocated cost of software and AI services;
- C_{infra} is compute, storage, networking, test-environment, and observability cost;
- C_{coord} is the cost of cross-person, cross-team, and cross-system alignment;
- C_{delay} is the economic cost of waiting and deferred benefit;
- $C_{quality}$ is expected rework, defects, incidents, and support cost;
- C_{risk} is the expected cost of low-probability, high-impact outcomes not fully represented by observed defects.

This decomposition is intentionally additive for explanation, but the components are not always independent. A tool may reduce authoring time while increasing quality. More review may increase direct human cost while reducing expected defect cost. A slower rollout may increase delay but reduce operational risk.

A useful model therefore records both component values and the assumptions connecting them.



6.4. Human effort cost

The simplest human-cost estimate is:

$$C_{labor}(e) = \sum_{p \in P_e} h_{p,e} \times r_p$$

where P_e is the set of participants, $h_{p,e}$ is the number of hours participant p spent on the change, and r_p is that participant's loaded hourly cost.

Loaded cost commonly includes salary, benefits, payroll taxes, facilities, management overhead, and other allocated expenses. Organizations may use standardized rates by role or level to reduce sensitivity and protect individual compensation data.

This estimate is useful for budgeting, but three refinements are required.

6.4.1. Active time versus elapsed time

Elapsed time cannot be treated as labor time. A pull request open for forty hours did not necessarily consume forty engineer-hours. Event data should distinguish:

- active authoring;
- active review;
- active response and rework;
- automated processing;
- queue time;
- blocked time;
- inactive calendar time.

Because perfect time tracking is neither realistic nor desirable, active effort is usually estimated from event sequences,

IDE activity, review sessions, code updates, build interactions, and calibrated task-duration models.

An effort estimate should be reported with uncertainty. For example:

$$\hat{H}_{author}(e) = 6.2 \text{ hours}, \quad 90\% \text{ interval} = [4.7, 8.5]$$

False precision undermines trust. A modeled estimate of 6.237 hours communicates more certainty than the data supports.

6.4.2. Attention-weighted effort

Not all hours have equal opportunity cost. One hour from a broadly available contributor may be less constraining than one hour from the only engineer authorized to approve a cryptographic change.

Define an attention scarcity multiplier s_p for participant or role p:

$$C_{attention}(e) = \sum_{p \in P_e} h_{p,e} \times r_p \times s_p$$

The multiplier is not a judgment of personal worth. It represents organizational scarcity. It can be estimated from queue depth, demand-to-capacity ratio, reviewer utilization, time-to-assignment, and the number of qualified substitutes.

For example, if database-migration reviewers receive twice as much demand as available review capacity, their marginal attention may constrain delivery more than their accounting rate suggests.

6.4.3. Context recovery and interruption cost

When work is interrupted, resumption often requires rereading code, rebuilding assumptions, restoring local state, and recovering unresolved decisions. Let n_{switch} be the number of material context switches and $t_{recover}$ the expected recovery time per switch:

$$C_{recovery}(e) = n_{switch}(e) \times t_{recover}(e) \times r_{effective}$$

This is a model, not a direct observation. Relevant proxies include:

- time between review request and author response;
- number of unrelated repositories or tasks touched during the wait;
- repeated file navigation after resumption;
- repeated builds or test runs without code changes;
- reopening of previously resolved discussions;
- multiple partially active branches;
- change in active issue or repository before return.

These signals must be used carefully. The goal is to estimate system-level friction, not to monitor individual behavior minute by minute.

6.5. Tool and AI cost

Tool cost includes both fixed and variable components.

Let a tool j have annual fixed cost F_j, usage-based cost $u_{j,e}$, and allocated population N_j. A simple per-change allocation is:

$$C_{tool}(e) = \sum_j \left(\frac{F_j}{N_j} \times a_{j,e} + u_{j,e} \right)$$

where $a_{j,e}$ is an allocation factor representing the share of licensed capacity attributed to the change.

Per-seat allocation is often sufficient for financial reporting but weak for experiment analysis. To compare two AI review workflows, capture marginal usage:

- model and model version;
- input and output tokens;
- number of review passes;
- repository indexing cost;
- embeddings and retrieval cost;
- tool-call count;
- latency;
- cache hits;
- retry and failure cost.

The full economic cost of an AI tool must also include the human work it creates:

$$C_{AI,net}(e) = C_{inference}(e) + C_{validation}(e) + C_{administration}(e) - V_{offload}(e)$$

where $V_{offload}$ is the value of human effort genuinely avoided or quality improvement created.

A cheap model producing many low-value comments can be more expensive than a costly model producing a small number of trusted findings. Cost per comment is therefore rarely useful. Cost per validated defect prevented, or cost per hour of quality-adjusted human attention saved, is more meaningful.

6.6. Infrastructure cost

Infrastructure cost includes more than cloud invoices. A change can trigger builds, tests, ephemeral environments, security scans, artifact storage, deployment waves, and production observation.

For Engineering Change e:

$$C_{infra}(e) = C_{build} + C_{test} + C_{env} + C_{artifact} + C_{deploy} + C_{observe}$$

Useful event-level inputs include:

- runner type and duration;
- CPU, GPU, memory, and storage consumption;
- cache effectiveness;
- test-shard count;
- environment lifetime;
- artifact size and retention;
- deployment target count;
- log, metric, and trace volume.

Shared infrastructure requires allocation rules. The rule should match the decision. For cost optimization, marginal cloud cost may be appropriate. For total-cost comparison, allocated platform and staffing cost may be needed. Mixing marginal

and fully allocated cost in the same analysis produces misleading results.

6.7. Coordination cost

Coordination cost is often the dominant hidden cost of large enterprise changes.

It arises from:

- identifying owners;
- scheduling reviewers;
- resolving conflicting requirements;
- negotiating interfaces;
- coordinating release order;
- obtaining approvals;
- communicating status;
- managing dependencies;
- handling handoffs;
- aligning across time zones and organizations.

A simple coordination model might use:

$$C_{coord}(e) = \alpha T_e + \beta B_e + \gamma H_e + \delta A_e$$

where:

- T_e is the number of teams materially involved;
- B_e is the number of ownership or approval boundaries;
- H_e is the number of handoffs;
- A_e is active coordination effort;
- $\alpha, \beta, \gamma, \delta$ are empirically calibrated coefficients.

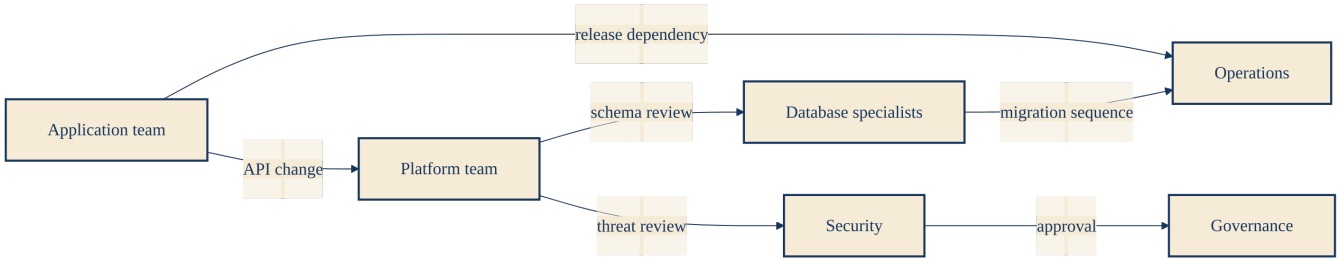
This should not be interpreted as a universal formula. It is an example of a model whose coefficients can be learned from historical changes.

Coordination cost is nonlinear. Adding a second team may be inexpensive if the relationship is routine and interfaces are stable. Adding a fifth team can increase the number of communication paths and scheduling constraints dramatically.

The theoretical number of pairwise communication paths among n participants is:

$$L(n) = \frac{n(n-1)}{2}$$

Real organizations do not require every participant to communicate directly, but the equation illustrates why coordination structures, clear ownership, and interface contracts matter.



6.8. Delay and cost of delay

Waiting is not free merely because no one is actively working.

A change delayed in review may postpone:

- customer value;
- revenue;
- risk reduction;
- regulatory compliance;
- operational savings;
- learning from production;
- dependent work;
- retirement of an expensive legacy system.

Define the instantaneous value-delivery rate of change e as $v_e(t)$. If delivery is delayed from time t_0 to t_1, the cost of delay is:

$$C_{delay}(e) = \int_{t_0}^{t_1} v_e(t)\,dt$$

In practice, value curves are rarely known precisely. Organizations can use categories:

Delay profile	Typical behavior
Standard	Value is approximately linear over the relevant window.
Fixed date	Value drops sharply after a regulatory, contractual, seasonal, or launch deadline.
Expedite	Delay creates immediate and unusually high cost, as with an active incident or critical vulnerability.
Intangible or exploratory	Value comes primarily from learning; delay postpones information rather than a known revenue stream.

Cost of delay should not be applied indiscriminately to manufacture urgency. Its purpose is prioritization and trade-off analysis. When uncertain, report a range and sensitivity analysis.

6.8.1. Queueing and inventory cost

Open Engineering Changes behave like work-in-process inventory. They consume attention, create dependency uncertainty, age, and can require context recovery.

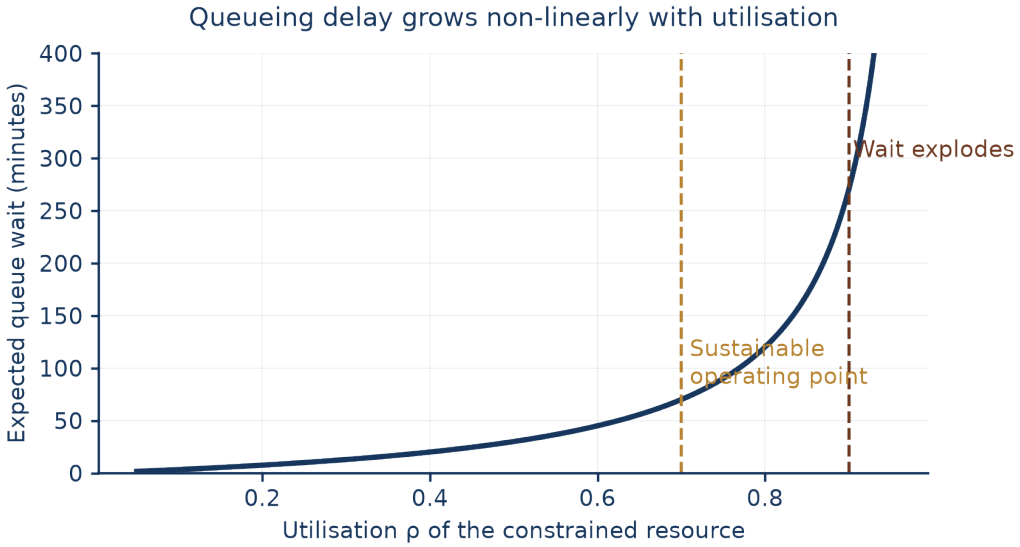
Little's Law relates average work in process L, throughput λ, and average flow time W:

$$L = \lambda W$$

If a team completes ten Engineering Changes per week and average flow time is three weeks, average work in process is approximately thirty changes, assuming a stable system and consistent boundaries.

This relationship does not prescribe a target, but it exposes trade-offs. Increasing work in process without increasing throughput raises flow time. Reducing batch size and limiting concurrent work can reduce latency even if individual task execution speed does not change.

The cost of queueing at a constrained resource is not linear in utilization. As Figure 3 shows, the expected wait time of an M/M/1 queue rises gently until roughly 70% utilization and then explodes as the resource approaches saturation. A review queue or CI cluster operating at 90% average load is not "a bit slower" than one at 70%: it is qualitatively different, and small variance in arrival rate produces the long tail of delayed changes that dominate lead-time distributions.



A carrying-cost approximation is:

$$C_{carry}(e) = W_e \times k_e$$

where W_e is time in process and k_e is an estimated daily carrying cost incorporating delay, coordination overhead, and context risk.

6.9. Quality cost

Quality cost should be modeled prospectively and retrospectively.

Retrospective quality cost includes observed rework, rollback, support, incident response, customer remediation, and follow-up changes.

Prospective quality cost is expected loss:

$$C_{quality}^{expected}(e) = \sum_{f \in F_e} P(f \mid e) \times I(f)$$

where F_e is the set of relevant failure modes, $P(f \mid e)$ is the probability of failure f given the change, and $I(f)$ is its impact.

The expected-cost model allows review and testing investments to be evaluated economically. Suppose an additional security review costs $600 and reduces the estimated probability of a $200,000 loss from 1.0% to 0.4%. The expected loss reduction is:

$$(0.010 - 0.004) \times 200{,}000 = 1{,}200$$

Under those assumptions, the review produces $1,200 of expected risk reduction for $600 of direct cost. The estimate is uncertain, but the structure makes the decision explicit.

6.9.1. Prevention, appraisal, and failure

A useful classification from quality economics separates:

- **prevention cost** — design practices, training, automation, and architecture intended to prevent defects;
- **appraisal cost** — review, testing, scanning, and verification;
- **internal failure cost** — defects found before release;
- **external failure cost** — defects found after release.

Optimization does not mean minimizing appraisal. Excessively reducing review may lower immediate cost while increasing external failure. Excessive appraisal may create delay with little incremental risk reduction. The objective is to minimize total quality-adjusted cost.

6.10. Risk and tail events

Observed defect rates often underrepresent security, privacy, financial, and availability risk because severe events are rare. Historical averages can therefore create false confidence.

Risk analysis should combine:

- empirical incident frequency;
- threat and failure-mode analysis;
- potential blast radius;
- control effectiveness;
- expert priors;
- scenario analysis;
- stress testing;
- simulation.

For rare events, expected value alone can be inadequate. Two changes may have the same expected loss while one carries catastrophic tail risk. Decision support may therefore include quantiles, conditional value at risk, or explicit policy constraints.

A change violating a regulatory requirement is not made acceptable merely because its estimated expected loss is low.

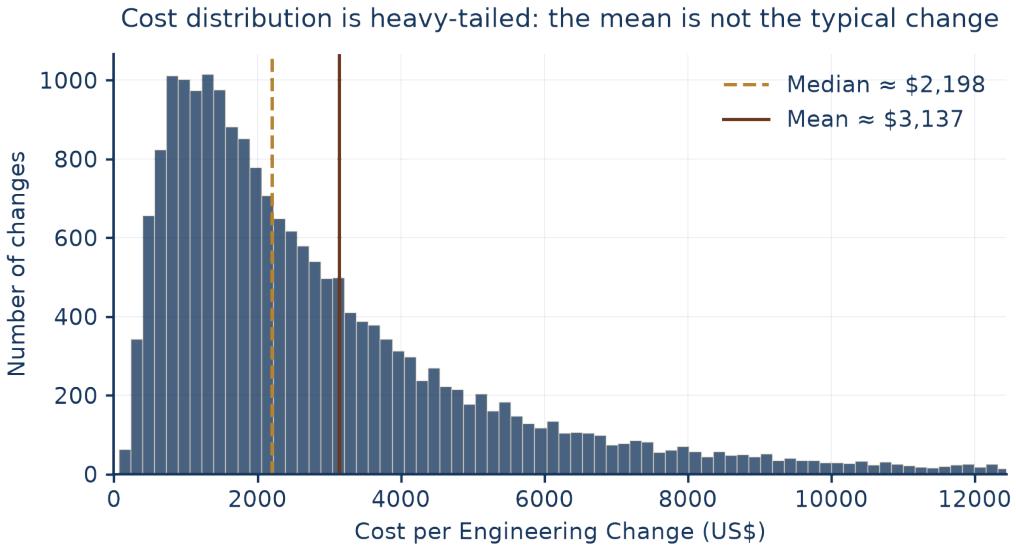


Figure 4 illustrates the same phenomenon for the cost of an Engineering Change: the distribution is right-skewed, and the mean lies well above the median. A policy tuned to the average change under-invests in the tail; a policy tuned to the tail over-invests in the routine. Reporting both statistics, together with the shape of the distribution, avoids either failure mode.

6.11. Value and return

Cost analysis becomes dangerous when disconnected from value.

Let $V(e)$ be the discounted value created by Engineering Change e. A basic net value is:

$$NV(e) = V(e) - C_{total}(e)$$

Return on engineering investment can be expressed as:

$$ROEI(e) = \frac{V(e) - C_{total}(e)}{C_{total}(e)}$$

These formulas are simple; estimating value is not.

Engineering value commonly falls into six categories:

1. **Growth value** — revenue, acquisition, conversion, retention, and market expansion.
2. **Efficiency value** — avoided labor, infrastructure savings, lower support cost, and faster delivery.
3. **Risk value** — reduced probability or impact of failure, fraud, security compromise, or noncompliance.
4. **Learning value** — information obtained through experiments, prototypes, instrumentation, and production feedback.
5. **Capability value** — reusable platforms, skills, data, architecture, and tooling that improve future work.
6. **Option value** — preserving the ability to make future choices under uncertainty.

Not every change should be forced into a precise dollar value. The purpose of valuation is to improve comparisons, expose assumptions, and support portfolio decisions. Qualitative value classes and bounded estimates are preferable to fabricated precision.

6.12. Worked example: evaluating an AI review tool



Runnable versions of this example live in the companion notebooks: * `companion/notebooks/01-queueing-cost.py` * `companion/notebooks/13-monte-carlo-cost.py`

Consider an organization processing 100,000 pull requests per year. It is evaluating an AI reviewer with an annual license and infrastructure cost of $1.2 million.

The observed treatment effects from a controlled rollout are:

- average human review effort decreases by 7 minutes per pull request;
- author response and validation effort increases by 2 minutes;
- post-merge defect probability falls from 2.0% to 1.8%;
- average review latency falls by 45 minutes;
- the average loaded engineering cost is $150 per hour;
- the average cost of a qualifying post-merge defect is $3,000.

Net human time saved per pull request is five minutes:

$$100{,}000 \times \frac{5}{60} \times 150 = 1{,}250{,}000$$

Expected defect savings are:

$$100{,}000 \times (0.020 - 0.018) \times 3{,}000 = 600{,}000$$

Ignoring delay value, total estimated benefit is $1.85 million. Subtracting the $1.2 million tool cost yields an estimated annual net benefit of $650,000.

This result is incomplete until uncertainty and heterogeneity are considered. The effect may differ by language, repository, change complexity, team experience, or comment type. The defect-cost distribution may be heavy-tailed. Adoption may be incomplete. The seven-minute review reduction may be concentrated in trivial changes.

A publication-quality evaluation should therefore report:

- confidence or credible intervals;
- treatment compliance;
- primary and guardrail outcomes;
- effects by relevant change classes;
- sensitivity to defect-cost assumptions;
- inference and platform costs;
- long-term effects after novelty and learning periods.

The point-estimate of $650,000 is a starting hypothesis, not a conclusion. Reported in isolation it is nearly indistinguishable from an educated guess; reported with the accompaniments above, it becomes a claim that other engineering leaders can inspect, replicate, or dispute. Chapter 6 develops the causal machinery required to turn such an evaluation into evidence that survives contact with an audit or a change of leadership.

6.13. Worked example: a required specialist approval



Runnable version: `companion/notebooks/01-queueing-cost.py`.

A platform team proposes requiring a database specialist to approve every schema-changing pull request.

Historical data shows 4,000 such changes per year. Each approval requires an average of 12 specialist minutes and creates an average queueing delay of 5.5 hours. The specialist's loaded cost is $190 per hour. The approval is estimated to prevent eight production incidents annually, each costing an average of $45,000.

Direct review cost:

$$4{,}000 \times \frac{12}{60} \times 190 = 152{,}000$$

Expected incident savings:

$$8 \times 45{,}000 = 360{,}000$$

On direct cost and expected quality benefit, the policy appears favorable. But the queueing delay affects 4,000 changes, and the specialist becomes a single constrained resource. The correct decision may be to retain specialist review only for high-risk migrations while automating classification and standard checks for routine changes.

This illustrates a recurring pattern: the optimal solution is often not "add review" or "remove review," but route scarce attention according to predicted risk and uncertainty.

6.14. Estimation methods

Different cost components require different methods.

Component	Primary evidence	Common method
Human active effort	IDE, SCM, review, build, and calibrated task samples	Session reconstruction and hierarchical duration models
Tool cost	Contracts, billing, token and request telemetry	Direct allocation and marginal-cost calculation
Infrastructure cost	Runner, cloud, storage, deployment, and observability telemetry	Usage attribution and cost allocation
Coordination cost	Teams, handoffs, approvals, messages, dependencies	Activity models and graph-derived features
Delay cost	Priority, product economics, deadlines, dependency effects	Value curves and sensitivity analysis
Quality cost	Rework, defects, incidents, support, customer impact	Expected-loss models and retrospective attribution

Component	Primary evidence	Common method
Risk	Threat models, control evidence, incident history, expert judgment	Bayesian models, scenarios, and simulation

6.14.1. Calibration samples

Automated models should be calibrated against small, carefully designed samples. This does not require enterprise-wide surveys or invasive time sheets. Useful calibration methods include:

- short-term voluntary work diaries for representative populations;
- structured observation of selected changes;
- expert labeling of activity sequences;
- controlled tasks with known timing;
- post hoc reconstruction during incident or project reviews;
- comparison with independently recorded pair-programming or meeting sessions.

Calibration data should be refreshed when workflows or tools change. An effort model trained before widespread AI adoption may systematically misestimate AI-assisted work.

6.14.2. Hierarchical models

Enterprise data is nested: changes within repositories, repositories within teams, teams within organizations. A hierarchical model can separate local variation from organization-wide effects.

For example, active review time may be modeled as:

$$\log H_{review,e} = \beta_0 + \beta^T X_e + u_{team(e)} + v_{repo(e)} + \epsilon_e$$

where X_e contains change-level features, u is a team-level effect, and v is a repository-level effect.

This prevents a small repository from receiving an unstable estimate based only on a handful of changes and allows partial pooling across similar populations.

6.15. Marginal versus average economics

Averages answer questions about total allocation. Marginal values answer questions about change.

The average cost of a CI system includes fixed platform engineering and infrastructure. The marginal cost of one additional test run may be only compute and storage. When deciding whether to run a new test, marginal cost is often appropriate. When comparing operating models or vendors, total allocated cost matters.

Likewise, reducing average review time by ten minutes does not necessarily free ten minutes of usable capacity. The saved fragments may be too dispersed to reallocate. A constrained reviewer queue may show a large nonlinear benefit from a small capacity increase, while an unconstrained queue may show almost none.

Economic analyzes should state explicitly whether they use:

- marginal cost;
- avoidable cost;

- fully allocated cost;
- replacement cost;
- opportunity cost.

Choosing incorrectly is the more common failure mode than computing incorrectly. A team that reports "we saved five million dollars" using fully allocated cost is describing a spreadsheet reallocation, not real cash freed for other work. A team that treats avoidable cost as marginal will systematically under-invest in capacity because it counts the savings but not the queuing that produced them. Naming the cost category next to every economic claim is therefore the smallest change with the largest effect on the honesty of the resulting decision.

6.16. Uncertainty and sensitivity analysis

Engineering economic models contain uncertain inputs. The proper response is not to hide uncertainty but to propagate it.

Suppose annual benefit is:

$$B = N \times (m_s - m_v) \times r + N \times (p_c - p_t) \times d$$

where N is annual volume, m_s is minutes saved, m_v is validation minutes added, r is hourly labor cost, p_c and p_t are control and treatment defect probabilities, and d is defect cost.

Each parameter can be represented as a distribution rather than a point. Monte Carlo simulation can then estimate the probability that benefit exceeds cost.

A decision report might state:

- median annual net benefit: $650,000;
- 80% credible interval: -$120,000 to $1.5 million;
- probability of positive net benefit: 74%;
- most influential uncertainty: average cost of escaped defects.

This is more useful than a single number because it identifies which additional evidence would improve the decision.

6.17. Common failure modes

6.17.1. Monetizing everything

Not every important outcome can be credibly converted into dollars. Forced monetization can create artificial certainty and hide ethical or strategic constraints. Use qualitative classes, ranges, or guardrails when necessary.

6.17.2. Treating salary as value

Compensation is an input price, not the value created by an engineer. A lower-cost hour is not automatically a better hour, and a high-cost engineer is not automatically a poor economic choice.

6.17.3. Ignoring displaced work

A review process may appear cheap because the cost of waiting and task switching is assigned nowhere. A tool may appear productive because verification work moves from reviewers to authors.

6.17.4. Double counting

Time savings, increased throughput, and cost-of-delay benefits can describe overlapping value. Counting all three independently may inflate benefits.

6.17.5. Optimizing component cost

Reducing CI expense by running fewer tests may increase failure cost. Reducing review time may increase rework. Local optimization must be evaluated against total system cost and guardrail outcomes.

6.17.6. Using cost metrics for individual ranking

Cost estimates designed for process analysis are not reliable measures of individual performance. People receive work of different complexity, perform invisible mentoring and coordination, and operate under different constraints. Ranking individuals creates gaming and corrupts the data-generating process.

6.18. Designing an economic scorecard

A practical scorecard should avoid collapsing everything into one number. Recommended views include:

View	Question answered
Cost composition	Where is money and attention consumed?
Flow and delay	Where does work wait, and what is the estimated carrying cost?
Quality-adjusted cost	What is the total cost after rework and expected failure?
Scarce-attention use	Which roles or expertise pools constrain the system?
Tool economics	What is the net effect of each tool on labor, quality, latency, and infrastructure?
Change-class economics	How do costs differ by complexity, blast radius, repository topology, and domain?
Experiment economics	What causal effect did an intervention have, with what uncertainty?

Executive reporting should emphasize distributions, trends, and decisions rather than league tables. Operational reporting should permit drill-down to the Engineering Change graph and raw evidence while respecting privacy and access controls.

6.19. Key takeaways

- The cost of an Engineering Change includes human attention, tools, infrastructure, coordination, delay, quality, and risk.
- Elapsed time is not labor time; queueing and context recovery must be modeled separately.
- Human attention should be valued by both loaded cost and organizational scarcity.
- AI economics must include validation burden and quality effects, not merely license or inference cost.
- Cost of delay connects engineering flow to business impact, but should be estimated with ranges rather than fabricated

precision.

- Quality and risk are expected economic quantities, including failures that did not occur because controls worked.
- Marginal, avoidable, and fully allocated costs answer different questions and must not be mixed silently.
- The objective is not minimum local cost; it is maximum value under quality, risk, human, and strategic constraints.

6.20. Open questions

- Which telemetry best estimates active cognitive effort without invasive monitoring?
- How should fragmented minutes saved by AI tools be converted into realizable capacity?
- How can organizations value scarce expertise without reinforcing unhealthy single points of dependency?
- What is the best method for attributing cost and value across multi-repository Engineering Changes?
- How should option value and learning value be included in portfolio decisions?
- Which risk measures are most useful for rare, catastrophic engineering failures?

6.21. Implementation checklist

1. Define the cost components your organization will and will not estimate.
2. Separate active effort, waiting, blocked time, and context recovery.
3. Establish standardized loaded rates by role or level, with appropriate privacy controls.
4. Capture marginal AI and infrastructure usage at Engineering Change level.
5. Build ownership, handoff, and approval-boundary features for coordination analysis.
6. Define cost-of-delay classes and require assumptions to be visible.
7. Link post-merge rework, incidents, and support activity back to Engineering Changes with confidence scores.
8. Report uncertainty intervals and sensitivity analysis for derived dollar values.
9. Prohibit the use of process-cost estimates as direct individual performance rankings.
10. Evaluate workflow interventions using total quality-adjusted economics rather than a single component metric.

6.21.1. Research foundations and source notes

The economic framework in this chapter combines established cost-benefit, productivity, queueing, operations, and organizational research with several Engineering Intelligence-specific constructs.

Technology investment and realized productivity

The distinction between acquiring technology and realizing measurable productivity is central to the information-technology productivity-paradox literature. Solow's observation that computers were visible everywhere except measured productivity became a concise statement of the problem. Brynjolfsson subsequently examined measurement error, delayed learning, redistribution, and organizational transformation as explanations. Later work on artificial intelligence argued that general-purpose technologies often require substantial complementary innovation before gains appear in aggregate productivity. See [solow1987], [brynjolfsson1993], and [brynjolfsson2017].

This literature supports the chapter's claim that purchasing an engineering tool, increasing its usage, or observing local time savings does not by itself establish realized organizational value.

Cost-benefit analysis and decision alternatives

The chapter's treatment of incremental cost, credible alternatives, discounting, risk, sensitivity analysis, and uncertainty is grounded in standard cost-benefit-analysis practice. See [boardman2018].

The relevant comparison is the difference between the intervention and a realistic counterfactual—not the gross benefit of the intervention considered in isolation. This is why acquisition, implementation, operating, validation, migration, switching, risk, and opportunity costs must all be included when they differ across alternatives.

Opportunity cost and scarce attention

Opportunity cost is the value of the best alternative use of a constrained resource. In Engineering Intelligence, that resource is frequently specialist attention rather than generic labor hours.

The chapter's attention-scarcity multiplier is an operational construct, not a standard accounting rule. Its purpose is to represent the fact that a scarce reviewer, architect, security specialist, or incident expert can constrain organizational throughput beyond the person's loaded hourly cost.

Cognitive-load and interruption research supports the distinction between uninterrupted reasoning and fragmented or resumed work. See [sweller1988] and [mark2008]. Developer research on perceived productivity further supports treating invisible coordination, interruption, and progress as economically meaningful rather than assuming visible activity is the complete work product. See [meyer2014].

Queueing, utilization, and work in progress

Little's Law provides the formal relationship among average work in progress, throughput, and average flow time:

$$L = \lambda W$$

See [little1961].

The chapter uses this relationship descriptively rather than as a universal target. It applies only when the system boundary and averaging period are coherent and the system is sufficiently stable.

Kingman's heavy-traffic approximation and the broader operations literature explain why queue delay grows nonlinearly as utilization and variability rise. See [kingman1961] and [hopp2011]. This supports several important conclusions:

- small increases in constrained capacity can create large delay reductions near saturation;
- increasing upstream output can destabilize a downstream queue;
- reducing variability may be as important as increasing average service rate;
- average service time and end-to-end elapsed time answer different questions;
- saved minutes create little throughput value when the improved stage is not the bottleneck.

The constraints perspective described in [goldratt1984] provides the practitioner-oriented basis for evaluating improvements at the system bottleneck rather than optimizing every component independently.

Coordination and socio-technical cost

The chapter's treatment of team boundaries, handoffs, ownership, and interface negotiation is informed by socio-technical research.

Conway's observation links organizational communication structures to the systems organizations design. See

[conway1968]. Socio-technical congruence research examines alignment between technical dependencies and coordination relationships. See [cataldo2006]. Distributed-development research explains why distance, organizational boundaries, and communication dependencies create additional coordination difficulty. See [herbsleb2007].

The pairwise communication-path formula in the chapter is illustrative, not a predictive law for engineering effort. Real organizations structure communication through teams, interfaces, roles, and coordination mechanisms. The economic point is that coordination burden can grow nonlinearly when boundaries and dependencies are poorly designed.

Architecture as economic infrastructure

The treatment of platform work, modularity, migration, and future change cost is informed by information hiding and modular design. See [parnas1972].

A good boundary can reduce the cost of understanding, changing, testing, deploying, and coordinating future work. A poor boundary can impose repeated costs on every later Engineering Change. Team cognitive load and interaction modes provide another perspective on these costs. See [skelton2019].

The chapter's capability and option-value categories are intended to capture these durable future effects without forcing them into falsely precise short-term labor savings.

Quality economics and expected loss

The chapter models quality prospectively as probability multiplied by consequence and retrospectively through observed rework, incidents, support, remediation, and follow-up change.

This is consistent with standard decision analysis and cost-benefit methods described in [boardman2018]. It also aligns with reliability engineering's emphasis on service objectives, risk, toil, and operational consequence. See [beyer2016] and [beyer2018].

Expected value is not sufficient for every decision. Catastrophic tail risk, regulatory constraints, security policy, and ethical boundaries may require explicit limits even when expected monetary loss appears acceptable.

Cost of delay

Cost of delay represents value lost while an outcome, capability, risk reduction, or learning opportunity remains unavailable.

The chapter intentionally treats value curves as uncertain and context-specific. Fixed-date, expedite, standard, and exploratory work have different delay profiles. Cost of delay should therefore be reported as assumptions, ranges, or scenarios rather than used to manufacture universal urgency.

Its interaction with queues and constraints is grounded in [little1961], [kingman1961], [hopp2011], and [goldratt1984].

Marginal, avoidable, and allocated cost

The chapter distinguishes several cost perspectives:

- **marginal cost** — the additional cost of one more unit of activity;
- **avoidable cost** — future cost eliminated by a decision;
- **fully allocated cost** — direct and shared cost assigned across a population;

- **replacement cost** — cost of recreating or substituting a capability;
- **opportunity cost** — value of the best displaced alternative.

These perspectives are not interchangeable. The cost of adding one test run may be marginal compute, while a vendor-selection decision must also consider platform staffing, fixed contracts, support, migration, and exit.

AI economics

The chapter's AI cost model extends ordinary tool economics by including:

- inference and infrastructure;
- indexing and retrieval;
- validation effort;
- false-positive handling;
- rework;
- administration and governance;
- downstream quality effects;
- genuinely avoided human effort.

Controlled and observational research on AI-assisted programming demonstrates that effects depend on task, tool, user behavior, and evaluation boundary. See [peng2023], [vaithilingam2022], and [barke2023].

These studies do not justify a universal conversion from accepted suggestions or generated code into productivity value. The chapter therefore evaluates AI through net attention, quality-adjusted outcomes, and realized capacity rather than gross generation or acceptance.

Capacity realization and the productivity paradox

The chapter's distinction between released capacity and realized savings is consistent with the productivity-paradox literature in [brynjolfsson1993] and [brynjolfsson2017].

Technology may release time locally but fail to improve organizational output because:

- the time is fragmented;
- another stage remains constrained;
- complementary processes have not changed;
- incentives direct effort toward low-value activity;
- quality or validation cost offsets the gain;
- the organization does not reduce spending or redirect capacity;
- measured outcomes mature only after a delay.

The **realization factor** and **continuity factor** introduced in this book are operational modeling constructs for making those assumptions explicit.

Uncertainty and sensitivity

A decision-grade economic model should expose distributions or scenarios for uncertain inputs rather than present a precise point estimate.

The chapter's use of scenario analysis, sensitivity analysis, expected value, and Monte Carlo simulation follows standard decision-analysis and cost-benefit practice. See [boardman2018].

The purpose is not to make uncertainty disappear. It is to identify:

- the probability the decision creates positive value;
- plausible downside;
- the assumptions that dominate the result;
- the additional evidence most valuable to collect;
- the conditions under which the decision should be reversed.

Measurement and gaming risk

Economic metrics can change behavior when they become targets. Reporting cost per Engineering Change, review effort, or estimated capacity at individual level can encourage work avoidance, change slicing, under-reporting, or selection of easier tasks.

The chapter's warning against individual ranking and local cost optimization is supported by the indicator-corruption concerns in [campbell1979] and [goodhart1984], and by the use-specific validity framework in [messick1995].

Original Engineering Intelligence contributions

The following concepts are original synthesis or domain-specific operational extensions in this book rather than externally established standards:

- total Engineering Change cost as an integrated evidence model;
- attention-scarcity multipliers;
- context-recovery cost from engineering event evidence;
- net AI attention economics;
- capacity-realization and continuity factors;
- quality-adjusted engineering economics;
- value-realization ledgers connected to Engineering Change evidence;
- explicit labels separating cash impact, avoided cost, released capacity, modeled outcome value, and option value;
- benefit-dependency maps to prevent double counting;
- integration of causal effect estimates with engineering business cases.

These constructs should be calibrated, challenged, and validated in each organizational context rather than treated as universal constants.

Chapter 7. The Engineering Change



A pull request is a platform artifact, not a reliable unit of engineering work. One coherent intent *may span repositories, migrations, approvals, feature flags, deployments, and operational tasks; one pull request may contain several unrelated intents. This chapter formalizes the Engineering Change as the primary analytical unit of Engineering Intelligence. It defines its entities and boundaries, models it as a temporal property graph, describes deterministic and probabilistic reconstruction, introduces confidence and provenance requirements, and explains how to instrument the lifecycle at enterprise scale.*

7.1. Why the pull request is insufficient

Pull requests are attractive analytical units because they are visible, structured, and supported by mature APIs. They contain timestamps, authors, commits, reviewers, comments, checks, and merge outcomes. For many local questions, the pull request is exactly the right unit.

It becomes misleading when used as a universal proxy for engineering work.

A customer-visible capability may require a web change, a mobile change, an API update, a shared-library release, a database migration, infrastructure configuration, security approvals, a feature-flag rollout, documentation, and post-deployment verification. Each artifact may appear in a different repository or system. Measuring them independently hides the actual coordination, critical path, and outcome.

The reverse problem also occurs. A pull request may combine a refactor, dependency update, and product feature that should be analyzed separately. Repository conventions, not engineering intent, determine the boundary.

A reliable measurement system must therefore distinguish between observable artifacts and the logical change those artifacts collectively realize.

7.2. Formal definition



An **Engineering Change** is the temporally bounded set of human, AI, code, configuration, data, infrastructure, verification, coordination, deployment, and operational activities undertaken to transform one coherent engineering intent into one or more accepted outcomes.

Let an Engineering Change be represented as:

$$E = (I, A, R, O, \tau, \Pi)$$

where:

- I is the engineering intent;
- A is the set of participating artifacts and activities;
- R is the set of relationships among them;
- O is the set of intended and realized outcomes;
- τ is the temporal extent;
- Π is provenance, confidence, and reconstruction metadata.

This representation separates what the organization intended from what its tools observed.

7.2.1. Engineering intent

Engineering intent is the reason the work exists. Examples include enabling a capability, remediating a vulnerability, migrating data, reducing cost, retiring a service, improving build performance, investigating a failure, satisfying regulation, or testing a hypothesis.

Intent may be explicit in a change record or inferred from tickets, descriptions, and semantic evidence. It may also evolve. A defect fix can become a migration; a prototype can become a product feature. The model should preserve revisions rather than overwrite history.

7.2.2. Artifacts and activities

Artifacts are durable objects such as commits, pull requests, tickets, design documents, build records, binaries, deployments, feature flags, schemas, runbooks, and incidents.

Activities are events or intervals such as authoring, reviewing, testing, approving, deploying, observing, rolling back, discussing, and validating.

The distinction matters because a pull request is an artifact while a review session is an activity. Both contribute to cost and flow.

7.2.3. Relationships

Relationships connect the graph. Common edge types include `IMPLEMENTS`, `DEPENDS_ON`, `BLOCKS`, `REVIEWS`, `GENERATES`, `DEPLOYS`, `VERIFIES`, `ROLLS_BACK`, `SUPERSEDES`, `CAUSED_BY`, `REMEDIATES`, `PART_OF`, `OWNED_BY`, and `AFFECTS`.

Edges should carry time, source, confidence, and reconstruction method. A link declared in a release manifest is not equivalent to one inferred from title similarity.

7.2.4. Outcomes

An outcome is the terminal or intermediate state the Engineering Change is intended to create. A merged pull request is not necessarily an outcome. A feature may be merged but disabled, deployed but not exposed, exposed but not adopted, or rolled back.

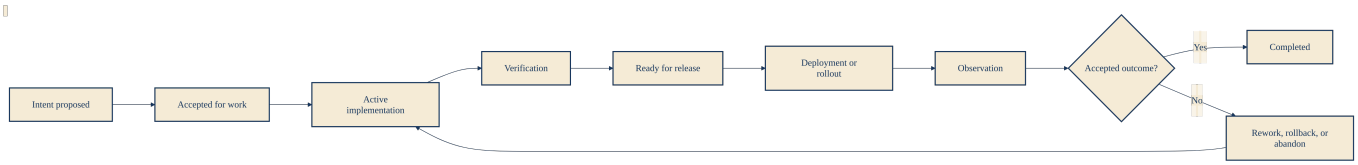
Useful outcome states include accepted and operating, partially accepted, deployed but disabled, abandoned, superseded, rolled back, failed validation, completed without deployment, and informational outcome achieved.

7.3. The Engineering Change graph

The natural representation is a temporal property graph:

$$G_E = (V_E, E_E, X_V, X_E, T)$$

where V_E is the set of nodes, E_E the set of edges, X_V node attributes, X_E edge attributes, and T temporal information.



A graph representation enables questions that flat pull-request tables cannot answer well:

- Which artifact lies on the critical path?
- How many repository and ownership boundaries were crossed?
- Which approvals delayed the accepted outcome?
- Which deployments were associated with the same intent?
- Did an incident originate from a change spanning several repositories?
- How much human and AI effort accumulated across the complete change?
- Which dependencies repeatedly create coordination cost?

Each of these is a traversal over the Engineering Change graph rather than a query against a flat activity log. The graph therefore does more than reshape the data: it makes coordination, dependency, and provenance first-class citizens of the analysis. Subsequent chapters use this representation for cost accounting, blast-radius estimation, and experimental unit-of-analysis decisions.

7.4. Boundaries and identity

Every metric depends on a boundary. The Engineering Change boundary determines which activities, costs, outcomes, and failures are attributed together.

A robust identity scheme should provide a stable change identifier, parent-child relationships, aliases from external systems, versioned intent, explicit lifecycle state, confidence when membership is inferred, and merge-split history.

An opaque organization-wide ID such as `EC-01J8…` is preferable to a ticket number or repository-local key. External identifiers become aliases.

7.4.1. Parent and child changes

Large initiatives should not become one unbounded Engineering Change. They should form a hierarchy:

$$E_{initiative} \supset \{E_1, E_2, \ldots, E_n\}$$

A parent can aggregate value and dependency context, while child changes retain measurable lifecycle boundaries. For example, a payment migration may have children for dual-write support, historical backfill, read cutover, and legacy retirement.

7.4.2. Merge and split operations

Reconstruction is imperfect. Two changes initially believed separate may later prove to be one coordinated intent. One reconstructed change may contain unrelated work and need splitting.

The data model should support `MERGED_INTO`, `SPLIT_FROM`, effective time, reason, actor or algorithm, previous membership, and metric recomputation. Historical reports must remain reproducible.

7.5. Lifecycle states

A generic lifecycle is proposed, accepted for work, active, under verification, ready for release, releasing, observing, and accepted, abandoned, or rolled back.

Real workflows are not strictly linear. Work may return from review, pause, fork, or deploy incrementally. State should therefore be derived from events rather than assumed from a waterfall sequence.

A state transition can be represented as:

$$S_{t+1} = f(S_t, event_t, context_t)$$

The transition function should be versioned. Changing the definition of "active" or "ready for release" can materially alter flow metrics.

7.5.1. Event time and processing time

Enterprise event pipelines must distinguish event time, ingestion time, and processing time. Late and corrected events are normal. A deployment record may arrive after a source-control webhook. An incident may be linked to a change days later. Metrics must support retractions and recomputation.

7.6. Deterministic reconstruction

The strongest reconstruction uses explicit identifiers propagated through the toolchain.

A change ID can be attached to issue metadata, branch names, commit trailers, pull-request fields, CI environment variables, artifact metadata, deployment manifests, feature flags, telemetry attributes, and incident records.

```
Engineering-Change: EC-01J8Z6D4K7J6Q2Q9H1K3B4M5N6
```

A deployment could emit the same identifier as an OpenTelemetry resource attribute:

```
engineering.change.id = "EC-01J8Z6D4K7J6Q2Q9H1K3B4M5N6"
```

Explicit propagation produces high-confidence linkage and allows production telemetry to connect back to intent.

7.6.1. Adoption strategy

A practical adoption sequence is to generate IDs for new work, add optional pull-request and deployment fields, automate propagation in CI/CD templates, backfill high-value repositories, use probabilistic reconstruction for missing links, measure coverage and confidence, and make IDs mandatory only where the benefit justifies the friction.

7.7. Probabilistic reconstruction

In existing estates, most changes must initially be inferred.

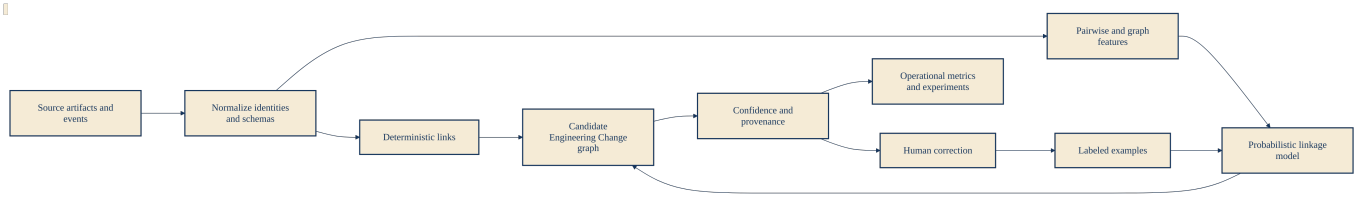
For artifacts a_i and a_j, define a linkage probability:

$$P(z_{ij} = 1 \mid X_{ij}) = \sigma(\beta_0 + \beta^T X_{ij})$$

where $z_{ij} = 1$ indicates common Engineering Change membership, X_{ij} is a feature vector, and σ is a logistic function.

Features may include exact ticket matches, explicit references, shared authors and reviewers, temporal proximity, repository dependency, coordinated deployment, shared feature flags, semantic similarity, common design documents, service dependencies, and incident linkage.

A graph clustering or correlation-clustering method can then propose Engineering Change groups.



7.7.1. Negative evidence

Reconstruction must include evidence that artifacts do not belong together: conflicting issue IDs, unrelated release trains, large temporal separation, mutually exclusive product areas, explicit independence labels, distinct accepted outcomes, or common shared-library relationships that are not change-specific.

Ignoring negative evidence causes giant meaningless clusters around active engineers or frequently changed repositories.

7.7.2. Confidence

Every inferred membership should carry confidence:

$$conf(a \in E) \in [0, 1]$$

Class	Typical confidence	Interpretation
Declared	1.00	Explicit identifier or authoritative manifest.
Strongly inferred	0.90–0.99	Multiple independent high-quality signals.
Inferred	0.70–0.89	Likely membership with meaningful uncertainty.
Tentative	0.50–0.69	Useful for exploration, unsuitable for high-stakes attribution.
Rejected	below threshold	Not included in the graph.

Thresholds depend on the decision. Executive cost attribution may require stronger confidence than exploratory dependency discovery.

7.8. Provenance and explainability

A reconstruction system must answer, "Why are these artifacts linked?"

For each edge or membership decision, retain source systems, contributing signals, model version, feature values, explanation, confidence, creation time, correction time, and human override history.

```
{
  "artifact": "repo-a#1842",
  "engineeringChange": "EC-01J8...",
  "confidence": 0.96,
  "evidence": [
    {"type": "ticket_match", "value": "PAY-431", "weight": 0.52},
    {"type": "explicit_pr_reference", "value": "repo-b#991", "weight": 0.27},
    {"type": "coordinated_deployment", "windowMinutes": 18, "weight": 0.11}
  ],
  "modelVersion": "reconstruct-v3.4"
}
```

Explainability is necessary for trust, correction, and debugging. It also reveals systematic data-quality problems.

7.9. Multi-repository changes

Multi-repository work should be analyzed as a dependency graph, not a bag of pull requests.

Let P_E be pull requests in Engineering Change E, and let edges represent ordering or compatibility dependencies. The critical path is the longest-duration path through the directed acyclic portion of the graph.

$$T(E) = \max_{p \in Paths(E)} \sum_{a \in p} d(a)$$

Useful graph measures include repository count, dependency depth, fan-in, fan-out, ownership boundaries, critical-path length, parallelism ratio, synchronization points, cycles, and release-order constraints.

Cycles deserve special attention. Two changes that each require the other can create compatibility layers, coordinated merges, or manual sequencing. The coordination cost may far exceed implementation complexity.

7.10. Repository topology as context

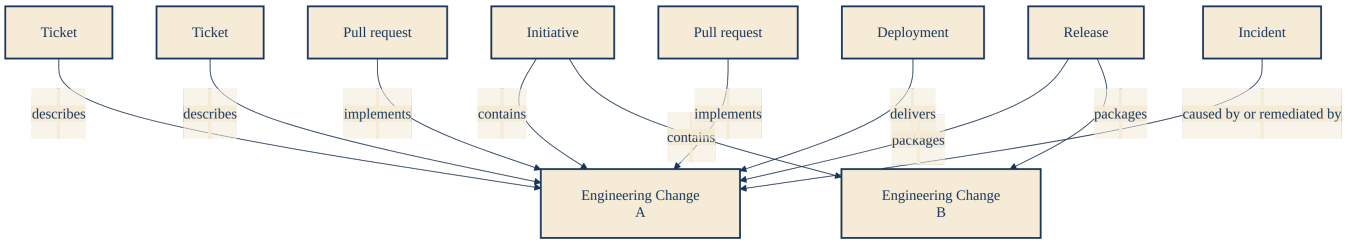
In a monorepo, one pull request may touch many services and teams. In a polyrepo environment, the same intent may produce a chain of pull requests. Raw pull-request counts and sizes are therefore not comparable without topology context.

The Engineering Change abstraction distinguishes intrinsic complexity, topological complexity, and organizational complexity. This supports experiments that estimate the economic effect of repository layout and ownership structure.

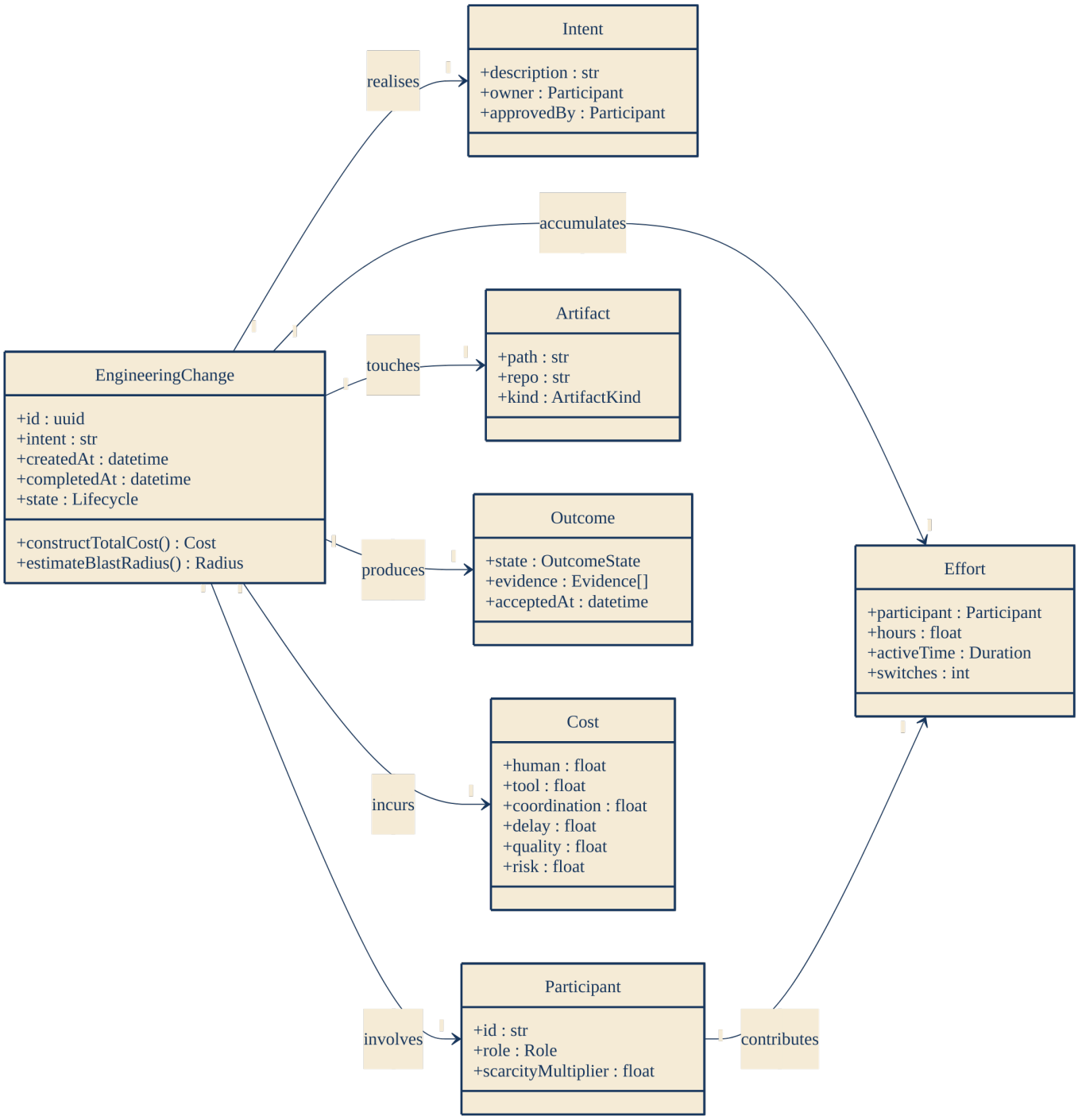
7.11. Relationship to initiatives, tickets, and releases

The Engineering Change does not replace all planning entities. An initiative is usually broader and longer-lived. A ticket may describe one task or defect. A release is a packaging concept that may include many changes. A pull request is a source-control collaboration object.

The relationships are many-to-many:



A strong platform preserves each entity and connects them rather than forcing one system to become the universal source of truth.



The domain model above shows the Engineering Change with the attributes and derived quantities it must expose to support cost accounting, blast-radius estimation, and cohort selection later in the book. The cardinalities matter: because a single Engineering Change may accumulate Effort from many Participants and produce exactly one Outcome, aggregation queries that treat any of these relationships as one-to-one will systematically under- or over-count.

7.12. Membership over time

Membership can change as work evolves. An artifact may initially belong to a prototype and later be incorporated into a production change. A shared-library release may serve several Engineering Changes.

The graph therefore needs temporal membership:

$$membership(a, E, [t_s, t_e], role, confidence)$$

An artifact can participate in multiple changes with different roles. Cost allocation should reflect role and causality rather than duplicating the full cost into every parent.

7.13. Cost and value allocation

When one artifact supports several Engineering Changes, possible allocation methods include equal allocation, usage-weighted allocation, causal attribution, Shapley-value approximations, primary-change ownership with secondary references, or reporting shared-platform cost separately.

No method is universally correct. The allocation method must match the decision.

7.14. Instrumentation architecture

Engineering Change instrumentation should be event-driven. Source systems emit normalized events into an immutable log. Identity-resolution services map users, repositories, services, teams, and tools. A reconstruction service creates and updates graphs. Stream processors maintain near-real-time lifecycle state, while batch jobs recompute topology, semantics, and historical corrections.

Core event domains include planning, source control, AI, CI, deployment, production, and organization.

A normalized event envelope should include:

```
{
  "eventId": "01J...",
  "eventType": "pull_request.review.submitted",
  "eventTime": "2026-07-15T21:42:13Z",
  "ingestionTime": "2026-07-15T21:42:15Z",
  "source": "github-enterprise",
  "sourceEntityId": "repo-a#1842/review/7781",
  "actor": {"type": "human", "id": "person-392"},
  "engineeringChangeId": "EC-01J8...",
  "changeLinkConfidence": 1.0,
  "schemaVersion": "1.2",
  "payload": {}
}
```

7.15. Near-real-time and batch responsibilities

Near-real-time computation is appropriate for current state, review and CI queues, aging changes, experiment exposure, AI interactions, deployment progression, and operational alerts.

Batch computation is appropriate for repository topology, semantic clustering, historical reconstruction, ownership networks, code-complexity analysis, model training, incident attribution, and cost recomputation.

The two paths must converge on versioned state. A batch correction should update the graph without erasing the evidence that produced the previous state.

7.16. Data-quality metrics

Recommended data-quality measures include explicit ID coverage, inferred membership coverage, percentage above confidence thresholds, orphan artifact rate, contradictory-link rate, late-event rate, source-event loss, identity ambiguity, reconstruction correction rate, event-to-state latency, and metric recomputation lag.

Coverage and confidence should be displayed alongside derived measures.

7.17. Validation of reconstruction

Reconstruction quality can be evaluated as an information-retrieval problem.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Pairwise precision and recall measure whether artifact pairs are correctly grouped. Cluster metrics such as B-cubed precision and recall, adjusted Rand index, and normalized mutual information can evaluate whole-group quality.

Ranking metrics such as mean reciprocal rank and NDCG are useful when the system proposes candidate related artifacts.

Evaluation should be stratified by monorepo versus polyrepo, single-team versus cross-team, work type, explicit versus missing ticket IDs, repository activity level, and long-running versus short changes.

7.18. Human correction and active learning

Users should be able to attach or detach an artifact, merge or split changes, correct intent, confirm an outcome, or mark a relationship as invalid.

Corrections become labeled data for active learning. The system should prioritize uncertain or high-value cases for review rather than asking users to validate every graph.

The correction interface must show why a link exists and what downstream reports will change. Silent mutation destroys trust.

7.19. Privacy and governance

Engineering Change reconstruction combines data across many systems and can expose sensitive organizational behavior. Governance requirements include purpose limitation, role-based access, retention rules, aggregation thresholds, audit logs, model transparency, and explicit prohibition of unsupported individual ranking.

Human identity is often unnecessary for system analysis. Pseudonymous identifiers and role-level aggregation should be used whenever the decision does not require named attribution.

7.20. Common failure modes

7.20.1. Treating ticket links as ground truth

Tickets are useful but often stale, missing, or reused. They are one source of evidence, not an infallible boundary.

7.20.2. Clustering by author and time alone

Active engineers often work on several unrelated changes. This creates false joins.

7.20.3. Ignoring deployment and production evidence

A change is not complete at merge. Without release and outcome linkage, flow and quality metrics are truncated.

7.20.4. Hiding uncertainty

Inferred links should not be displayed as facts. Confidence and provenance must travel with the data.

7.20.5. Building one giant initiative node

Overly broad changes make cost, latency, and causality meaningless. Use hierarchical decomposition.

7.20.6. Recomputing history without versioning

Changing reconstruction rules can alter past metrics. Preserve model versions and support reproducible analysis.

7.21. Worked example: reconstructing an Engineering Change under confidence



Runnable version: `companion/notebooks/08-engineering-change-reconstruction.py`.

A platform team wants to attribute cost and outcome to an Engineering Change titled "add refund reason codes to checkout." The reconstruction service assembles evidence from six systems: the intent ticket, three pull requests across two repositories, a feature-flag configuration change, and a deployment record.

Deterministic linkage — where the pull-request description or commit trailer carries `refs REFUND-8842` — connects two of the three pull requests to the intent with confidence 1.0. The third pull request modifies the payment-integration service and shares an author but no explicit reference; a probabilistic model links it with confidence 0.62 based on temporal co-occurrence, semantic similarity of description, and shared file paths with the other two changes.

The feature-flag change is emitted by a separate system without repository references. It links to the intent via a normalized flag name (`checkout.refund_reason_codes_enabled`) that matches text in the intent description; confidence 0.88.

Total human effort attributed:

- deterministic edges: 14.2 authored hours + 6.1 review hours = 20.3 hours ± low uncertainty
- probabilistic edges: additional 4.4 authored hours and 1.7 review hours, weighted by their link confidences: $4.4 \times 0.62 + 1.7 \times 0.62 = 3.78$ expected additional hours

Total expected effort: 24.1 hours. A conservative interpretation reports 20.3 hours (deterministic only) as the floor and 26.4 hours (all evidence, unweighted) as the ceiling. The versioned reconstruction stamps every metric derived from this Engineering Change with reconstruction model v3.4, so if the linkage model is later revised the effort estimate can be

recomputed rather than retconned.

7.22. Key takeaways

- Pull requests are observable artifacts, not universal units of engineering work.
- The Engineering Change connects coherent intent to accepted outcome across tools, repositories, teams, and deployment stages.
- A temporal property graph is the natural representation for multi-repository and multi-team work.
- Explicit identifiers provide the strongest linkage, but probabilistic reconstruction is essential for existing estates.
- Every inferred membership requires confidence, provenance, explanation, and correction history.
- Repository topology, ownership boundaries, and deployment sequencing are context features, not substitutes for intent.
- The reconstruction system itself must be measured for coverage, precision, recall, latency, and correction rate.

7.23. Open questions

- What graph-clustering methods best reconstruct changes at enterprise scale while respecting temporal constraints?
- How should shared artifacts be allocated across several Engineering Changes?
- Which confidence thresholds are appropriate for operational dashboards, experiments, and financial attribution?
- How can organizations make change-ID propagation useful enough that teams adopt it voluntarily?
- What is the right terminal outcome for platform, research, and infrastructure changes?

7.24. Implementation checklist

1. Define the Engineering Change entity, identifier, aliases, lifecycle, and outcome vocabulary.
2. Add optional explicit change IDs to planning, SCM, CI/CD, and telemetry templates.
3. Build a normalized event envelope with event time, source identity, schema version, and provenance.
4. Create deterministic linking rules before introducing machine learning.
5. Add probabilistic reconstruction with positive and negative evidence.
6. Store confidence and explanation for every inferred relationship.
7. Support parent-child, merge, split, and temporal membership operations.
8. Build a labeled validation set and report pairwise and cluster-quality metrics.
9. Expose graph corrections through a low-friction user interface.
10. Display reconstruction coverage and confidence alongside every high-level metric.

7.24.1. Research foundations and source notes

The Engineering Change model combines established ideas from modularity, socio-technical systems, provenance, event-sourced data architecture, and temporal modeling with a domain-specific analytical unit proposed by this book.

Observable artifacts versus semantic work

The distinction between a tool artifact and the engineering intent it helps realize is rooted in a broader systems principle: implementation boundaries do not necessarily match conceptual or organizational boundaries.

Parnas's information-hiding work established that useful modular boundaries should be organized around design decisions and likely sources of change rather than merely around execution steps or file structure. See [parnas1972].

Conway's observation that system structures reflect organizational communication structures, together with later work on socio-technical congruence, explains why repository, team, and coordination boundaries affect how work is represented and executed. See [conway1968], [cataldo2006], and [herbsleb2007].

These foundations support the chapter's argument that pull-request boundaries are useful source-control artifacts but unreliable universal proxies for engineering consequence.

The Engineering Change as original synthesis

The **Engineering Change** is a domain-specific construct introduced in this book. It is not a renamed pull request, ticket, release, work item, or deployment.

Its defining contribution is to connect:

- intent;
- human and machine actors;
- artifacts and activities;
- validation and decisions;
- builds, deployments, and releases;
- incidents and operational evidence;
- economic, user, quality, and learning outcomes;
- confidence and provenance across the complete lifecycle.

The model is intended to provide a stable unit of consequence even when tool topology, repository strategy, or organizational structure changes.

Provenance entities, activities, and agents

The W3C PROV data model defines a general framework involving entities, activities, agents, derivation, generation, attribution, and association. See [moreau2013].

The Engineering Change graph is compatible with this foundation:

- artifacts and records are provenance entities;
- authoring, review, validation, build, deployment, and operation are activities;
- people, services, automation, assistants, and agents are actors;
- edges represent generation, use, attribution, derivation, delegation, and domain-specific relationships.

The chapter adds engineering-specific lifecycle semantics such as `IMPLEMENTS`, `VERIFIES`, `DEPLOYS`, `ROLLS_BACK`, `REMEDIATES`, and `PART_OF`.

Event logs and reconstructable state

The architecture described in this chapter follows the durable-log principle developed in modern data-intensive systems. Source evidence is preserved as immutable or append-oriented events, while current state is derived through replayable transformations. See [kleppmann2017].

Streaming-systems research provides the formal distinction among event time, processing time, late data, watermarks, and corrections. See [akidau2018].

These principles support several requirements in the chapter:

- lifecycle state should be derived from events rather than overwritten without history;
- late incident and deployment evidence must be incorporated;
- reconstruction rules must be versioned;
- corrected relationships must trigger recomputation;
- historical metrics must remain reproducible;
- ingestion time must not be mistaken for occurrence time.

Temporal identity and ownership

Ownership, membership, and artifact relationships change over time. A repository's current owner may not have owned it when an earlier decision or incident occurred.

The temporal representation in this chapter is therefore an architectural requirement rather than an optional analytical detail. Events, relationships, ownership, and membership should carry effective intervals or as-of semantics.

This treatment is consistent with slowly changing dimensions and temporal data patterns common in data-intensive system design, as described broadly in [kleppmann2017]. The Engineering Change model extends these patterns to organizational and lifecycle evidence.

Explicit identifiers and propagated context

The strongest linkage is created when a stable change identifier is propagated through planning, source control, CI, artifacts, deployment, release, telemetry, and incident systems.

This design is compatible with distributed tracing and provenance practices: a durable identifier creates a causal and operational thread across otherwise separate systems. The chapter's specific `engineering.change.id` convention is a proposed domain standard rather than an externally established specification.

The adoption strategy deliberately begins with optional propagation because mandatory metadata can create friction before sufficient organizational value has been demonstrated.

Deterministic and probabilistic reconstruction

When explicit identifiers are absent, reconstruction becomes an entity-resolution and graph-linkage problem.

The chapter distinguishes:

- authoritative declared relationships;
- deterministic lineage from commits, artifacts, builds, and manifests;

- rule-based inference;
- probabilistic or model-based inference;
- human correction.

This hierarchy reflects a general evidence principle: not all links have equal authority. Provenance, feature values, model version, and confidence must therefore accompany inferred membership.

The probability expression in the chapter is illustrative. It does not prescribe logistic regression as the universal reconstruction method. Practical systems may use rules, supervised classifiers, graph clustering, correlation clustering, retrieval models, or hybrid methods.

Negative evidence

Entity-resolution systems often fail when they use only evidence of similarity. Shared contributors, active repositories, common libraries, and nearby timestamps can create false giant clusters.

The chapter therefore requires negative evidence such as incompatible intent, distinct release trains, explicit independence, temporal separation, or non-change-specific shared dependencies.

This is an Engineering Intelligence-specific operational principle, but it follows standard classification and record-linkage reasoning: valid inference requires evidence both for and against membership.

Confidence and calibration

A numeric confidence score has value only when its meaning is defined and calibrated for the target population.

The chapter's confidence bands—declared, strongly inferred, inferred, tentative, and rejected—are controlled operational categories. They should be calibrated against adjudicated Engineering Change samples and reviewed by decision type.

Exploratory dependency analysis can tolerate lower-confidence relationships than financial attribution, causal studies, or governance investigations.

Measurement-validity principles in [messick1995] support this use-specific treatment: the acceptability of a relationship depends on the interpretation and consequence of its use.

Graph structure and critical paths

The chapter represents multi-repository work as a dependency graph rather than an unordered collection of pull requests.

Graph measures such as dependency depth, fan-in, fan-out, ownership boundaries, cycles, synchronization points, and critical-path length help characterize topological and coordination difficulty.

The broader rationale is consistent with modularity and socio-technical research in [parnas1972], [conway1968], and [cataldo2006]. The specific Engineering Change graph schema and measures are domain-specific extensions.

Repository topology and organizational context

A monorepo and a polyrepo environment can encode the same intent into radically different numbers and shapes of pull requests. Raw artifact counts therefore cannot be compared without topology context.

Team Topologies emphasizes the relationship among team boundaries, cognitive load, interaction modes, and fast flow.

See [skelton2019]. This supports the chapter's separation of:

- intrinsic implementation complexity;
- repository and dependency topology;
- organizational ownership and coordination complexity.

The Engineering Change creates a common semantic layer across these different implementation structures.

Shared artifacts and cost allocation

A shared library, platform capability, or deployment artifact can support several Engineering Changes. Assigning its full cost to every change would double count cost; assigning it to none would hide shared investment.

The chapter therefore presents several allocation choices, including equal allocation, usage weighting, causal attribution, primary ownership, and separate shared-platform reporting.

This is consistent with the decision-specific cost perspective developed in Chapter 2 and the cost-benefit methods summarized in [boardman2018]. No allocation rule is universally correct; the method must match the decision.

Reconstruction quality as a first-class metric

Every metric derived from Engineering Changes inherits the quality of change reconstruction.

The chapter therefore treats the following as core platform-quality measures:

- explicit identifier coverage;
- artifact and deployment linkage coverage;
- precision and recall against adjudicated examples;
- confidence calibration;
- unresolved identity and ownership rate;
- correction volume;
- reconstruction drift;
- recomputation latency.

This follows the broader measurement principle that source and transformation quality are part of metric validity, not separate implementation details. See [messick1995] and [hand2004].

Privacy and acceptable use

The Engineering Change model can expose detailed individual and team activity. That capability creates governance obligations.

Privacy-risk and security-management principles are informed by [nist2020-privacy] and [nist2024-csf]. The book's governance standard further restricts individual employment use, requires purpose-bound access, and provides correction and appeal mechanisms.

The analytical graph should use pseudonymous actor identifiers by default, isolate re-identification, and avoid making inferred relationships appear more authoritative than their evidence supports.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific standards proposed by this book:

- the Engineering Change as the primary unit of consequence;
- the formal tuple connecting intent, activities, relationships, outcomes, time, and provenance;
- a temporal Engineering Change property graph;
- explicit merge, split, and supersession history;
- confidence-aware artifact membership;
- negative-evidence requirements for reconstruction;
- Engineering Change identifier propagation across the delivery and operational toolchain;
- lifecycle-state derivation from versioned events;
- multi-repository critical-path and coordination analysis;
- decision-dependent confidence thresholds;
- correction-triggered metric and decision impact analysis;
- integration of human and AI participation within one provenance model.

These contributions should be validated empirically through labeled reconstruction datasets, practitioner review, linkage-quality studies, and decision-outcome analysis.

Chapter 8. Human Attention and Cognitive Load



Human attention is the most constrained resource in modern software engineering. *Compute can be provisioned, repositories can be replicated, and AI inference can be scaled, but expert judgment, working memory, and sustained focus remain scarce. This chapter develops a measurement framework for cognitive burden without relying on intrusive surveillance or large-scale surveys. It distinguishes intrinsic task complexity from process-induced friction, models context switching and attention residue, introduces cognitive offload and cognitive reload, and describes how AI-assisted review can be evaluated as a redistribution of cognitive work.*

8.1. Why attention is the scarce resource

Engineering organizations often describe their constraints in terms of headcount, budget, or compute. These are real constraints, but they are incomplete. The binding constraint is frequently the availability of focused, appropriately skilled human attention at the moment it is needed.

A security specialist may be employed full time yet remain unavailable to a particular change because dozens of other changes compete for the same expertise. An author may have calendar capacity but lack the uninterrupted period required to reconstruct a distributed-system failure. A reviewer may technically complete a review while fatigued, fragmented, or unfamiliar with the domain.

Engineering Intelligence therefore treats attention as an economic and operational resource rather than as an abstract psychological concept.



The objective is not to maximize visible activity. It is to maximize valuable engineering outcomes per unit of scarce human attention while preserving quality, learning, and sustainability.

8.2. Cognitive burden is multidimensional

The phrase *cognitive load* is often used as if it were one measurable quantity. In engineering work, several distinct burdens should be separated.

Dimension	Description
Intrinsic complexity	Reasoning required by the problem itself: algorithms, concurrency, security semantics, financial rules, data consistency, or advanced interaction design.
Representational complexity	Difficulty created by how the system is expressed: poor naming, fragmented documentation, indirect abstractions, generated layers, or inconsistent conventions.
Navigational complexity	Effort required to locate relevant code, owners, tests, dependencies, and operational context.
Coordination complexity	Mental effort required to track people, decisions, approvals, dependencies, and release sequencing.
Temporal complexity	Burden caused by delays, interruptions, changing assumptions, and partially completed work.
Verification complexity	Effort required to determine whether a proposed change is correct, safe, complete, and appropriate.
Tool interaction burden	Effort required to operate development environments, CI systems, review tools, and AI assistants.

The distinction matters because interventions act on different dimensions. Better documentation may reduce navigational complexity while leaving intrinsic complexity unchanged. An AI reviewer may reduce verification search but increase tool interaction burden. A monorepo may reduce dependency discovery while increasing local code volume.

8.3. A cognitive work model

Let the human cognitive work associated with Engineering Change e be:

$$W_{human}(e) = W_{create} + W_{understand} + W_{verify} + W_{coordinate} + W_{recover} + W_{decide}$$

where:

- W_{create} is reasoning used to produce designs, code, tests, and documentation;
- $W_{understand}$ is effort spent building a mental model of the relevant system;
- W_{verify} is effort spent checking correctness, safety, and completeness;
- $W_{coordinate}$ is effort spent aligning with other people and systems;

- $W_{recover}$ is effort spent restoring context after delay or interruption;
- W_{decide} is effort spent resolving ambiguity and choosing among alternatives.

These quantities are latent. They cannot be observed directly at enterprise scale. The measurement problem is therefore one of inference and validation.

8.4. Measuring latent burden through proxies

A proxy is useful only if it has predictive or convergent validity. The fact that a signal is easy to collect does not mean it measures cognitive burden.

Potential non-invasive proxies include:

- number of materially distinct files and concepts revisited;
- time between opening a review and taking the first substantive action;
- repeated navigation through the same code regions;
- number and duration of review-response cycles;
- unresolved discussion branches;
- number of simultaneous active Engineering Changes;
- repository and domain transitions during an active change;
- repeated CI runs without meaningful code change;
- comment edits, reversals, and reopened decisions;
- delay between a comment and the author's response after controlling for availability;
- amount of new information introduced after the author believed the change complete;
- proportion of AI findings requiring independent investigation.

No single proxy should become a cognitive-load score. A robust model combines signals, controls for change complexity, and validates against outcomes such as resumption time, error probability, review latency, and downstream rework.

8.5. Context switching and attention residue

When a person leaves an unfinished task, part of their attention remains attached to it. Returning later requires context reconstruction and may temporarily degrade performance on both the old and new task.

For engineer p and Engineering Change e, define a context-switch event whenever active attention moves to another materially distinct task for longer than a configured threshold.

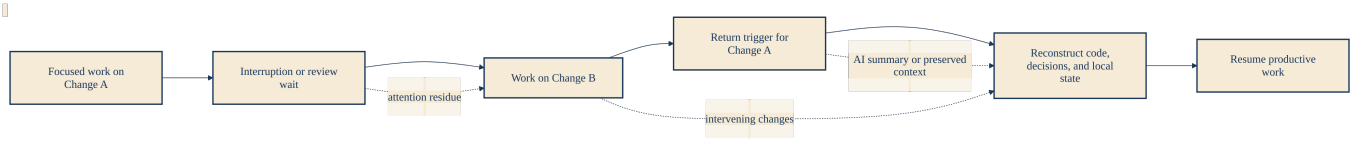
A simple recovery model is:

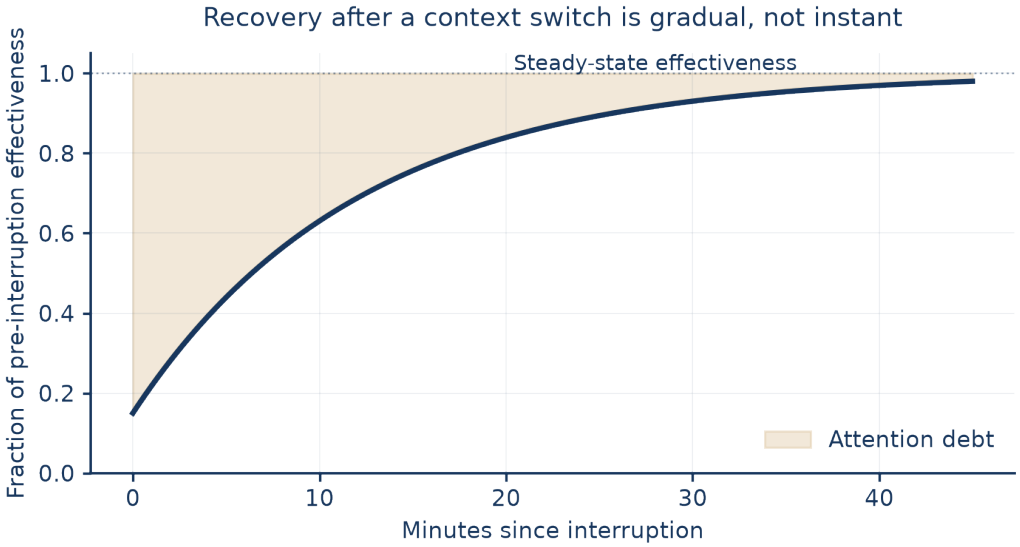
$$T_{recover}(p, e) = \alpha + \beta_1 \Delta t + \beta_2 N_{intervening} + \beta_3 C_e + \beta_4 D_e + \epsilon$$

where:

- Δt is time away from the change;
- $N_{intervening}$ is the number of materially distinct intervening tasks;
- C_e is change complexity;
- D_e is the amount of change to the underlying artifacts while the engineer was away.

The coefficients should be learned from data rather than treated as universal constants.





The shape of the recovery curve above matters more than any single coefficient: the shaded region is attention debt — cumulative work-below-steady-state that the interruption imposes even after the engineer returns to the task. A schedule that appears to fit "just one more meeting" into a heads-down morning is quietly increasing the size of that region rather than trading fifteen minutes for fifteen minutes.

8.5.1. Concurrent work in progress

The number of simultaneously active changes is a useful but imperfect indicator. Two tiny maintenance changes may impose less burden than one complex migration. The model should therefore weight active work by complexity, novelty, urgency, and unresolved state.

Define weighted cognitive inventory:

$$CI_p(t) = \sum_{e \in Active_p(t)} w_e \times u_{p,e}(t)$$

where w_e represents change burden and $u_{p,e}$ represents the engineer's unresolved involvement.

A rising cognitive inventory may predict slower responses, more context recovery, increased defect risk, or abandonment. These are hypotheses to validate, not assumptions to enforce as individual targets.

8.6. Review as cognitive work

Review is not merely elapsed time or comment count. It is a sequence of cognitive operations:

1. construct a mental model of intent;
2. identify changed behavior;
3. connect the change to architecture and domain constraints;

4. search for omissions and failure modes;
5. judge whether tests provide sufficient evidence;
6. communicate findings in actionable form;
7. verify responses and revisions.

Review burden depends on more than lines changed. Important factors include semantic novelty, number of concepts, architectural distance, dependency impact, generated versus handwritten code, test observability, author familiarity, reviewer familiarity, and potential blast radius.

A thousand-line generated client may require less reasoning than a five-line authorization change.

8.7. Comment counts are weak measures

More review comments can indicate careful review, poor code quality, stylistic noise, misunderstanding, or a highly consequential change. Fewer comments can indicate excellent code, superficial review, high trust, or reviewer fatigue.

Comment count becomes useful only when enriched with type, severity, novelty, action, resolution, and downstream outcome.

Recommended comment attributes include:

- defect, design, security, test, maintainability, documentation, or style;
- blocking versus advisory;
- accepted, modified, rejected, deferred, duplicated, or unresolved;
- human- or AI-originated;
- evidence supplied;
- estimated validation effort;
- eventual relationship to rework, defect, or incident.

Enriched this way, comment data reconstructs the intent of a review rather than counting its surface activity. A team can then distinguish between an eight-comment review that caught two defects and an eight-comment review that relitigated formatting, even though a naive count would rank both equally. That distinction is a prerequisite for any downstream claim that AI review either helps or harms.

8.8. AI as a cognitive participant

AI can perform parts of creation, understanding, verification, and explanation. Its economic value depends on whether it reduces total human cognitive work while maintaining or improving outcomes.

This chapter defines two opposing constructs.



Cognitive offload is human cognitive work avoided because a sufficiently reliable AI contribution performs or prepares that work.

Cognitive reload is additional human cognitive work created by the need to inspect, interpret, correct, reject, or recover from AI output.

Net cognitive effect is:

$$NCE(e) = W_{offload}(e) - W_{reload}(e)$$

A positive value indicates net offload. A negative value indicates that the AI increased burden.

8.9. Why acceptance rate is insufficient

Suggestion acceptance rate is useful operational telemetry, but it is not a direct measure of value or offload.

A suggestion may be accepted because it is trivial, automatically applicable, socially persuasive, or difficult to evaluate. A valuable finding may be rejected because the reviewer used the wrong wording or because the author solved the underlying problem differently. A comment may create value without producing a code edit by prompting investigation or confirming safety.

Acceptance telemetry should therefore distinguish:

- direct application;
- application with modification;
- independent equivalent fix;
- follow-up issue creation;
- acknowledgement without change;
- rejection with reason;
- no observable response;
- duplicate of a human finding;
- duplicate of another AI finding.

These categories can be recovered from IDE, review, and version-control telemetry with modest instrumentation, and together they let acceptance be treated as a distribution rather than a single number. The next section introduces an index that uses this richer view to estimate whether AI actually reduced human work.

8.10. A cognitive offload index

A proposed Cognitive Offload Index for an experiment population is:

$$COI = \frac{\hat{W}_{human,control} - \hat{W}_{human,treatment}}{\hat{W}_{human,control}}$$

subject to quality and outcome guardrails.

A COI of 0.15 means estimated human cognitive work fell by fifteen percent relative to the counterfactual. It should never be interpreted without confidence intervals and guardrails.

A stronger quality-adjusted form is:

$$QCOI = COI \times Q_r$$

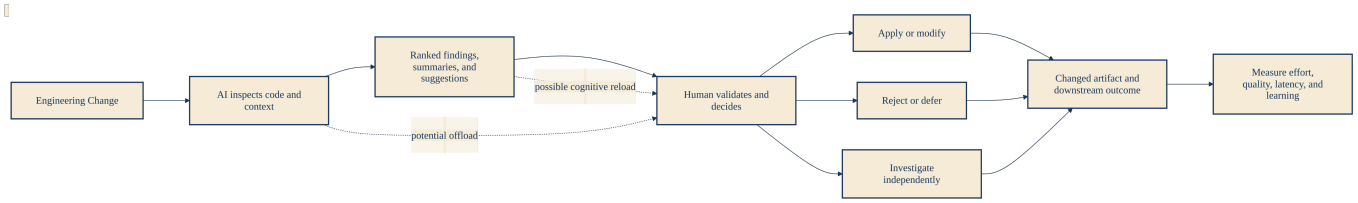
where Q_r is a bounded quality-retention factor. If defects, incidents, or rework worsen, the quality-adjusted offload declines.

The index is intentionally experimental. It is not suitable for ranking individuals or comparing unrelated teams without controlling for work mix.

8.11. Instrumenting AI review interactions

At minimum, an AI review system should emit:

- change and pull-request identifiers;
- model, prompt, policy, and tool versions;
- files and context inspected;
- finding type, confidence, severity, and evidence;
- time the finding became visible;
- human views and expansions;
- code suggestion application events;
- comment replies and resolution;
- subsequent related code edits;
- reviewer duplication;
- follow-up ticket creation;
- downstream quality outcomes;
- inference cost and latency.



Instrumentation should capture interactions, not private chain-of-thought or invasive behavioral data. The objective is to observe work distribution and outcomes.

8.12. Validating cognitive proxies

A latent cognitive measure should be validated in several ways.

8.12.1. Predictive validity

Does it predict resumption time, review latency, rework, error probability, or abandonment better than simpler baselines such as lines changed and file count?

8.12.2. Convergent validity

Do independent proxies move together when cognitive burden plausibly changes?

8.12.3. Discriminant validity

Can the model distinguish cognitive burden from ordinary elapsed time, team size, and repository activity?

8.12.4. Experimental responsiveness

When an intervention specifically designed to reduce burden is introduced, does the measure move in the expected direction while quality remains stable?

8.12.5. Calibration studies

Small voluntary studies can compare inferred burden with structured task reports, observed review sessions, or controlled engineering exercises. Surveys are not required as the primary operational method, but limited calibration data can be useful.

8.13. Experiment designs

Useful experiments include:

- AI review summary versus no summary after long review delays;
- prioritized AI findings versus unranked comment lists;
- one consolidated review pass versus repeated incremental comments;
- automatic low-severity fixes versus comments requiring manual action;
- context-preserving handoff packages for cross-team changes;
- WIP limits for review queues;
- routing specialist attention only to high-risk changes;
- AI-generated change maps for multi-repository work.

Primary outcomes might include human review effort, response cycles, recovery time, quality-adjusted completion time, and downstream rework. Guardrails should include severe defects, security findings, reviewer learning, author understanding, and tool trust.

8.14. Behavioral adaptation

People change behavior in response to AI systems and measurement. Initial acceptance may reflect novelty. Later users may learn when to ignore the tool. Authors may preemptively change code to avoid comments. Reviewers may reduce independent scrutiny because they assume AI covered routine risks.

Evaluation should therefore consider:

- novelty and learning periods;
- long-term trust calibration;
- automation bias;
- skill development or atrophy;
- changes in review allocation;
- movement of work outside the measured interface;
- differential effects by expertise.

A tool that improves short-term speed while weakening human understanding may create long-term risk that is invisible in a short experiment.

8.15. Ethical constraints

Cognitive telemetry can easily become surveillance. Engineering Intelligence requires explicit safeguards:

- collect the minimum data necessary for the decision;
- prefer artifact and workflow events over keystrokes or screen capture;
- aggregate by system, change class, or team where possible;
- prohibit unsupported individual productivity scoring;
- document purpose, retention, access, and downstream use;
- allow correction of inferred states;
- review models for disparate impact;
- separate research telemetry from performance management.

Trust is not merely an ethical concern; it is a measurement requirement. People who believe telemetry will be used against them will alter behavior and degrade the validity of the system.

8.16. Common failure modes

8.16.1. Equating speed with lower cognitive burden

Faster completion can result from skipped verification, automation bias, or simpler work. Quality and work mix must be controlled.

8.16.2. Treating AI output volume as offload

More comments or generated code can increase human burden. Measure the net effect on human work and outcomes.

8.16.3. Using one universal cognitive score

Burden is multidimensional and context-dependent. Preserve component measures and uncertainty.

8.16.4. Ignoring learning

Review contributes to skill development and shared knowledge. Eliminating all human review may reduce immediate cost while weakening the organization.

8.16.5. Measuring individuals rather than systems

Individual-level signals are heavily confounded by task assignment, role, expertise, and hidden coordination. Use them primarily to reconstruct change-level work, not to rank people.

8.17. Worked example: quantifying context-switch cost



Runnable version: `companion/notebooks/03-context-switch-cost.py`.

An eight-engineer team ships to a production service and reports that "meetings are killing us." Leadership wants to know

whether the complaint reflects a measurement gap or a real economic problem.

Instrumentation shows that each engineer averages 3.4 material context switches per working day, with a mean recovery time of 11.5 minutes per switch, derived from the recovery model in this chapter. The loaded rate for the team is $145 per engineer-hour.

Daily attention-recovery cost per engineer:

$$C_{recovery} = 3.4 \times \frac{11.5}{60} \times 145 \approx 94.5 \text{ USD}$$

Team daily cost is $8 \times 94.5 = 756$ USD. Annualised across roughly 230 working days, the team is paying roughly $174,000 per year in pure recovery overhead — before counting the reduced probability of catching a defect during fragmented reviews.

A proposed intervention consolidates status updates and shifts two recurring meetings to written form. Instrumentation after four weeks shows switches per day drop from 3.4 to 2.1 and mean recovery time falls modestly to 10.8 minutes as consolidated blocks let engineers rebuild deeper context. New cost per engineer per day is $2.1 \times (10.8/60) \times 145 = 54.8$ USD, and team annual recovery cost falls to roughly $100,800 — a saving of about $73,000, with a confidence interval that should be reported alongside the point estimate. Whether the saving is worth the coordination trade-offs elsewhere is a separate question, but the economic argument is now visible instead of anecdotal.

8.18. Key takeaways

- Human attention is often the binding constraint in software delivery.
- Cognitive burden includes intrinsic, representational, navigational, coordination, temporal, verification, and tool-related dimensions.
- Context switching creates recovery cost that can be estimated from non-invasive workflow signals.
- Review burden depends on semantic and architectural complexity, not merely code size.
- AI can create both cognitive offload and cognitive reload.
- Acceptance rate alone does not measure AI value.
- Cognitive proxies require predictive, convergent, discriminant, and experimental validation.
- Ethical safeguards and trust are necessary for valid measurement.

8.19. Open questions

- Which event-derived proxies best estimate context recovery without invasive monitoring?
- How can reviewer learning be valued alongside immediate review efficiency?
- What is the long-term effect of AI review on expert skill and automation bias?
- How should cognitive offload be measured when AI and humans iteratively transform the same work?
- Which interventions reduce cognitive inventory without merely shifting burden to another role?

8.20. Implementation checklist

1. Define cognitive burden as a multidimensional construct rather than one score.
2. Separate intrinsic change complexity from process-induced friction.
3. Instrument review interactions, response cycles, task transitions, and resumption events.

4. Record AI findings, human actions, duplication, validation effort, and downstream outcomes.
5. Build simple baselines before introducing complex cognitive models.
6. Validate proxies against resumption time, rework, quality, and controlled experiments.
7. Report confidence intervals and avoid individual rankings.
8. Add quality, learning, and trust guardrails to AI workflow experiments.
9. Establish governance rules before collecting fine-grained behavioral telemetry.
10. Evaluate both cognitive offload and cognitive reload.

8.20.1. Research foundations and source notes

The human-attention model in this chapter combines cognitive psychology, interruption research, developer-experience studies, socio-technical coordination, and queueing theory with several Engineering Intelligence-specific constructs.

Cognitive load as a constrained resource

Cognitive-load theory provides the foundational idea that working cognitive capacity is limited and that task structure affects the amount of capacity available for useful reasoning. See [sweller1988].

The chapter applies this principle to software engineering by treating architecture, unfamiliar code, domain semantics, tool interaction, coordination, and validation as competing consumers of limited attention.

This does not imply that cognitive load can be observed directly from telemetry. It is a latent construct that must be inferred cautiously through multiple indicators, practitioner evidence, and validation studies.

Interruption and resumption cost

Research on interrupted work shows that interruption can change speed, stress, and work patterns, and that returning to a task can require substantial reconstruction. See [mark2008].

The chapter's treatment of context recovery, attention residue, interrupted flow, and fragmented capacity builds on this foundation.

Important distinctions include:

- interruption frequency;
- interruption timing;
- task complexity at interruption;
- delay before resumption;
- number of intervening contexts;
- state preserved by tools and documentation;
- familiarity with the work;
- consequence of lost assumptions.

A short interruption during routine work is not equivalent to an interruption during complex architectural reasoning.

Developer perceptions of productivity

Developer studies show that practitioners frequently associate productivity with progress, focus, reduced blockers, meaningful outcomes, and effective collaboration rather than raw activity counts. See [meyer2014].

The SPACE framework formalizes productivity as multidimensional across satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow. See [forsgren2021-space].

The DevEx framework emphasizes feedback loops, cognitive load, and flow state as important drivers of developer experience and productivity. See [noda2023-devex].

These frameworks support the chapter's rejection of a single universal productivity score and its emphasis on measuring the work system rather than ranking individuals.

Flow, affect, and progress

Research examining developer emotions and progress indicates that being stuck, frustrated, or in flow is related to perceived progress and work experience. See [muller2019].

The chapter therefore treats flow state as an important but difficult-to-measure condition. It should not be inferred from one behavioral signal such as typing speed, application focus, or uninterrupted keyboard activity.

Useful evidence may combine:

- voluntary experience sampling;
- task progress;
- blocked-state evidence;
- build and test feedback latency;
- interruption and queue patterns;
- practitioner interviews;
- qualitative work reconstruction.

Developer experience as a socio-technical construct

Developer experience is produced by interactions among tools, architecture, documentation, ownership, policies, feedback, team practices, and organizational expectations. Early formal work on developer experience is represented by [fagerholm2012], with later operational framing in [noda2023-devex].

This supports the chapter's claim that cognitive burden is not merely an individual characteristic. It can be created or reduced by system design.

Examples include:

- slow or unreliable feedback;
- unclear ownership;
- fragmented documentation;
- excessive approval boundaries;
- poorly designed APIs;
- hidden deployment behavior;
- noisy AI suggestions;

- unstable local environments;
- incompatible team interaction modes.

Architecture and cognitive load

Modular design and information hiding reduce the number of concepts that must be understood simultaneously when boundaries are effective. See [parnas1972].

Team Topologies explicitly connects team design and interaction modes with cognitive load and fast flow. See [skelton2019].

The chapter extends these ideas by treating architectural and organizational boundaries as measurable contributors to attention demand at Engineering Change level.

Coordination attention

Software engineering is a socio-technical activity. Coordination demand emerges from technical dependencies, team boundaries, communication structure, and distributed work.

Conway's Law and socio-technical congruence provide relevant foundations. See [conway1968] and [cataldo2006]. Distributed software-engineering research further documents the importance of coordination across distance and organizational boundaries. See [herbsleb2007].

The chapter's coordination-attention category includes:

- locating owners;
- negotiating interfaces;
- resolving conflicting requirements;
- sequencing releases;
- obtaining decisions;
- maintaining shared situational awareness;
- transferring context across teams.

This attention can be substantial even when implementation effort is small.

Queueing and attention scarcity

Attention demand often appears operationally as a queue for reviewers, architects, security specialists, domain experts, approvers, or incident responders.

Little's Law connects work in progress, throughput, and average flow time. See [little1961]. Kingman's work and the broader operations literature explain why waiting increases rapidly under high utilization and variability. See [kingman1961] and [hopp2011].

This provides the foundation for treating scarce attention pools as constrained service systems rather than simply assigning an hourly labor rate.

A specialist can create high organizational leverage while simultaneously becoming a bottleneck. The appropriate intervention may involve reducing unnecessary demand, creating substitutes, improving routing, standardizing routine cases, or transferring knowledge—not simply asking the specialist to work faster.

Recovery, verification, and coordination effort

The chapter distinguishes four forms of attention:

- production effort;
- verification effort;
- coordination effort;
- recovery effort.

This taxonomy is an Engineering Intelligence-specific synthesis. Its purpose is to expose work that is hidden when all effort is collapsed into authoring time or elapsed cycle time.

Verification effort includes review, testing, validation, challenge, and acceptance. Recovery effort includes rebuilding task state after interruption, delay, failure, or handoff. Coordination effort includes the human work required to align dependencies and decisions.

Attention quality and attention yield

Not all attention produces equal value.

The chapter distinguishes:

- attention consumed;
- attention directed toward the actual constraint;
- useful outcomes produced;
- rework or risk prevented;
- knowledge retained;
- capacity released for other work.

The resulting concepts of **attention yield** and **attention amplification** are original operational constructs in this book.

They are intended to answer questions such as:

- Did specialist review focus on high-risk changes?
- Did automation remove routine verification while preserving judgment?
- Did improved tooling reduce reconstruction and search effort?
- Did an AI assistant reduce net reasoning effort after validation?
- Did a process change merely move attention to another actor?

AI cognitive offload and reload

Studies of AI-assisted programming show that tool effects vary by task, usability, comprehension, debugging burden, and interaction style. See [peng2023], [vaithilingam2022], and [barke2023].

The chapter uses these foundations to distinguish:

- **cognitive offload** — reliable human reasoning or production work genuinely avoided;
- **cognitive reload** — additional work required to inspect, validate, correct, contextualize, or reverse machine output.

Acceptance rate alone cannot establish offload. A suggestion may be accepted mechanically, may solve a trivial problem, or may create downstream maintenance burden. A rejected suggestion may still create useful inspection or learning.

The relevant measure is the net effect on human attention and quality-adjusted Engineering Change outcomes.

Measuring active effort

Active cognitive effort is not directly observable from ordinary engineering telemetry.

The chapter therefore treats active-time reconstruction as a model requiring uncertainty and calibration. Potential inputs include:

- source edits;
- review interactions;
- build and test activity;
- task transitions;
- repository and file navigation;
- voluntary work diaries;
- structured observation;
- practitioner reconstruction interviews.

Telemetry should not be interpreted as a continuous record of thought. Inactivity may represent reasoning, discussion, design, reading, or work outside the instrumented environment.

Privacy and surveillance risk

Attention measurement creates significant privacy and trust risk because the most detailed signals are often individual behavioral traces.

Measurement-validity concerns in [messick1995] and indicator-corruption concerns in [campbell1979] and [goodhart1984] support the chapter's warning against individual ranking.

Privacy-risk management principles are informed by [nist2020-privacy].

The chapter's operating position is that attention evidence should normally be used to improve systems, queues, tools, and organizational design—not to infer individual diligence, commitment, or performance.

Qualitative evidence and triangulation

Because cognitive burden is latent and context-dependent, quantitative telemetry should be combined with qualitative evidence.

Useful methods include:

- practitioner interviews;
- workflow walkthroughs;
- experience sampling;
- incident and project retrospectives;
- observation of representative Engineering Changes;
- review of documentation and ownership paths.

Agreement across several evidence types increases confidence. Disagreement is itself useful because it can expose

instrumentation gaps or invalid assumptions.

Organizational learning

Attention is not only consumed by delivery; it is also invested in learning, mentoring, reflection, and improvement.

Argyris and Schön distinguish single-loop learning, which changes actions within existing assumptions, from double-loop learning, which revises the assumptions themselves. See [argyris1978].

Senge's learning-organization work emphasizes systems thinking and collective capability. See [senge1990].

The chapter therefore treats slack, reflection, mentoring, and knowledge transfer as potential investments rather than automatically classifying them as inefficiency.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific constructs proposed in this book:

- the four-part production, verification, coordination, and recovery attention taxonomy;
- attention scarcity as an Engineering Change economic factor;
- attention yield and attention amplification;
- cognitive offload and cognitive reload as paired AI measures;
- context-recovery estimation from Engineering Change evidence;
- continuity and fragmentation adjustments for released capacity;
- attention routing by risk and uncertainty;
- combined quantitative and qualitative attention validation;
- explicit prohibition against converting attention telemetry into unsupported individual performance scores.

These constructs require empirical calibration and should be reported with uncertainty, boundaries, and acceptable-use controls.

Chapter 9. Measuring Engineering Systems



***Engineering organizations produce abundant telemetry, but telemetry is not measurement. This** chapter defines the measurement hierarchy from events to decisions, explains construct validity, reliability, sensitivity, and robustness, and introduces a taxonomy for flow, quality, economic, human, AI, and architectural metrics. It also covers derived measures, guardrails, metric versioning, statistical treatment, and common anti-patterns.*

9.1. Measurement as a decision discipline

The purpose of measurement is to improve a decision. A dashboard that cannot change a decision is descriptive at best and distracting at worst.

Engineering Intelligence therefore begins every metric definition with a decision question. Examples include:

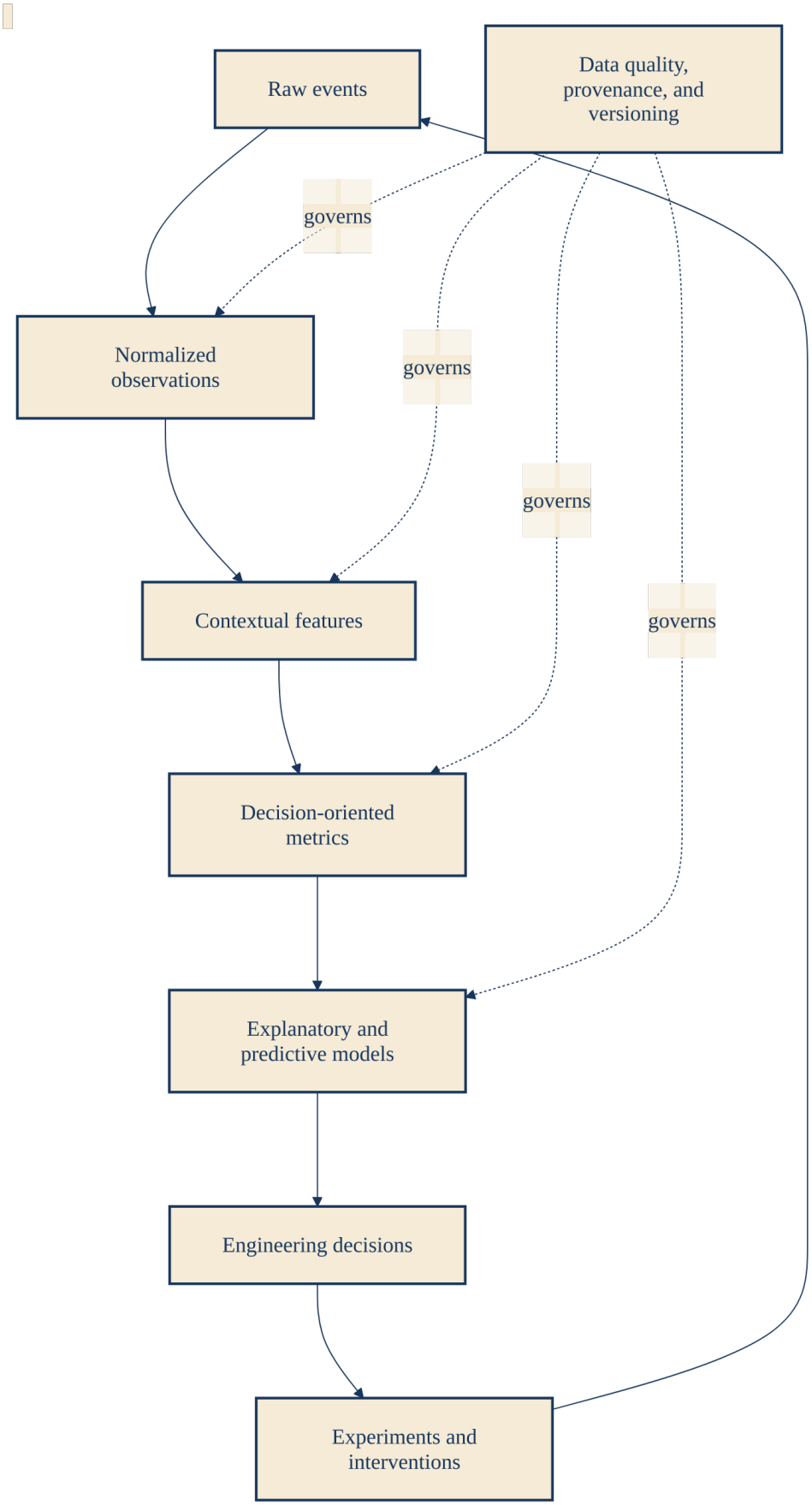
- Should review capacity be added?
- Is CI the dominant constraint?
- Is an AI reviewer reducing total human effort?
- Are changes becoming riskier after a workflow change?
- Which repository boundaries create avoidable coordination cost?

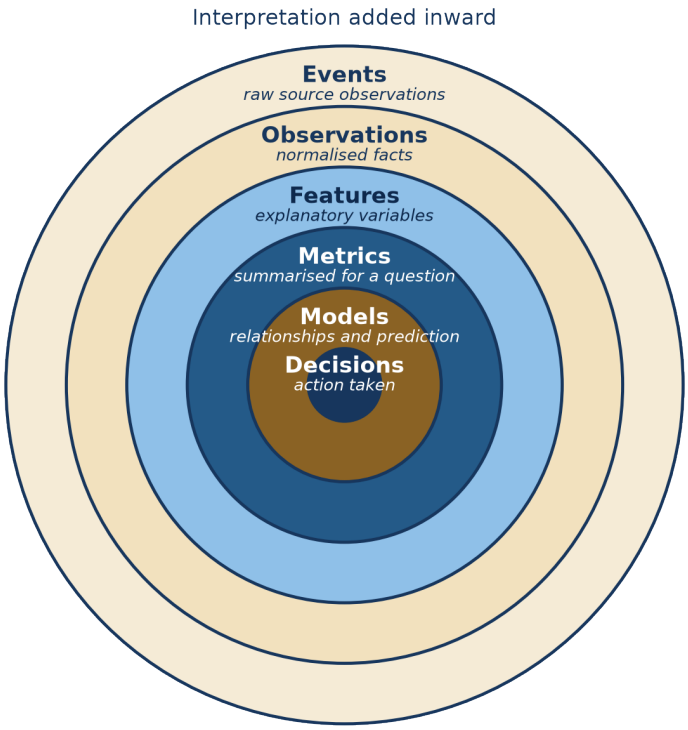
Each question implies a stakeholder, an action that would be taken if the answer changed, and a threshold at which the action becomes justified. A metric that cannot be traced to such a triad is descriptive telemetry, not measurement. The

remainder of this chapter treats every construct — from event to composite index — as machinery in service of one of these questions.

9.2. The measurement hierarchy

Engineering data becomes progressively more useful as interpretation is added:


Raw events
Data quality, provenance, and versioning
governs
Normalized observations
governs
governs
Contextual features
governs
Decision-oriented metrics
Explanatory and predictive models
Engineering decisions
Experiments and interventions



The layers are:

1. **Events** — source observations such as pull-request creation, review submission, CI completion, deployment, or incident declaration.
2. **Observations** — normalized facts derived from events, such as time to first review or number of failed checks.
3. **Features** — explanatory variables such as dependency depth, ownership boundaries, or semantic complexity.
4. **Metrics** — summarized quantities designed to answer a question.
5. **Models** — relationships used to explain, predict, or estimate counterfactual outcomes.
6. **Decisions** — actions taken because of the evidence.

Confusing these layers creates weak analytics. A commit count is an observation. It becomes a metric only when attached to a valid construct and decision.

9.3. Properties of a good metric

9.3.1. Decision relevance

A metric should inform a specific operational, strategic, or experimental decision.

9.3.2. Construct validity

The metric should measure the concept it claims to represent. Pull-request count has low validity as a general productivity measure because it is strongly affected by repository and slicing conventions.

9.3.3. Reliability

Repeated measurement under equivalent conditions should produce similar values. Instrumentation gaps, inconsistent timestamps, and identity errors reduce reliability.

9.3.4. Sensitivity

A metric should respond when the underlying construct changes. A review-friction measure that does not move after a major queue reduction has weak sensitivity.

9.3.5. Robustness

A metric should resist gaming and remain interpretable across reasonable workflow differences.

9.3.6. Explainability

Users should be able to determine why a metric changed and drill down to its component evidence.

9.3.7. Stability

Noise should be quantified. Important measures should include intervals, sample size, and control limits where appropriate.

9.3.8. Collection cost

Instrumentation, storage, computation, governance, and human correction all have costs. High-cost measures should support high-value decisions.

9.4. Metric taxonomy

Category	Examples
Flow	Lead time, active time, queue time, review latency, deployment latency, work in process, critical-path duration
Quality	Escaped defects, change-failure rate, rollback rate, severity-weighted incidents, security findings, rework
Economic	Human attention cost, AI cost, infrastructure cost, coordination cost, cost of delay, expected failure loss
Human	Context recovery, concurrent cognitive inventory, review burden, scarce-expertise utilization
AI	Finding precision, recall, duplication, validation burden, action rate, cognitive offload, cognitive reload

Category	Examples
Architecture	Dependency depth, fan-in, fan-out, blast radius, repository spread, ownership boundaries, deployment coupling

No category should be optimized independently. Faster flow with worse quality is not an improvement; lower review cost with greater cognitive reload may not create value.

9.5. Direct and derived metrics

A direct metric is computed closely from observed events. Examples include build duration, time to first review, or number of deployment attempts.

A derived metric depends on a model. Examples include cognitive burden, expected cost of delay, blast-radius score, or Engineering Change complexity.

Derived metrics are often more decision-relevant but require stronger governance:

- published definitions;
- model version;
- training and calibration data;
- confidence intervals;
- known limitations;
- validation evidence;
- recomputation history.

Without this scaffolding a derived metric behaves like a black box that quietly changes meaning when its underlying model is retrained. Governance is not administrative overhead: it is what allows a Cognitive Offload Index or a blast-radius score to survive scrutiny in a change-management review or a post-incident audit. Direct metrics can be re-derived from raw events; derived metrics can only be trusted to the extent that their assumptions are stated and traceable.

9.6. Leading and lagging measures

Leading indicators change before the outcome of interest. Review queue age, CI instability, and dependency growth may precede delivery or quality problems.

Lagging indicators confirm realized outcomes. Incidents, escaped defects, customer impact, and realized cost appear after release.

A strong scorecard pairs them. Leading indicators support intervention; lagging indicators validate whether the intervention mattered.

9.7. Local and system-level measures

Local measures describe one stage, team, repository, or participant. System-level measures describe the complete Engineering Change or portfolio.

Local optimization can harm the system. For example, reducing CI cost by running fewer tests may increase rework and incidents. Reducing reviewer effort may shift burden to authors.

Engineering Intelligence preserves both views and makes the relationship explicit.

9.8. Composite metrics

Composite metrics summarize several related signals. They should not conceal their components.

A Review Friction Index might combine normalized queue age, response cycles, unresolved comments, context-recovery events, and verification effort:

$$RFI(e) = \sum_{k=1}^{K} w_k z_k(e)$$

where z_k are standardized components and w_k are calibrated weights.

Weights should be learned or justified, not chosen merely to make the result look plausible. Users must be able to inspect component values.

9.9. Guardrail metrics

Every optimization metric should have guardrails.

An AI reviewer experiment may target lower human review effort while guarding against increases in severe defects, rollback rate, security escapes, author misunderstanding, and long-term reviewer disengagement.

A change is considered beneficial only if the primary outcome improves and guardrails remain within predeclared tolerances.

9.10. Statistical treatment

Engineering metrics are usually skewed, heavy-tailed, seasonal, and clustered by team and repository. Means alone are rarely sufficient.

Recommended summaries include:

- median and selected quantiles;
- confidence or credible intervals;
- sample size and missingness;
- distribution plots;
- control limits;
- stratification by change class;
- hierarchical estimates for sparse groups.

9.10.1. Control charts

Control charts help distinguish common-cause variation from exceptional changes. They are useful for operational monitoring when metric definitions and populations remain stable.

9.10.2. Time-series effects

Seasonality, release freezes, reorganizations, holidays, incidents, and tool migrations can create apparent trends. Time-series models should record external events and definition changes.

9.10.3. Bayesian updating

Bayesian models are useful when evidence arrives sequentially, sample sizes are small, or prior domain knowledge matters. They support direct statements such as the probability that an intervention improved a metric by at least a meaningful threshold.

9.11. Metric versioning

A metric without a version is not reproducible evidence.

Version changes may include event-source changes, identity-resolution changes, new exclusion rules, updated reconstruction models, cost assumptions, or revised severity weights.

Every material metric should retain:

- metric name and semantic version;
- definition and formula;
- source schemas;
- population and exclusions;
- model version;
- computation time;
- lineage to raw evidence.

Semantic versioning conveys the nature of a change: a patch corrects computation, a minor version adds a compatible refinement, a major version breaks historical comparability. A dashboard that mixes versions without labeling them silently confuses evidence. Where historical continuity matters, retain the prior version alongside the new one and backfill only when the semantics genuinely align.

9.12. Comparison and normalization

Comparisons should control for work mix. Relevant variables include Engineering Change complexity, potential blast radius, repository topology, team count, domain, novelty, and deployment type.

Raw comparisons between teams are usually invalid because teams receive different work and operate under different constraints.

Normalization can use stratification, regression adjustment, matching, hierarchical models, or experimental assignment. It should never erase meaningful heterogeneity.

9.13. Measurement anti-patterns

9.13.1. Activity as productivity

Commit count, lines changed, comments, and pull-request volume are easy to count and easy to game. They should not

become universal performance measures.

9.13.2. Individual ranking

System telemetry is confounded by assignment, role, complexity, and invisible work. Individual league tables damage trust and measurement validity.

9.13.3. One universal score

Engineering effectiveness is multidimensional. A single score hides trade-offs and invites optimization against the formula.

9.13.4. Silent definition changes

Historical comparisons become invalid when definitions change without versioning and backfill.

9.13.5. Correlation as causation

A team using a tool may outperform because it adopted the tool early, has stronger leadership, or receives different work. Experiments or causal methods are required.

9.13.6. Dashboard accumulation

Adding metrics without retiring unused ones creates noise. Every dashboard should have an owner, audience, decision, and review date.

9.14. Designing a measurement catalogue

Each metric specification should include:

Field	Purpose
Name and version	Stable identifier for reproducibility
Decision	Question the metric informs
Construct	Concept being measured
Formula	Exact computation
Unit and population	Scope and denominator
Sources	Required events and entities
Latency	Near-real-time, daily, weekly, or retrospective
Validation	Evidence of validity and reliability

Field	Purpose
Bias and gaming	Known failure modes
Guardrails	Outcomes that must not worsen
Owner	Accountability for definition and maintenance

9.15. Example: measuring review latency

A weak definition is "time from pull-request open to first review."

A stronger definition specifies business-time handling, draft exclusions, bot reviews, reviewer assignment, author self-review, reopened pull requests, time zones, and whether the unit is pull request or Engineering Change.

The metric should be paired with active review effort, queue age, complexity, reviewer capacity, and quality outcomes. A decrease can mean better flow or superficial review.

9.16. Example: measuring AI review quality

For labeled findings:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Ranking quality can be measured with NDCG when high-severity findings should appear first. Operational analysis should also include duplication, action type, validation effort, latency, and downstream outcome.

High precision with very low recall may miss important defects. High recall with poor ranking may overwhelm authors. No single number is sufficient.

9.17. Worked example: the cost of an unversioned metric change



Runnable versions of this example live in the companion notebooks: * `companion/notebooks/09-metric-versioning.py` * `companion/notebooks/10-composite-metric.py` * `companion/notebooks/11-bayesian-metric.py`

An organization reports "review latency" on a leadership scorecard. In Q2, latency drops from 8.4 hours to 5.9 hours — a 30% improvement widely attributed to the AI reviewer rolled out in that quarter.

Investigation reveals that in April the metric owner also changed the eligible population: draft pull requests and bot-authored dependency bumps, which had previously counted, were excluded. Historical recomputation with the new definition shows the true Q1 latency was 6.7 hours, not 8.4 — the "improvement" was 0.8 hours, not 2.5.

The economic implications are asymmetric. The AI reviewer's $1.2M annual cost was justified in part by the reported 30% improvement. The actual 12% improvement changes the expected annual net benefit from roughly $650k to roughly

$95k — a decision that would have been reversed had the metric been governed properly. A single unversioned metric change misdirected the equivalent of half a year of investment.

The remediation is procedural, not analytical: any change to a scorecard metric increments a version number, retains the prior version alongside the new one, and re-scores the last four quarters under both definitions. The dashboard displays both series until the review committee accepts the new version. In this case the correct label would have been "review-latency v2.0 (new population); v1.0 shown for comparison." The AI-reviewer decision would then have used v1.0-consistent numbers and reached a different conclusion.

9.18. Key takeaways

- Telemetry becomes measurement only when connected to a construct and decision.
- Good metrics are relevant, valid, reliable, sensitive, robust, explainable, stable, and worth their collection cost.
- Flow, quality, economics, human factors, AI, and architecture must be evaluated together.
- Derived metrics require model governance, versioning, uncertainty, and validation.
- Guardrails prevent local optimization from damaging the wider system.
- Comparisons must control for work mix and organizational context.
- Metric catalogues should be governed products, not undocumented queries.

9.19. Open questions

- Which constructs can be measured reliably without invasive telemetry?
- How should organizations retire metrics that have become targets and lost validity?
- What normalization methods best preserve meaningful differences while enabling comparison?
- How can metric catalogues remain understandable as models and schemas evolve?
- Which leading indicators provide actionable warning without excessive false alarms?

9.20. Implementation checklist

1. Inventory current dashboards and identify the decision supported by each metric.
2. Separate events, observations, features, metrics, models, and decisions in the data architecture.
3. Create a versioned metric catalogue with lineage and owners.
4. Add uncertainty, sample size, and missingness to important measures.
5. Define guardrails for every optimization objective.
6. Prohibit unsupported individual productivity rankings.
7. Stratify or adjust comparisons for Engineering Change context.
8. Validate derived measures against simpler baselines and downstream outcomes.
9. Record external events and definition changes in time-series analysis.
10. Retire metrics that no longer support a decision.

9.20.1. Research foundations and source notes

The measurement framework in this chapter draws on classical measurement theory, psychometrics, construct validity, indicator governance, developer-productivity research, and statistical reasoning. It extends those foundations into a

governed metric system for software engineering.

Measurement scales and permissible operations

Stevens's classification of nominal, ordinal, interval, and ratio scales provides the basic framework for understanding which mathematical operations are meaningful for a measure. See [stevens1946].

The practical implication is that engineering data types that look numeric are not automatically interchangeable.

Examples include:

- severity classes may be ordinal rather than interval;
- confidence bands may be ordered categories rather than probabilities;
- counts may be ratio-scale values but still require exposure or population normalization;
- ranks preserve order but not meaningful distance;
- composite scores may not support multiplication or ratio interpretation.

The chapter therefore requires every metric specification to identify its unit, scale, aggregation behavior, and interpretation.

Construct definition

A construct is the underlying concept an organization seeks to understand, such as maintainability, cognitive load, delivery responsiveness, review effectiveness, or quality.

Messick's treatment of validity emphasizes that validity concerns the evidence supporting an intended interpretation and use, not merely the mechanical correctness of a score. See [messick1995].

Hand provides a broader treatment of how measurement choices shape what can be observed and concluded. See [hand2004].

This supports a central rule in the chapter:

> A metric is not valid in the abstract. It is more or less valid for a specific construct, population, decision, and consequence.

Reliability

Reliability concerns the consistency and reproducibility of measurement.

Cronbach's coefficient alpha is one influential method for evaluating internal consistency among indicators. See [cronbach1951]. The broader principle is that a measure should behave consistently enough for its intended use.

Engineering examples include:

- whether independent labelers agree on review findings;
- whether a reconstruction model produces stable Engineering Change membership;
- whether a survey scale behaves consistently across repeated administrations;
- whether a metric changes because the system changed or because instrumentation changed;
- whether identical evidence produces the same metric version.

Reliability is necessary but not sufficient. A consistently wrong measure can be highly reliable and invalid.

Content, criterion, and construct evidence

The chapter's validation model includes several forms of evidence.

Content evidence asks whether the metric covers the important dimensions of the construct.

Criterion evidence asks whether the metric relates appropriately to an external outcome or reference.

Convergent evidence asks whether it agrees with different measures of the same construct.

Discriminant evidence asks whether it remains distinct from related but different constructs.

Predictive evidence asks whether it forecasts relevant future outcomes.

Consequential evidence asks what happens when the metric is used in practice.

This integrated treatment follows the use-oriented validity perspective in [messick1995].

Measurement invariance

A metric may not mean the same thing across teams, technologies, work types, or time.

Examples include:

- story points with different local semantics;
- pull-request size under different repository strategies;
- review latency where review policies differ;
- incident counts where reporting culture differs;
- AI acceptance where tools expose different suggestion units;
- survey responses across organizational cultures.

The chapter therefore requires subgroup, temporal, and workflow checks before cross-population comparison.

Measurement invariance is not guaranteed by a shared column name or dashboard label.

Multidimensional engineering productivity

The SPACE framework argues that developer productivity should be considered across satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow. See [forsgren2021-space].

The DevEx framework emphasizes feedback loops, cognitive load, and flow state. See [noda2023-devex].

Developer-experience research also supports the importance of practitioner perception and context. See [fagerholm2012] and [meyer2014].

These foundations support the chapter's metric-bundle approach rather than a universal productivity score.

A useful bundle often includes:

- primary outcome;
- quality and safety guardrails;

- mechanism indicators;
- attention or experience measures;
- adoption and exposure measures;
- economic measures;
- data-quality measures.

Delivery performance

Delivery-performance measures such as deployment frequency, lead time, restoration time, and change-failure behavior are important system-level outcomes. The research synthesis in [forsgren2018] provides the principal foundation used by this book.

These measures should still be interpreted with care:

- definitions differ across systems;
- repository and service boundaries matter;
- release and deployment are not always identical;
- work mix and risk vary;
- averages may conceal tails and subgroups;
- local optimization can damage another outcome.

The chapter therefore treats delivery measures as governed metrics within a larger system rather than complete proxies for engineering performance.

Queue and flow measurement

Little's Law relates average work in progress, throughput, and flow time. See [little1961].

Kingman's work and the operations literature describe the nonlinear effect of utilization and variability on waiting. See [kingman1961] and [hopp2011].

These foundations support measures such as:

- queue depth;
- wait time;
- service time;
- blocked time;
- work in progress;
- throughput;
- arrival and service variability;
- tail latency;
- utilization of constrained resources.

They also explain why reducing one stage's average duration may not improve end-to-end flow.

Indicator corruption and gaming

Campbell warned that when a quantitative indicator is used heavily for social decision-making, it becomes vulnerable to corruption and can distort the process it is intended to monitor. See [campbell1979].

Goodhart's Law expresses a related principle: once a measure becomes a target, it may cease to function as a useful measure. See [goodhart1984].

Engineering examples include:

- increasing commit count by splitting trivial changes;
- reducing review time by approving superficially;
- reducing incident counts by discouraging incident declaration;
- improving story-point velocity by changing estimates;
- increasing AI acceptance by suggesting trivial edits;
- reducing cycle time by excluding blocked or failed work.

The chapter therefore requires gaming analysis, balancing measures, and prohibited uses in the metric specification.

Missing data

Missing values have different meanings.

A field may represent:

- true zero;
- not applicable;
- not instrumented;
- not observed;
- delayed evidence;
- unresolved linkage;
- suppressed data;
- failed processing;
- outcome not yet mature.

Collapsing these states into zero or null can materially bias a metric.

The chapter treats missingness as part of metric semantics and quality rather than a downstream cleanup issue.

Denominators and eligibility

A rate is only meaningful when its denominator is governed.

For example, AI-finding acceptance may be defined over:

- all generated findings;
- findings shown to a human;
- findings eligible for disposition;
- unique findings after deduplication;

- findings with mature outcomes;
- findings within supported languages or repositories.

Different denominators answer different questions.

The chapter therefore requires executable eligibility and denominator definitions for every rate or proportion.

Aggregation and ecological error

Group-level patterns do not automatically describe individuals, and aggregate comparisons can reverse when population composition changes.

The chapter's work-mix adjustment, subgroup analysis, and restriction on individual interpretation are designed to reduce ecological and compositional errors.

A team with longer cycle time may handle more complex, risky, or cross-system changes. A portfolio average can improve because easy work increased rather than because the process improved.

Distributions rather than averages

Engineering measures are often skewed, heavy-tailed, censored, or multimodal.

The chapter therefore recommends reporting:

- medians and quantiles;
- tail probabilities;
- distribution plots;
- stratification by relevant work classes;
- uncertainty intervals;
- sample and eligibility counts;
- mature versus immature outcomes.

Averages remain useful, but they should not be the sole representation of flow, cost, review effort, or incident impact.

Precision, recall, and ranking

When engineering tools produce findings or ranked recommendations, information-retrieval measures become relevant.

Precision and recall are grounded in the standard information-retrieval treatment in [manning2008].

Discounted cumulative gain and normalized discounted cumulative gain support evaluation of ranked results where the position and graded relevance of findings matter. See [jarvelin2002].

The chapter's evaluation approach also requires:

- adjudication rules;
- duplicate handling;
- partial relevance;
- severity weighting;
- finding-level and change-level analysis;

- confidence intervals;
- disagreement review.

Calibration

A probabilistic score should be interpreted through calibration as well as discrimination.

A model assigning 0.8 probability should be correct approximately 80 percent of the time in the relevant calibrated population.

The chapter extends this principle beyond predictive models to:

- Engineering Change linkage confidence;
- defect-risk estimates;
- blast-radius predictions;
- finding-confidence scores;
- automated policy thresholds.

Calibration must be evaluated by population and over time because drift can invalidate earlier interpretations.

Metric lineage and reproducibility

A published metric should be traceable to:

- source evidence;
- source and schema versions;
- identity and linkage logic;
- transformations;
- metric version;
- quality state;
- publication time;
- correction history.

This requirement is compatible with the provenance concepts in [moreau2013] and the replayable data-system principles in [kleppmann2017] and [akidau2018].

A metric without a reproducible definition and evidence path is a report, not decision-grade evidence.

Metric governance

Metric governance is part of measurement quality because access, purpose, interpretation, and consequence affect the validity of use.

The chapter therefore requires:

- accountable owner;
- decision purpose;
- permitted and prohibited uses;

- quality threshold;
- review cadence;
- versioning and deprecation;
- correction procedure;
- privacy and aggregation controls.

Privacy-risk principles are informed by [nist2020-privacy], while professional concerns around harm, fairness, honesty, and accountability are informed by [acm2018-ethics].

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- a metric as a versioned decision product rather than a query;
- a required decision-purpose field;
- executable eligibility and denominator contracts;
- explicit prohibited-use declarations;
- a quality state published with every metric result;
- policy-grade metric maturity;
- integration of source, linkage, identity, and outcome maturity into metric quality;
- metric incidents and rollback;
- decision-specific validity thresholds;
- metric bundles connecting outcomes, guardrails, mechanisms, economics, exposure, and quality;
- mandatory distinction among observed, reconstructed, inferred, predicted, and causal values.

These constructs should be validated through practical use, correction records, decision outcomes, practitioner review, and periodic independent audit.

Chapter 10. Experimentation and Causal Inference



Engineering organizations frequently introduce new tools, review policies, repository structures, *and AI-assisted workflows, then compare before-and-after averages and declare success. This chapter explains why that approach is often misleading. It develops an experimentation framework for Engineering Intelligence, covering randomized trials, cluster assignment, staggered rollout, switchback designs, difference-in-differences, interrupted time series, synthetic controls, matching, heterogeneous treatment effects, statistical power, guardrails, interference, compliance, and the infrastructure required to evaluate engineering interventions at enterprise scale.*

10.1. Why observation is not enough

Suppose review latency falls after an AI reviewer is introduced. The tool may have caused the improvement, but several alternatives are plausible:

- the organization entered a quieter release period;
- repositories with simpler work adopted first;
- reviewer staffing increased;
- CI became faster at the same time;
- teams changed pull-request slicing conventions;
- engineers were temporarily more attentive because the tool was new;

- low-performing teams declined to participate.

The causal question is not whether the metric changed. It is whether the metric would have changed differently without the intervention.

For unit i, define potential outcomes:

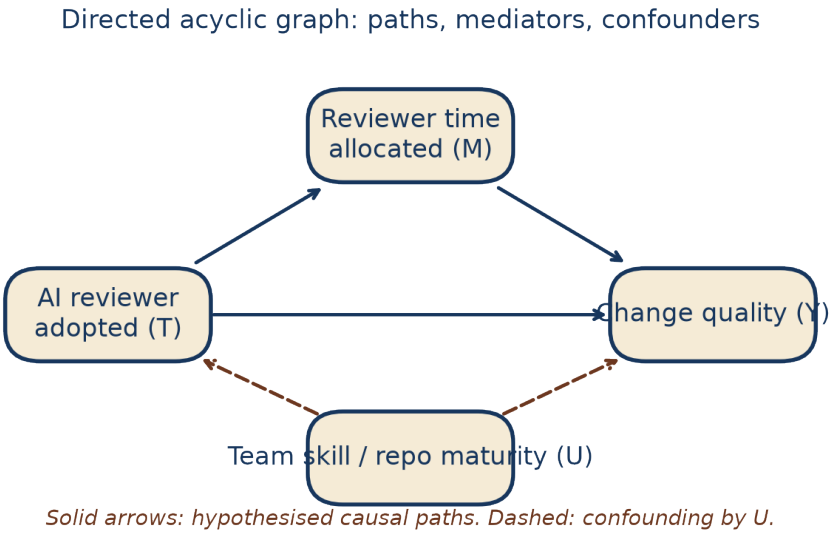
$$Y_i(1) = \text{outcome under treatment}$$

$$Y_i(0) = \text{outcome without treatment}$$

The individual treatment effect is:

$$\tau_i = Y_i(1) - Y_i(0)$$

Only one potential outcome is observed. Experimentation and causal inference provide disciplined ways to estimate the missing counterfactual.



The directed acyclic graph above makes the alternatives explicit. Adoption of the AI reviewer (T) affects the outcome (Y) both directly and through the mediator of allocated reviewer time (M). A common cause — team skill or repository maturity (U) — influences both adoption and outcome, creating a confounding path. Estimating the causal effect of T on Y requires either randomising T (which cuts the confounding arrow into T) or adjusting for U (which blocks the backdoor path). Drawing this DAG before an experiment is designed forces the assumptions to become visible before the numbers do.

10.2. Define the intervention precisely

"Use AI review" is not a sufficiently precise treatment definition. An experiment must specify:

- vendor, model, and model version;
- prompt and policy configuration;
- repositories and file types in scope;
- whether findings are advisory or blocking;
- when findings appear;

- ranking and suppression rules;
- human-review requirements;
- exposure start and end;
- fallback behavior during failures.

Without treatment fidelity, results cannot be reproduced or generalized.

10.3. Unit of assignment and unit of analysis

The assignment unit receives treatment. The analysis unit contributes an outcome. They may differ.

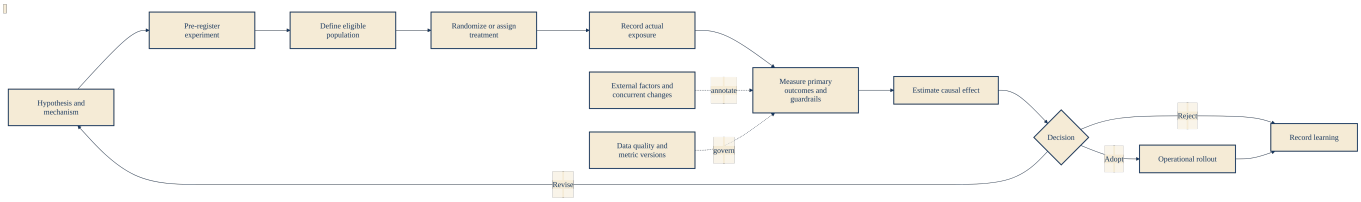
Possible assignment units include engineer, pull request, Engineering Change, repository, team, organization, or time interval.

Individual randomization is statistically efficient but can suffer contamination when reviewers and repositories are shared. Team-level randomization reduces contamination but requires more clusters. Pull-request assignment can work for isolated tool behavior, while Engineering Change assignment is preferable for multi-repository outcomes.

The choice must account for interference: one unit's treatment may affect another unit's outcome.

10.4. Randomized controlled experiments

Random assignment balances observed and unobserved confounders in expectation.



A basic estimator of average treatment effect is:

$$\widehat{ATE} = \bar{Y}_T - \bar{Y}_C$$

where $\bar{Y}_T$ and $\bar{Y}_C$ are treatment and control means.

Randomization should be stratified when important dimensions strongly predict outcomes. Examples include language, repository size, change risk, team, and baseline review latency.

10.5. Cluster-randomized experiments

When treatment is shared within a team or repository, randomize clusters rather than individuals.

Clustered outcomes are correlated, reducing effective sample size. The design effect is approximately:

$$DE = 1 + (m - 1)\rho$$

where m is average cluster size and ρ is intra-cluster correlation.

Ignoring clustering produces confidence intervals that are too narrow.

10.6. Switchback experiments

In a switchback design, the same population alternates between treatment and control across time blocks. This can be useful for merge-queue policies, CI routing, review prioritization, or operational processes.

Switchbacks control for stable differences between units but require attention to carryover effects, learning, and seasonality. Time blocks must be long enough for the treatment to operate and short enough to avoid major external drift.

10.7. Staggered rollout

A staggered rollout introduces treatment to groups at different times. It supports operational safety and can create useful causal variation.

Analysis must account for changing treatment effects and calendar-time shocks. Naive two-way fixed-effects estimators can be misleading when adoption timing and effects vary. Cohort-aware estimators or modern event-study methods are preferable.

10.8. Difference-in-differences

Difference-in-differences compares the before-after change in a treatment group with the change in a comparison group:

$$\widehat{\tau}_{DiD} = (\bar{Y}_{T,after} - \bar{Y}_{T,before}) - (\bar{Y}_{C,after} - \bar{Y}_{C,before})$$

The key assumption is parallel trends: absent treatment, the groups would have changed similarly.

Analysts should inspect pre-treatment trends, include relevant covariates, test alternative windows, and report sensitivity to plausible violations.

10.9. Interrupted time series

Interrupted time series evaluates whether an intervention changed the level or slope of a metric relative to its previous trend.

It is useful for enterprise-wide changes where no untreated population exists. It is vulnerable to concurrent events, autocorrelation, seasonality, and anticipatory behavior.

An external-events registry is essential. Reorganizations, release freezes, major incidents, staffing changes, acquisitions, and platform migrations should be recorded and incorporated into analysis.

10.10. Synthetic controls and matched comparisons

Synthetic control constructs a weighted combination of untreated units that resembles the treated unit before intervention. It is useful for large team, organization, or platform changes with few treated units.

Matching selects comparison units with similar pre-treatment characteristics. Useful features include baseline flow, work mix, repository topology, team size, domain, and adoption propensity.

Matching does not remove unobserved confounding. Balance diagnostics and sensitivity analysis remain necessary.

10.11. Regression discontinuity

If treatment is assigned using a threshold, such as mandatory review above a risk score, outcomes just above and below the threshold may be compared.

This design can estimate a credible local effect when the threshold cannot be manipulated precisely and other processes remain continuous around it.

10.12. Primary outcomes and guardrails

Experiments should have a small number of predeclared primary outcomes. Examples include quality-adjusted cycle time, human review effort, severe escaped-defect rate, or cost per accepted Engineering Change.

Guardrails prevent local optimization. For an AI reviewer experiment, guardrails might include:

- severe defects;
- security escapes;
- rollback rate;
- author understanding;
- reviewer learning;
- cognitive reload;
- tool trust;
- infrastructure cost.

A faster process is not successful if important guardrails deteriorate beyond tolerance.

10.13. Effect size and practical significance

Statistical significance does not establish practical value. With very large samples, trivial effects can become statistically significant.

Define a minimum practically important difference before the experiment. For example, a tool may need to reduce quality-adjusted human effort by at least five percent to justify migration and governance costs.

Report absolute effects, relative effects, uncertainty intervals, and economic consequences.

10.14. Power and sample size

Power is the probability of detecting an effect of a specified size when it exists.

For a two-group comparison of means, required sample size grows with outcome variance and shrinks with larger expected effect. Cluster assignment, rare defects, and heavy-tailed outcomes often require much larger samples than teams expect.

For rare severe defects, use longer observation windows, composite severity-weighted outcomes, Bayesian models, or historical augmentation. Do not replace meaningful rare outcomes with easy but weak proxies solely to obtain significance.

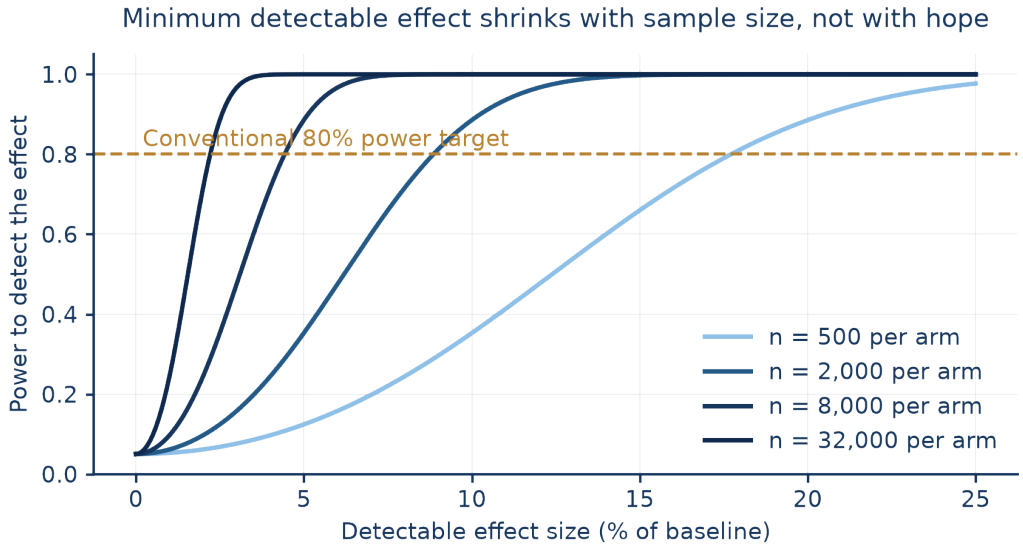


Figure 6 shows the practical implication. A study with 500 changes per arm cannot reliably detect anything smaller than about a 15% shift in outcome; even 8,000 per arm leaves proportional effects below 5% underpowered. The minimum detectable effect, not the observed effect, is what determines whether a negative result is evidence of no effect or simply evidence of insufficient data. Power analyzes belong in the pre-registration, not in the discussion.

10.15. Sequential testing

Engineering experiments often run continuously and are inspected repeatedly. Repeated naive testing inflates false positives.

Use group-sequential boundaries, alpha-spending, always-valid confidence sequences, or Bayesian stopping rules. Stopping criteria should be declared before observing results.

10.16. Multiple comparisons

Testing many metrics, subgroups, prompts, and tools increases the chance of false discoveries.

Approaches include family-wise error control, false-discovery-rate procedures, hierarchical testing, and predeclared primary hypotheses. Exploratory results should be labeled as exploratory and replicated.

10.17. Treatment exposure and compliance

Assignment is not the same as exposure. A developer assigned an AI reviewer may never open its findings. A control team may use another AI tool.

Record:

- assignment;
- eligibility;
- actual exposure;
- intensity of use;

- failures and fallback;
- cross-over;
- protocol deviations.

Intent-to-treat analysis estimates the effect of offering or assigning treatment. Per-protocol and treatment-on-the-treated analyzes answer different questions and require stronger assumptions.

10.18. Interference and network effects

The stable-unit-treatment-value assumption often fails in engineering systems. Reviewers work across teams, shared libraries affect many repositories, and treated engineers influence untreated peers.

Possible strategies include cluster randomization, graph-cluster randomization, saturation designs, explicit spillover estimation, and excluding highly connected shared infrastructure from simple experiments.

Interference is not merely a nuisance. Spillover may be a valuable effect, such as knowledge transfer from AI-assisted reviewers to the wider team.

10.19. Heterogeneous treatment effects

Average effects can conceal important variation. Estimate effects by:

- change complexity;
- blast radius;
- programming language;
- repository topology;
- domain;
- author and reviewer experience;
- AI finding type;
- baseline process maturity.

Subgroup analysis should be predeclared when possible and regularized to avoid noisy conclusions. Hierarchical models can partially pool estimates.

10.20. External factors

Engineering cycles are affected by factors outside the measured loop:

- product launches;
- seasonal demand;
- organizational restructuring;
- staffing and attrition;
- regulatory deadlines;
- incidents;
- infrastructure outages;
- new coding assistants;
- changes in work mix;

- macroeconomic constraints.

The experimentation platform should maintain an external-factor timeline with scope, severity, start and end time, affected populations, and evidence. These factors can be modeled as covariates, exclusions, stratification variables, or explicit interruptions.

10.21. Experiment registry

Every experiment should be registered before launch with:

- hypothesis;
- intervention definition;
- owner;
- assignment and analysis units;
- eligible population;
- primary outcomes;
- guardrails;
- expected effect;
- power analysis;
- start, stop, and stopping rules;
- exposure telemetry;
- concurrent changes;
- analysis plan;
- decision rule.

```
{
  "experimentId": "EXP-AI-REVIEW-014",
  "hypothesis": "Ranked AI findings reduce human review effort without increasing severe
defects",
  "assignmentUnit": "engineering_change",
  "primaryOutcome": "quality_adjusted_human_review_minutes",
  "guardrails": ["severe_escape_rate", "rollback_rate", "cognitive_reload"],
  "minimumPracticalEffect": -0.05,
  "analysis": "cluster_randomized_hierarchical",
  "status": "registered"
}
```

10.22. Experiment infrastructure

A mature platform requires:

- population and eligibility service;
- deterministic assignment;
- configuration delivery;
- exposure logging;
- metric and guardrail computation;
- external-events registry;

- sequential-monitoring service;
- analysis notebooks or pipelines;
- decision and rollout records;
- audit and reproducibility controls.

Assignment should be stable, versioned, and independent of outcome pipelines. Exposure must be joined to the Engineering Change graph.

10.23. Example experiment: AI comment prioritization

Hypothesis: showing only the highest-ranked five AI findings reduces validation burden while preserving severe-defect detection.

Treatment: top-five ranked findings with expandable remainder. Control: all findings displayed chronologically.

Assignment: Engineering Change, clustered by repository to reduce contamination.

Primary outcome: human validation minutes per accepted change.

Guardrails: severe finding recall, post-merge defects, unresolved blocking comments, and author action latency.

Secondary outcomes: comment duplication, rejection reasons, tool trust, and inference cost.

The analysis should estimate average and heterogeneous effects by change complexity and finding volume.

10.24. Example experiment: specialist review routing

Hypothesis: risk-based routing of database specialists reduces queue delay without increasing migration incidents.

Treatment: specialist approval required only above a preregistered risk threshold. Control: specialist approval required for all schema changes.

Possible design: regression discontinuity around the risk threshold, supplemented by randomized review for a subset near the boundary.

Primary outcome: quality-adjusted time to release.

Guardrails: severe migration incidents, rollback, data repair, and false-negative risk classification.

10.25. Common failure modes

10.25.1. Before-and-after attribution

A metric changed after rollout, therefore the tool caused it. Concurrent changes make this claim weak.

10.25.2. Self-selection bias

Early adopters differ from non-adopters. Comparing them directly confounds treatment with adoption propensity.

10.25.3. Underpowered rare outcomes

A short test cannot establish that severe defects are unchanged when only a handful occur.

10.25.4. Metric substitution

Teams replace an important but difficult outcome with an easy proxy, then optimize the proxy.

10.25.5. Peeking and optional stopping

Repeatedly checking significance and stopping on a favorable day inflates false positives.

10.25.6. Ignoring noncompliance

Assignment without actual exposure dilutes treatment and obscures operational failures.

10.25.7. Global averages

A positive average can hide harm to high-risk or specialized populations.

10.25.8. Unrecorded concurrent interventions

Multiple tools and policies change simultaneously, making attribution impossible.

10.26. Worked example: did the AI reviewer actually reduce review latency?



Runnable versions of this example live in the companion notebooks: * `companion/notebooks/04-causal-did-cace.py` * `companion/notebooks/05-power-and-mde.py` * `companion/notebooks/12-sequential-testing.py`

An organization rolls out an AI reviewer to half of its 240 backend teams over a two-week window. Six weeks later, mean review latency in treated teams is 4.9 hours; in control teams it is 6.8 hours. A leader concludes the AI saved 1.9 hours per review.

Two adjustments change the interpretation.

First, pre-treatment baselines differ: treated teams averaged 5.6 hours in the six weeks before rollout; controls averaged 6.4 hours. A difference-in-differences estimate is:

$$\hat{\tau}_{DiD} = (4.9 - 5.6) - (6.8 - 6.4) = -1.1 \text{ hours}$$

The estimated effect falls from -1.9 to -1.1 hours once we allow for the fact that early-adopting teams were already faster.

Second, exposure is imperfect. Only 71% of assigned reviewers actually opened at least one AI finding in the study window. An intention-to-treat estimate treats the assignment as the treatment; a complier-average estimate rescales:

$$\hat{\tau}_{CACE} \approx \frac{-1.1}{0.71} \approx -1.55 \text{ hours}$$

The two estimands answer different questions: -1.1 hours is what a policy that *offers* the tool produces, and -1.55 hours approximates what the tool produces *among those who use it*. Neither is inherently correct; the decision at hand —

mandating the tool versus opening it — determines which one belongs in the executive summary. Reporting only the raw before-after gap of -1.9 hours would have overstated the effect and mispriced the rollout.

10.27. Key takeaways

- Causal evaluation asks what would have happened without the intervention.
- Treatment, assignment unit, analysis unit, exposure, and outcomes must be defined precisely.
- Randomization is preferred when feasible; quasi-experimental designs are valuable when it is not.
- Clustered engineering systems require clustered designs and standard errors.
- Primary outcomes, practical effect thresholds, and guardrails should be declared in advance.
- External factors, noncompliance, interference, and heterogeneous effects are central—not peripheral—to engineering experiments.
- Experimentation requires durable platform infrastructure and a versioned registry.

10.28. Open questions

- Which assignment units best balance statistical efficiency and contamination in shared repositories?
- How can severe quality outcomes be evaluated without impractically long experiments?
- What designs best estimate long-term skill and trust effects of AI assistance?
- How should network spillovers across reviewers and shared libraries be modeled?
- When should experimentation stop because ethical, safety, or regulatory constraints dominate?

10.29. Implementation checklist

1. Create a mandatory experiment registry for material workflow interventions.
2. Define assignment, analysis, exposure, primary outcomes, and guardrails before launch.
3. Use randomization where practical and document why when it is not.
4. Calculate power using clustering, baseline variance, and expected compliance.
5. Record external events and concurrent interventions.
6. Instrument actual treatment exposure and protocol deviations.
7. Use valid sequential-testing methods for continuous monitoring.
8. Report practical significance, uncertainty, and heterogeneous effects.
9. Preserve analysis code, metric versions, and population snapshots.
10. Record the final decision and whether rollout matched the evidence.

10.29.1. Research foundations and source notes

The experimentation framework in this chapter draws on the potential-outcomes tradition, structural causal models, randomized experiments, quasi-experimental design, preregistration, and practical online experimentation. It extends those foundations into an Engineering Intelligence study system connected to Engineering Changes, tool exposure, guardrails, and economic outcomes.

Potential outcomes and the causal question

The potential-outcomes framework defines causal effects by comparing outcomes that would occur under different treatment conditions for the same unit. See [rubin1974], [holland1986], [imbens2015], and [hernan2020].

For an Engineering Change, team, repository, or service i, let:

$$Y_i(1)$$

represent the outcome under treatment and:

$$Y_i(0)$$

represent the outcome under the comparator.

The individual causal effect is:

$$\tau_i = Y_i(1) - Y_i(0)$$

Only one potential outcome can normally be observed for a given unit at a given time. This is the fundamental problem of causal inference described in [holland1986].

The purpose of experimental and quasi-experimental design is to construct a credible estimate of the missing counterfactual.

Estimands before estimators

A study should define the causal quantity of interest before selecting a statistical method.

Relevant estimands include:

- average treatment effect;
- average treatment effect on treated units;
- intention-to-treat effect;
- treatment-on-the-treated effect;
- complier average causal effect;
- noninferiority margin;
- effect within a declared Engineering Change class;
- effect on the median, tail, or probability of a harmful outcome.

The distinction among assignment, exposure, adoption, and treatment compliance determines which estimands are identifiable.

This ordering follows the design-first perspective in [imbens2015] and [hernan2020].

Randomization

Random assignment provides a principled way to make treatment groups comparable in expectation.

Randomization does not guarantee that every observed sample will be balanced, nor does it solve problems of missing outcomes, noncompliance, spillover, measurement error, or treatment changes.

The chapter therefore requires:

- deterministic and auditable assignment;
- a declared assignment unit;
- balance and sample-ratio checks;
- treatment-version recording;
- outcome and exposure quality monitoring;
- analysis consistent with the assignment design.

These requirements are consistent with randomized-experiment foundations in [rubin1974], [shadish2002], [imbens2015], and practical experimentation guidance in [kohavi2020].

Assignment unit and analysis unit

The randomization unit may differ from the analysis unit.

Examples include:

- assigning repositories while analyzing Engineering Changes;
- assigning teams while analyzing individual deployments;
- assigning time windows while analyzing review events;
- assigning reviewers while analyzing review findings;
- assigning pull requests while analyzing downstream incidents.

When assignment is clustered, standard errors and power calculations must account for within-cluster dependence. Treating correlated observations as independent can materially overstate precision.

Treatment exposure

Assignment is not equivalent to treatment exposure.

A repository may be assigned an AI review tool while the tool fails to run, does not support the language, produces no visible findings, or is bypassed by the developer. A developer may have access to a coding assistant without using it for the observed task.

The chapter therefore requires observed exposure evidence such as:

- treatment invocation;
- version and configuration;
- successful completion;
- visible output;
- interaction or disposition;
- duration and dosage;
- fallback or crossover.

Intention-to-treat analysis should generally preserve original assignment. Exposure-based analyzes may be useful but can reintroduce selection bias unless supported by an appropriate identification strategy.

Causal graphs

Structural causal models and directed acyclic graphs provide a language for representing causal assumptions, confounding, mediation, selection, and collider bias. See [pearl2009].

A causal graph is not evidence by itself. It is a formal statement of assumptions that helps determine:

- which variables should be controlled;
- which variables should not be controlled;
- what mechanisms are hypothesized;
- where selection can bias the estimate;
- which outcomes and mediators require separate analysis.

For example, controlling for a post-treatment variable such as tool usage or review comments can distort an intention-to-treat estimate when that variable is itself affected by treatment.

Confounding

A confounder influences both treatment selection and outcome.

In engineering organizations, common confounders include:

- team maturity;
- repository complexity;
- work mix;
- developer experience;
- release season;
- organizational change;
- incident load;
- language and framework;
- voluntary tool adoption;
- leadership sponsorship.

Randomization addresses measured and unmeasured pre-treatment confounding in expectation. Observational studies must rely on stronger assumptions and design strategies.

Interference and spillover

The stable-unit-treatment-value assumption can fail in software organizations because one unit's treatment can affect another unit's outcome.

Examples include:

- treated and untreated developers sharing code or reviewers;
- AI-generated patterns propagating across repositories;
- one team's merge queue changing another team's dependency timing;
- a treated reviewer influencing untreated pull requests;
- shared CI capacity creating cross-group queue effects.

The chapter therefore requires explicit interference analysis and may use cluster randomization, network-aware assignment, geographic or organizational separation, saturation designs, or spillover estimands.

These concerns are part of the design and validity traditions discussed in [shadish2002], [imbens2015], and [hernan2020].

Noncompliance and crossover

Units may not receive their assigned treatment, and control units may gain access to treatment.

The study should distinguish:

- assignment failure;
- treatment unavailability;
- user non-use;
- technical incompatibility;
- contamination;
- deliberate crossover;
- treatment interruption.

Intention-to-treat remains a primary policy estimand because it measures the effect of assigning the intervention under realistic compliance. Instrumental-variable methods may estimate effects among compliers under additional assumptions. See [angrist2009] and [imbens2015].

Quasi-experimental designs

Randomization is not always feasible. The chapter therefore includes several quasi-experimental methods.

Difference-in-differences

Difference-in-differences compares changes over time between treated and comparison groups.

Its credibility depends on assumptions such as parallel untreated trends, no treatment anticipation, stable composition, and appropriate handling of staggered timing.

The method is discussed in [angrist2009] and broader causal-inference references including [hernan2020].

Regression discontinuity

Regression discontinuity uses a treatment threshold in an assignment variable.

It can provide strong local causal evidence when the threshold is enforced, the running variable cannot be manipulated precisely, and outcomes are smooth around the cutoff absent treatment. See [angrist2009].

Interrupted time series

Interrupted time series examines level or trend changes after a clearly timed intervention.

It requires careful modeling of seasonality, autocorrelation, concurrent interventions, and preexisting trends. A single before-and-after comparison is not an interrupted-time-series design.

Matched controls and weighting

Matching and weighting attempt to create comparable treated and control populations based on measured pre-treatment covariates.

They cannot remove bias from unmeasured confounding and should be accompanied by overlap diagnostics, balance checks, sensitivity analysis, and explicit causal assumptions.

Synthetic controls

Synthetic-control methods construct a weighted comparison from untreated populations to approximate the treated unit's pre-intervention trajectory.

They are most useful when a small number of aggregate units receive treatment and a credible donor pool exists.

Switchback and time-based designs

Switchback experiments alternate treatment across time windows and are useful when infrastructure or policies cannot be assigned permanently at individual-unit level.

They require:

- appropriate washout periods;
- control of time-of-day and day-of-week effects;
- stable treatment implementation;
- analysis at the randomization-window level;
- protection against carryover and anticipation.

Shared CI capacity, routing policy, or reviewer-assistance treatments may be suitable candidates.

Stepped-wedge rollout

In a stepped-wedge design, clusters transition from control to treatment over scheduled periods.

This can support operationally necessary phased rollout, but treatment effects can be confounded with calendar time unless the design and analysis account for period effects.

The chapter treats stepped-wedge rollout as a study design, not as automatic evidence merely because deployment was staggered.

Outcomes, mechanisms, and guardrails

A strong study separates:

- primary outcomes;
- guardrail outcomes;
- mechanism measures;
- adoption and exposure;
- data-quality measures;

- economic realization measures.

This prevents a mechanism such as increased tool usage or reduced authoring time from being mistaken for the final outcome.

Practical online-experiment guidance in [kohavi2020] supports the use of primary metrics, guardrails, data-quality checks, and trustworthy experimentation infrastructure.

Power and minimum detectable effect

Power analysis links sample size, variability, clustering, significance threshold, and the smallest effect the study is designed to detect.

A study that cannot detect a decision-relevant effect should not interpret a nonsignificant result as evidence of no effect.

Engineering studies often face:

- clustered assignment;
- highly skewed outcomes;
- rare defects or incidents;
- delayed outcome maturity;
- heterogeneous work classes;
- changing treatment versions.

Simulation-based power analysis is often more credible than a simple independent-sample formula.

Noninferiority and equivalence

Some engineering decisions ask whether a cheaper, faster, or more autonomous process is not unacceptably worse than the current process.

Noninferiority designs require a predeclared margin representing the largest acceptable degradation. The margin should be justified by operational, safety, or economic consequence rather than chosen after observing results.

Equivalence testing asks whether effects are sufficiently close within both positive and negative bounds.

These designs are more appropriate than failure to reject a superiority null when the decision is about acceptable similarity.

Multiple outcomes and repeated analysis

Testing many metrics, subgroups, or time windows increases the chance of false-positive findings.

The chapter therefore distinguishes:

- one or a small number of confirmatory primary outcomes;
- predeclared guardrails;
- exploratory secondary analysis;
- correction or hierarchical methods where appropriate;
- transparent reporting of all examined outcomes.

Repeatedly inspecting results and stopping when significance appears also changes error properties. Sequential designs require declared monitoring and stopping methods.

Practical concerns around trustworthy online experimentation are covered in [kohavi2020].

Preregistration and pre-analysis plans

Preregistration separates confirmatory analysis from analyzes chosen after observing results. See [nosek2018].

The Engineering Intelligence pre-analysis plan records:

- causal question and estimand;
- eligibility;
- assignment and exposure definitions;
- outcome and guardrail versions;
- exclusion rules;
- covariates;
- missing-data handling;
- statistical model;
- subgroup analysis;
- stopping rules;
- decision thresholds.

Preregistration does not prohibit exploration. It makes the distinction between confirmatory and exploratory work visible.

Missing outcomes and attrition

Outcome data may be missing because work is abandoned, deployment is delayed, telemetry fails, or the observation window closes before incidents mature.

The study should report missingness by treatment group and reason. Complete-case analysis can be biased when missingness depends on treatment or outcome.

Approaches may include:

- inverse-probability weighting;
- multiple imputation;
- bounds;
- sensitivity analysis;
- time-to-event methods;
- explicit immature-outcome states.

No method compensates for unexamined missingness assumptions.

Heterogeneous treatment effects

Average effects can conceal important differences across:

- language;
- repository size;
- change complexity;
- developer experience;
- team workflow;
- risk class;
- model or prompt version;
- type of AI contribution;
- baseline performance.

Subgroup analysis should be motivated, adequately powered, and protected against overinterpretation. Exploratory heterogeneity can generate hypotheses for replication rather than immediate policy.

External validity and transportability

A causal estimate is local to its study population, intervention version, comparator, period, and operating context.

Results from one organization, task, repository class, or AI model may not transport to another context.

The chapter therefore records boundary conditions and promotes replication across teams and time. This follows the internal- and external-validity tradition in [shadish2002].

Reproducibility

A study should preserve enough information to reproduce the analytical result:

- source snapshots or durable references;
- assignment table;
- treatment and exposure versions;
- metric definitions;
- transformation lineage;
- analysis code;
- environment and dependency versions;
- exclusions and corrections;
- generated tables and figures.

The broader importance and difficulty of reproducibility is illustrated by [open-science2015], while FAIR stewardship principles are described in [wilkinson2016].

Decision rules and post-study follow-up

A statistical result does not make the organizational decision automatically.

The decision should integrate:

- estimated effect and uncertainty;
- guardrails;

- operational feasibility;
- cost and value;
- fairness and governance;
- implementation risk;
- reversibility;
- evidence maturity.

The chapter therefore links the study result to a separate decision record and value-realization plan.

Long-term follow-up is required because novelty effects, adoption, skill development, maintenance burden, and downstream quality can change after the experimental window.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- Engineering Change as an assignment, exposure, and outcome unit;
- explicit machine-treatment provenance including model, prompt, retrieval, and tool versions;
- integration of assignment, actual exposure, human disposition, and downstream consequence;
- experiment readiness reviews tied to data-quality and governance floors;
- economic and attention outcomes within the causal protocol;
- treatment-version drift as a first-class validity threat;
- value-realization follow-up after causal evaluation;
- decision records linked to immutable study evidence;
- study registries spanning randomized, quasi-experimental, and observational evidence;
- replication across Engineering Change classes as an organizational learning mechanism.

These extensions should be evaluated through repeated studies, independent review, and transparent reporting of null, negative, and contradictory results.

Chapter 11. AI-Assisted Engineering



AI systems now author code, review changes, generate tests, explain failures, summarize *discussions, and increasingly execute multi-step engineering tasks. Treating them merely as software licenses obscures their effect on engineering work. This chapter models AI as a first-class participant in the Engineering Change. It defines contribution, attribution, validation, trust, quality, and economic measures; distinguishes assistance from autonomy; explains how AI can create both cognitive offload and cognitive reload; and provides evaluation patterns for coding assistants, reviewers, and agents.*

11.1. From tool to participant

A compiler transforms source code deterministically. An AI reviewer makes judgments, produces ranked findings, and influences human decisions. An agent may inspect repositories, modify files, run tests, and open a pull request. These systems participate in the engineering process in a way that is qualitatively different from passive tooling.

Engineering Intelligence represents every contribution with an actor type:

- human;
- AI model;
- agent or orchestration system;
- deterministic automation;

- hybrid human-AI action.

The distinction supports attribution without implying that AI and humans are equivalent. Humans remain accountable for organizational decisions, while AI contributions are measured as evidence and work products.

11.2. Levels of AI participation

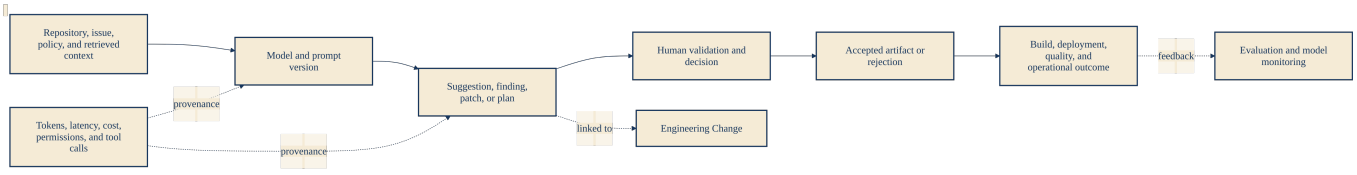
Level	Description
Assistive	Suggests text, code, explanations, or tests while a human controls each action.
Advisory	Reviews work, identifies risks, ranks findings, or recommends decisions.
Delegated	Executes a bounded task under explicit constraints, such as updating dependencies or generating tests.
Agentic	Plans and executes multiple steps, uses tools, revises work, and reports outcomes.
Autonomous	Operates with minimal direct supervision within a governed scope and escalation policy.

Evaluation requirements increase with autonomy. A code completion can be judged at suggestion level; an autonomous migration agent requires task, system, and operational evaluation.

11.3. AI contribution records

Every AI contribution should retain:

- Engineering Change and artifact identifiers;
- actor, model, provider, and version;
- prompt, policy, context-selection, and tool versions;
- input scope and data sources;
- output artifact or finding;
- confidence or ranking score;
- latency, tokens, tool calls, and cost;
- human actions and validation;
- downstream revisions and outcomes;
- governance and permission context.



Without provenance, organizations cannot explain why behavior changed after a model, prompt, retrieval index, or policy update.

11.4. Attribution is a graph problem

AI-generated work is rarely accepted unchanged. A model proposes code, a human edits it, another model reviews it, and a reviewer requests further modification. Simple authorship percentages are misleading.

Represent contribution as a derivation graph with edges such as `GENERATED`, `EDITED`, `VERIFIED`, `REJECTED`, `REPLACED`, `INSPIRED`, and `DUPLICATED`.

A useful distinction is:

- **production attribution** — who or what produced artifact content;
- **decision attribution** — who or what influenced a decision;
- **accountability** — which human role accepted responsibility;
- **causal contribution** — whether the AI changed the final outcome relative to the counterfactual.

Only the last directly supports value claims, and it normally requires experimentation.

11.5. Evaluating AI code generation

Suggestion acceptance is a useful interaction measure but not a complete outcome. Evaluation should include:

- acceptance with and without modification;
- time to usable result;
- compilation and test success;
- defect and security outcomes;
- maintainability and architectural fit;
- duplicated or unnecessary code;
- review burden;
- downstream rework;
- author understanding;
- task completion and abandonment.

At the task level, compare quality-adjusted human effort and accepted outcome, not merely generated lines.

11.6. Evaluating AI review

An AI review finding is an information-retrieval and decision-support object. For labeled findings, use precision and recall:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Severity-weighted versions are often more meaningful:

$$Recall_w = \frac{\sum_{i \in TP} w_i}{\sum_{i \in TP \cup FN} w_i}$$

Ranking quality matters because human attention is limited. NDCG, precision at k, and severe-finding recall at k measure whether the most important issues appear first.

Operational measures should also include duplication, action type, validation effort, comment latency, suppression quality, and downstream defect prevention.

11.7. Ground truth and golden data

AI review evaluation requires a labeled corpus containing Engineering Changes, known defects and non-defects, severity, category, relevant code context, expected evidence, and acceptable alternative findings.

Golden datasets should include:

- real historical defects;
- injected or mutation-based defects;
- clean changes to estimate false positives;
- security, concurrency, data, UI, API, and operational categories;
- multiple languages and repository types;
- ambiguous cases adjudicated by experts.

A fixed golden set risks overfitting. Maintain hidden holdouts, refresh samples, and evaluate on live outcomes.

11.8. Human validation burden

AI output must be interpreted and verified. Define validation burden:

$$VB(e) = T_{inspect} + T_{investigate} + T_{correct} + T_{recover}$$

High output volume can increase VB even when inference cost is low.

Useful proxies include finding views, code navigation after a finding, time to action, independent reproduction, comment discussion, suggestion modification, and rejection reason.

11.9. Trust calibration

The goal is calibrated trust, not maximum trust. Users should rely on the system when it is likely correct and scrutinize it when uncertainty is high.

Over-trust creates automation bias. Under-trust destroys offload value. Trust calibration depends on precision, consistency, explanation quality, confidence honesty, domain fit, and recovery from errors.

Measure:

- reliance conditional on actual correctness;
- rejection of incorrect findings;
- acceptance of correct high-value findings;
- changes in behavior after visible errors;
- calibration by category and confidence band.

11.10. Cognitive offload and reload

AI creates offload when it reduces human reasoning, search, production, or verification while preserving outcomes. It creates reload when humans must inspect noisy output, reconstruct hidden assumptions, correct generated artifacts, or recover from misplaced trust.

Net cognitive effect:

$$NCE = W_{offload} - W_{reload}$$

The effect should be estimated experimentally and stratified by task type, expertise, and risk.

11.11. Quality-adjusted AI leverage

A proposed AI Leverage Factor is:

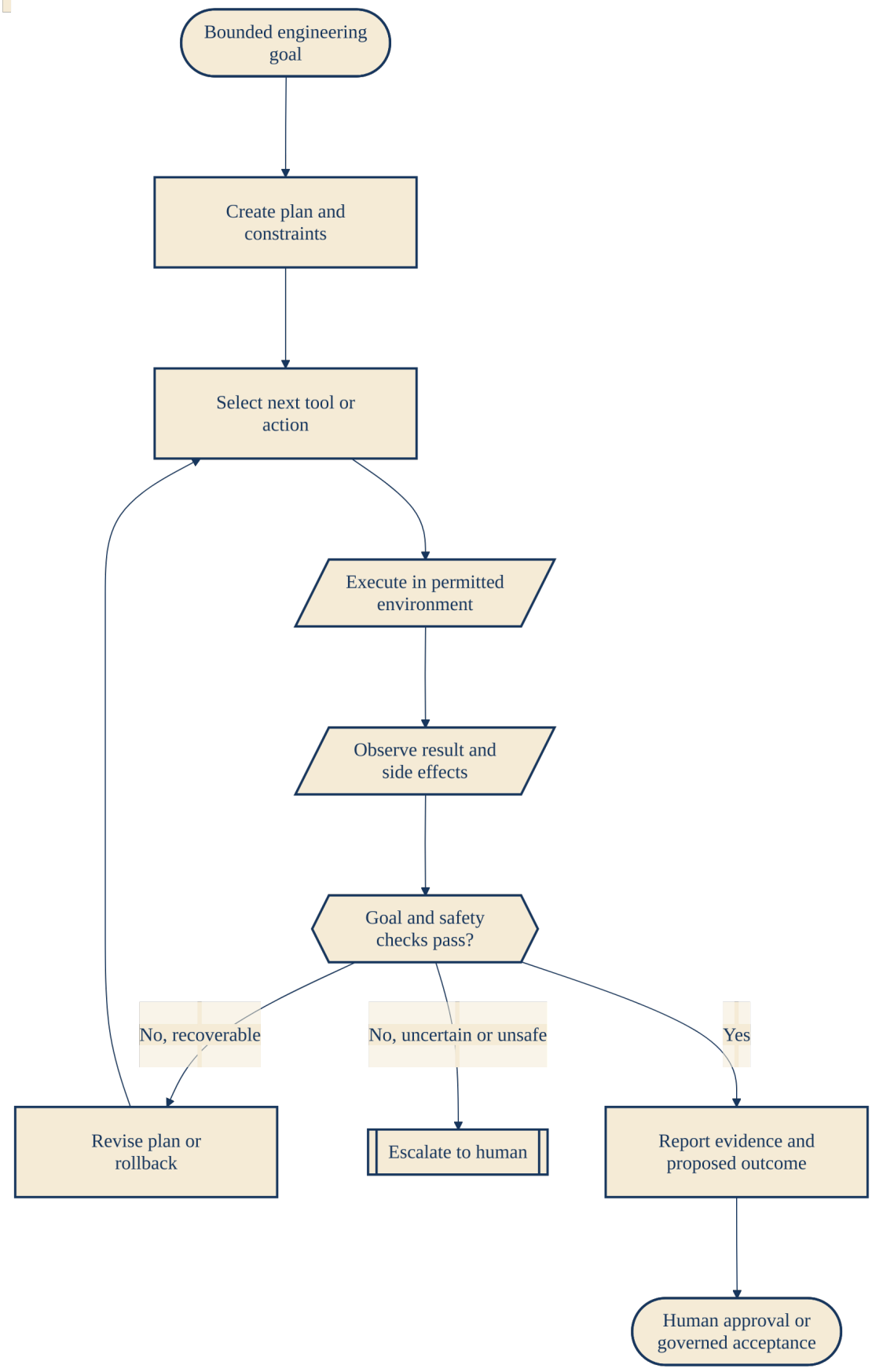
$$ALF = \frac{\Delta V + \Delta C_{human} + \Delta C_{quality}}{C_{AI} + C_{validation}}$$

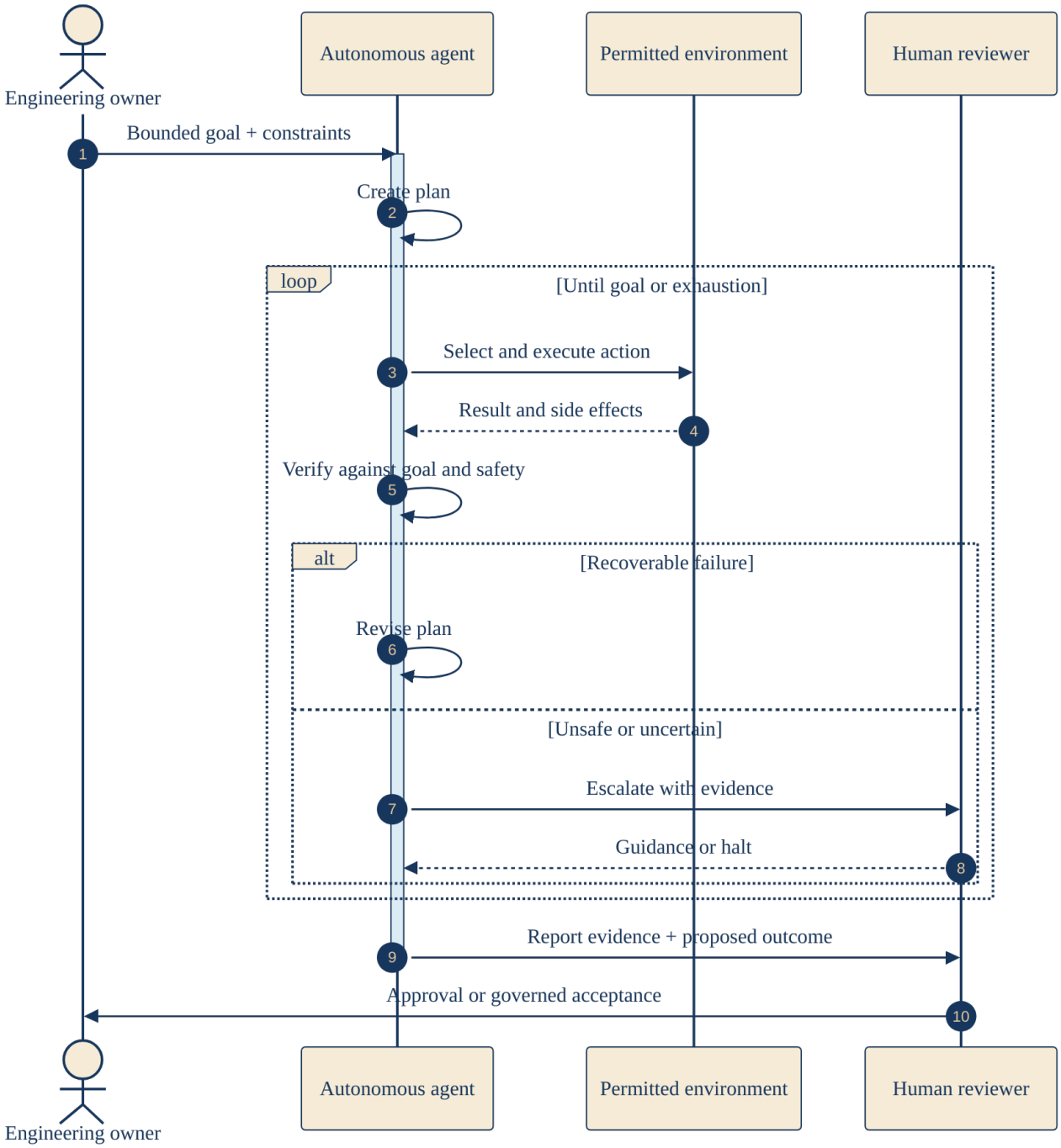
where positive numerator terms represent additional value, human cost avoided, and expected quality benefit. The formula is a decision model, not a universal score. Assumptions and uncertainty must be visible.

11.12. Agents and multi-step tasks

Agent evaluation requires additional dimensions:

- task planning quality;
- tool selection and permission use;
- state tracking;
- recovery from failed steps;
- test and verification discipline;
- boundedness and escalation;
- reproducibility;
- side effects;
- final task success.


Bounded engineering goal
Create plan and constraints
Select next tool or action
Execute in permitted environment
Observe result and side effects
Goal and safety checks pass?
No, recoverable
No, uncertain or unsafe
Yes
Revise plan or rollback
Escalate to human
Report evidence and proposed outcome
Human approval or governed acceptance



The flowchart above collapses the loop into its logical states; the sequence diagram makes the addressees explicit. Every arrow crosses a trust boundary — from a human intent to an autonomous plan, from the plan to a permitted execution surface, and back to a human review of the resulting evidence — and each boundary is a place where instrumentation, permissions, and audit trails must be defined.

A successful pull request is not sufficient if the agent used unsafe permissions, modified unrelated files, or produced hidden operational risk.

11.13. Autonomy and risk tiers

Recommended tiers include:

Tier	Controls
Low risk	Advisory output, easy reversal, non-production artifacts, standard validation.
Moderate risk	Bounded writes, required tests, human approval, audit trail, rollback.
High risk	Sensitive data, security, financial logic, infrastructure, or broad blast radius; specialist review and constrained permissions.
Prohibited or exceptional	Actions exceeding policy, legal, safety, or accountability boundaries.

Risk routing should consider both task risk and model uncertainty.

11.14. Model, prompt, and policy drift

AI behavior can change because of model upgrades, prompt edits, retrieval changes, repository evolution, or policy configuration. Treat each as a versioned intervention.

Production monitoring should detect changes in finding volume, precision proxies, action rates, cost, latency, category mix, failure modes, and guardrail outcomes.

A vendor model update without an experiment record can invalidate historical comparisons.

11.15. Human learning and skill

AI can accelerate learning by explaining code, demonstrating patterns, and exposing defects. It can also reduce learning when users accept outputs without understanding.

Evaluation should include long-term outcomes such as independent task performance, reviewer capability, onboarding speed, knowledge concentration, and ability to operate when AI is unavailable.

Learning is not captured by immediate cycle time.

11.16. Governance and security

AI engineering systems require controls for:

- data access and retention;
- source-code and secret exposure;
- identity and authorization;
- tool-call permissions;
- model and vendor approval;
- auditability;
- generated-code licensing and provenance;

- prompt injection and untrusted repository content;
- escalation and shutdown;
- accountability for accepted changes.

Least privilege should apply to agents. Read, write, execute, deploy, and production-access capabilities should be independently controlled.

11.17. Experiment patterns

Useful experiments include:

- completion versus no completion for bounded tasks;
- ranked review findings versus full unranked output;
- AI summary versus raw change after long delays;
- one model versus another on a shared golden and live evaluation;
- human-first versus AI-first review order;
- autonomous patch generation versus advisory findings;
- different confidence thresholds and suppression policies;
- agent autonomy levels with equal task populations.

Primary outcomes should be quality-adjusted effort or accepted task outcome. Guardrails should include severe defects, security, validation burden, trust, and learning.

11.18. Common failure modes

11.18.1. Counting generated code as productivity

Generated volume can increase review, maintenance, and defect cost.

11.18.2. Treating acceptance as correctness

Users can accept incorrect output or reject valuable findings.

11.18.3. Ignoring human validation

Inference cost is often small relative to attention spent validating output.

11.18.4. Evaluating only synthetic benchmarks

Benchmarks provide control but may not reflect repository context, organizational conventions, or live behavior.

11.18.5. Global average performance

AI quality varies substantially by language, domain, complexity, and risk.

11.18.6. Silent model changes

Unversioned updates make experiments and trends irreproducible.

11.18.7. Automating beyond evidence

Strong advisory performance does not automatically justify autonomous execution.

11.19. Worked example: measuring net cognitive offload from an AI reviewer



Runnable version: `companion/notebooks/14-cognitive-offload-index.py`.

A twelve-week study evaluates whether an AI code reviewer delivers net cognitive offload on 12,400 pull requests across a treated and matched control cohort.

Baseline human review effort in the control cohort averages 17.2 minutes per pull request. In the treatment cohort, direct review time falls to 13.9 minutes — an apparent 3.3 minute per-review saving.

But the treatment cohort's authors spend an additional 1.4 minutes per pull request triaging and dispositioning AI findings, and reviewers spend 1.1 minutes verifying findings they did not surface themselves. The treatment cohort also shows a 6% increase in reopened-comment cycles when AI findings prompt rework the reviewer would not have requested.

Net attention accounting:

- direct review time saved: -3.3 min
- author verification of AI findings: +1.4 min
- reviewer verification: +1.1 min
- additional rework cycle effort (probability-weighted): +0.4 min

Net saving: -0.4 minutes per pull request. The AI reviewer produces a real but modest 2.3% net reduction in human cognitive work, not the 19% headline reduction the direct-time comparison suggested.

The Quality-adjusted Cognitive Offload Index applies the guardrail: escaped-defect rate falls by 8% in the treatment cohort (statistically significant). The quality-retention factor $Q_r = 1.05$ raises QCOI slightly, making the intervention marginally net-positive on a quality-adjusted basis. The decision record notes that the case for the tool depends on the quality effect, not the raw time saving, and re-evaluation is scheduled after the next model upgrade.

11.20. Key takeaways

- AI should be modeled as a first-class participant with explicit provenance, cost, and outcomes.
- Attribution must distinguish production, influence, accountability, and causal contribution.
- AI review should be evaluated using precision, recall, ranking, validation burden, and downstream quality.
- Acceptance rate is an interaction measure, not a value measure.
- Net value depends on cognitive offload minus cognitive reload.
- Agent evaluation must include planning, permissions, side effects, recovery, and boundedness.
- Model, prompt, retrieval, and policy versions are part of the treatment definition.
- Long-term trust, learning, and skill effects require measurement.

11.21. Open questions

- How can causal contribution be attributed across iterative human-AI editing?
- What live evaluation designs best detect rare but severe AI-induced failures?
- How should reviewer learning and skill atrophy be valued?
- Which confidence displays produce the best trust calibration?
- How should autonomy be increased safely as evidence accumulates?

11.22. Implementation checklist

1. Define actor types and versioned AI contribution records.
2. Capture model, prompt, context, tool, policy, latency, and cost metadata.
3. Build labeled golden sets with clean and defective changes.
4. Measure precision, recall, ranking, duplication, action, and validation effort.
5. Link AI contributions to downstream code, defects, incidents, and accepted outcomes.
6. Evaluate cognitive offload and reload experimentally.
7. Establish autonomy tiers and least-privilege permissions.
8. Monitor model and prompt drift as versioned interventions.
9. Add trust, learning, security, and severe-quality guardrails.
10. Preserve human accountability for accepted engineering outcomes.

11.22.1. Research foundations and source notes

The AI-assisted engineering framework in this chapter draws on empirical studies of coding assistants, human–AI interaction, software-agent benchmarks, information retrieval, causal evaluation, and current AI risk-management standards. It extends those foundations into a production-oriented model centered on Engineering Changes, human validation, provenance, economics, and long-term capability effects.

Task-level productivity evidence

Controlled studies have reported that AI coding assistants can reduce completion time for some bounded programming tasks. See [peng2023].

These findings are useful but should be interpreted narrowly. A task-level time reduction does not automatically imply:

- lower end-to-end Engineering Change cost;
- lower review or validation effort;
- improved maintainability;
- fewer production defects;
- increased portfolio throughput;
- realized financial savings;
- durable benefit after novelty and learning effects.

The chapter therefore treats task completion time as one possible mechanism measure rather than as a universal productivity outcome.

Usability, comprehension, and debugging

Research on AI code-generation tools has identified challenges involving expectation mismatch, verification, debugging, comprehension, and trust. See [vaithilingam2022].

These findings support the chapter's distinction between generation and useful contribution. Code that is produced quickly may still require substantial effort to inspect, understand, integrate, test, secure, or maintain.

The relevant work system includes both the machine's production and the human's response.

Interaction patterns

Studies of programr interaction with code-generating models describe iterative behaviors such as acceleration, exploration, steering, repair, and evaluation. See [barke2023].

This supports a richer event model than simple accepted-versus-rejected suggestions.

Useful interaction states include:

- requested;
- generated;
- displayed;
- inspected;
- accepted unchanged;
- accepted with modification;
- rejected;
- ignored;
- retried;
- superseded;
- reverted;
- independently reimplemented;
- associated with downstream rework or defect evidence.

Repository-level agent benchmarks

Repository-level benchmarks such as SWE-bench evaluate whether software agents can resolve real issue-derived tasks in existing repositories. See [jimenez2024].

These benchmarks provide important capability evidence because they require navigation, localization, editing, and test interaction across realistic codebases.

They still differ from enterprise production in several ways:

- the task and acceptance criterion are predefined;
- organizational coordination may be absent;
- production deployment and operational consequence are usually out of scope;
- security, privacy, and policy constraints differ;
- repository and tool access may be simplified;

- long-term maintainability is rarely observed;
- treatment cost and human oversight may not match production practice.

The chapter therefore treats benchmark performance as one evidence layer rather than as proof of autonomous production readiness.

Acceptance is not value

Suggestion acceptance is an adoption or interaction measure.

It does not establish that the accepted contribution:

- was correct;
- was novel;
- reduced net human effort;
- improved code quality;
- prevented a defect;
- reduced delivery latency;
- remained in the final artifact;
- improved downstream outcomes.

This distinction follows the broader measurement-validity principle in [messick1995] and the indicator-gaming concerns in [campbell1979] and [goodhart1984].

The chapter therefore separates acceptance, contribution, quality, attention, and outcome measures.

Precision, recall, and ranked findings

AI review systems often produce findings or recommendations rather than code.

Precision and recall provide standard measures for evaluating finding relevance and coverage. See [manning2008].

Ranking quality can be measured with discounted cumulative gain or normalized discounted cumulative gain where severity and position matter. See [jarvelin2002].

Production evaluation should also account for:

- duplicate findings;
- equivalent findings expressed differently;
- severity calibration;
- false-positive handling time;
- findings already known to the author;
- findings that prompt useful inspection without direct acceptance;
- defects prevented downstream;
- unsupported or unverifiable claims.

Human validation burden

AI output transfers part of the work from production to verification.

The chapter defines validation burden as the human attention required to determine whether machine output is correct, appropriate, safe, maintainable, and consistent with intent.

This burden can include:

- reading generated code or analysis;
- checking assumptions;
- reproducing claims;
- running tests;
- consulting domain experts;
- correcting subtle errors;
- evaluating security and privacy consequences;
- documenting why output was accepted.

Cognitive-load and interruption research in [sweller1988] and [mark2008] provides part of the theoretical basis for treating this work as economically meaningful.

Cognitive offload and cognitive reload

The chapter defines paired concepts:

- **cognitive offload** — human reasoning or production work genuinely avoided because a reliable machine contribution performed it;
- **cognitive reload** — additional human reasoning created by noisy, ambiguous, incomplete, or untrustworthy machine output.

Net attention change can be expressed as:

$$NetAttentionChange = AuthoringAttentionReduced - ValidationAttentionAdded - ReworkAttentionAdded$$

This is an Engineering Intelligence-specific operational model. It should be calibrated with task observation, event evidence, and practitioner studies rather than assumed from output volume.

Provenance and reproducibility

An AI contribution cannot be interpreted correctly without treatment provenance.

The chapter therefore requires capture of:

- provider;
- model and model version;
- prompt or instruction version;
- retrieved context;
- tool permissions and calls;
- temperature and relevant configuration;
- initiating actor;

- delegation chain;
- timestamps and latency;
- input and output size;
- cost;
- policy and guardrail version.

This is compatible with the entity, activity, and agent concepts in W3C PROV. See [moreau2013].

Because models and hosted services can change, reproducibility may require retained outputs, evaluation fixtures, configuration snapshots, and vendor version attestations even when exact model weights are unavailable.

Causal evaluation

Voluntary adoption creates strong selection bias. Early users may differ in skill, enthusiasm, task mix, or management support.

The chapter therefore relies on the causal methods described in Chapter 6 and grounded in [rubin1974], [holland1986], [shadish2002], [imbens2015], [hernan2020], and [kohavi2020].

A credible AI evaluation distinguishes:

- assignment;
- access;
- actual invocation;
- successful exposure;
- interaction and disposition;
- downstream contribution;
- mature outcomes.

Randomized, clustered, switchback, stepped-wedge, or quasi-experimental designs may be appropriate depending on the intervention.

Treatment-version drift

AI systems change rapidly through model updates, prompt changes, retrieval changes, policy changes, and tool integrations.

A study can become difficult to interpret when the treatment changes during the observation period.

The chapter therefore treats treatment version as a first-class causal variable and requires:

- version-effective intervals;
- exposure linked to exact configuration;
- shadow evaluation before major changes;
- separate analysis when treatment changes materially;
- rollback capability;
- post-update replication.

Quality and security guardrails

Faster completion is not sufficient when quality or safety degrades.

Relevant guardrails include:

- escaped defects;
- security findings;
- privacy violations;
- rollback and incident rates;
- test adequacy;
- maintainability and complexity;
- unsupported dependency introduction;
- license or provenance risk;
- reviewer confidence;
- long-term rework.

AI risk-management principles are supported by the NIST AI Risk Management Framework and its generative-AI profile. See [nist2023-airmf] and [nist2024-genai].

Management-system and risk-process standards are represented by 001 and 894.

These standards guide governance and control; they do not prove that any particular AI system is effective or safe.

Economic evaluation

AI economics should include more than licensing or token cost.

The chapter's cost model includes:

- subscription and platform cost;
- inference, retrieval, storage, and indexing;
- integration and administration;
- security and governance;
- human validation;
- false-positive handling;
- rework and rollback;
- migration and switching;
- vendor concentration and exit risk;
- quality-adjusted attention released.

The broader cost-benefit framework is grounded in [boardman2018], while delayed productivity realization is informed by [brynjolfsson1993] and [brynjolfsson2017].

Released minutes should not be reported as cash savings unless spending is actually avoided or removed.

Work-mix and heterogeneity

AI effects are likely to vary across:

- language and framework;
- repository maturity;
- change complexity;
- developer experience;
- task novelty;
- domain risk;
- available tests and documentation;
- model and prompt version;
- organization-specific tooling.

Average effects can conceal both high-value and harmful subgroups.

The chapter therefore recommends predeclared subgroup analysis, work-mix adjustment, and replication before broad policy.

Skill formation and capability effects

Short-term efficiency may coexist with long-term changes in skill.

Potential outcomes include:

- faster learning through explanation and examples;
- broader access to unfamiliar domains;
- reduced practice in core reasoning;
- weaker debugging or code-reading capability;
- dependence on model availability;
- changed mentoring and apprenticeship patterns;
- convergence toward model-preferred solutions.

The current evidence base is not mature enough to support universal conclusions. The chapter treats skill formation and atrophy as long-horizon research and governance questions requiring longitudinal study.

Automation bias and overreliance

Users may accept machine output too readily when it appears authoritative, fluent, or confident.

The chapter addresses this through:

- uncertainty and evidence display;
- required human accountability for consequential acceptance;
- independent validation for high-risk actions;
- calibrated confidence rather than stylistic certainty;
- logging of overrides and reversals;
- periodic tests of human capability without assistance.

Professional obligations concerning harm, competence, honesty, and accountability are informed by [acm2018-ethics].

Agent authority and delegation

Autonomous agents require a richer governance model than interactive suggestions.

The chapter distinguishes:

- permission to inspect;
- permission to propose;
- permission to modify;
- permission to execute tests;
- permission to merge;
- permission to deploy;
- permission to alter infrastructure or data;
- permission to delegate to other agents.

Each action should carry actor identity, delegated authority, policy version, evidence, validation responsibility, and rollback path.

The delegation-graph and agent-authority model are original Engineering Intelligence extensions compatible with provenance concepts in [moreau2013] and risk-governance principles in [nist2023-airmf] and [nist2024-genai].

Model monoculture and correlated risk

Broad adoption of one model or provider can create correlated failure modes:

- repeated insecure patterns;
- common reasoning blind spots;
- synchronized service disruption;
- shared supply-chain risk;
- systemic licensing or policy issues;
- reduced diversity of solution approaches.

The chapter therefore treats provider concentration, model diversity, fallback, and correlated-error testing as portfolio-level risk concerns.

This is a forward-looking operational extension rather than a settled empirical standard.

Continuous evaluation

One-time vendor evaluation is insufficient because models, prompts, repositories, and workflows change.

A continuous evaluation system should include:

- stable golden datasets;
- representative production samples;
- shadow comparisons;
- quality and safety guardrails;
- drift detection;
- model and prompt versioning;

- cost tracking;
- human disagreement review;
- downstream outcome follow-up.

Benchmark evidence, online experiment evidence, and production evidence should remain distinct but connected.

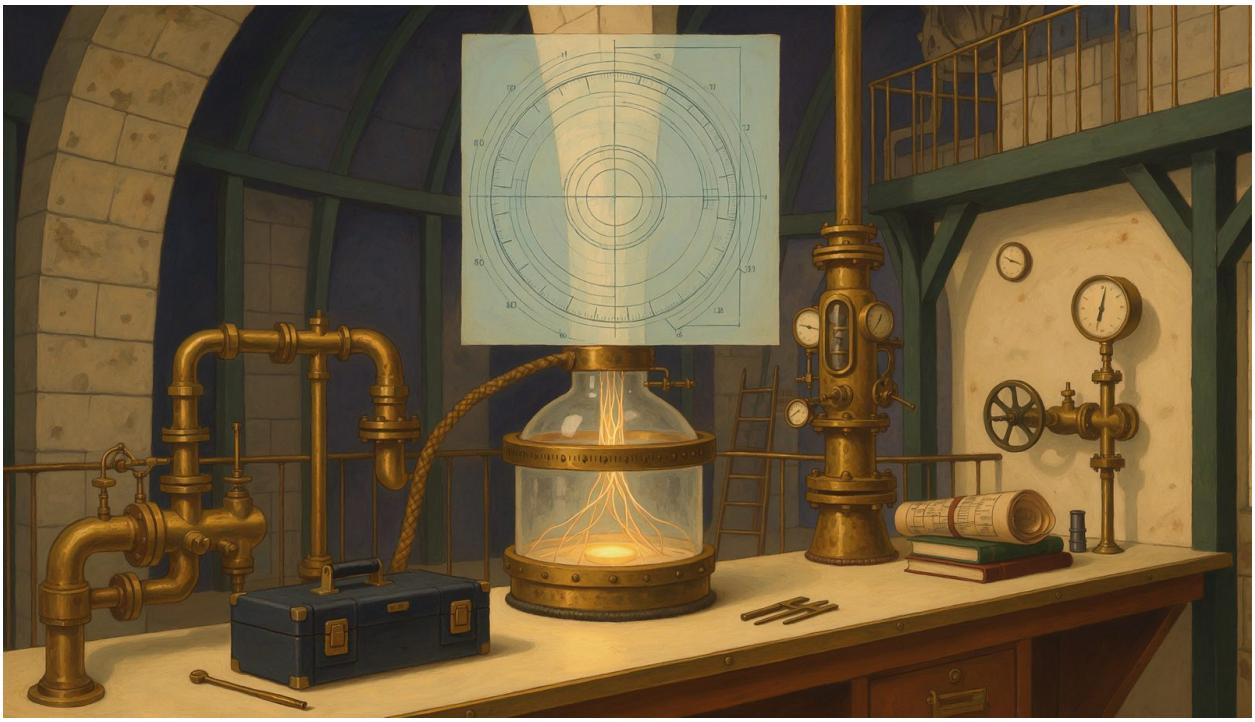
Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- AI as a first-class Engineering Change participant;
- machine-contribution provenance across model, prompt, retrieval, tools, and delegation;
- paired cognitive-offload and cognitive-reload measures;
- net attention economics;
- accepted-contribution share rather than acceptance alone;
- validation burden as a primary outcome;
- treatment-version drift as a causal and operational risk;
- continuous evaluation linked to downstream Engineering Change outcomes;
- agent authority and delegation graphs;
- autonomous-action reversal and rollback measures;
- model-monoculture and correlated-risk analysis;
- long-term skill and capability guardrails.

These constructs should be treated as hypotheses and operating standards to validate, not as settled universal laws.

Chapter 12. Enterprise Instrumentation



***Engineering Intelligence depends on reliable, explainable, and governable telemetry across** source control, planning, CI, deployment, AI, production, and organizational systems. This chapter describes the infrastructure required to capture that data at enterprise scale. It defines event contracts, identity resolution, event-time processing, immutable logs, stream and batch responsibilities, Engineering Change reconstruction, knowledge graphs, feature stores, metric computation, lineage, data quality, privacy, and operational architecture.*

12.1. Instrument the system, not the dashboard

Many engineering analytics programs begin with dashboards and work backward toward data. This often produces brittle queries, inconsistent definitions, and local extracts that cannot support experimentation or historical recomputation.

A stronger approach begins with a measurement architecture:

1. identify decisions and constructs;
2. define canonical entities and events;
3. capture raw evidence immutably;
4. normalize identity and time;
5. reconstruct Engineering Changes;
6. compute versioned features and metrics;

7. expose evidence through decision products.

The dashboard is the final presentation layer, not the foundation.

12.2. Source domains

A complete implementation commonly integrates:

Domain	Representative evidence
Planning	Initiatives, issues, priority, intent, ownership, status, deadlines
Source control	Repositories, branches, commits, pull requests, reviews, comments, approvals
Developer environment	Builds, tests, branch context, agent interactions, explicit work sessions
AI systems	Model, prompt, context, findings, suggestions, actions, cost, latency
Continuous integration	Jobs, steps, tests, scans, retries, queues, artifacts, runner resources
Deployment	Releases, environments, flags, rollout waves, approvals, rollback
Production	Metrics, traces, incidents, alerts, customer and business outcomes
Organization	Teams, roles, ownership, reporting boundaries, expertise, calendars

No source should be treated as universally authoritative. Each system is authoritative for a bounded type of evidence.

12.3. Canonical event envelope

Every source event should be wrapped in a common envelope while preserving the original payload.

```
{
  "eventId": "01J9K3...",
  "eventType": "pull_request.review.submitted",
  "schemaVersion": "1.3",
  "eventTime": "2026-07-15T21:42:13Z",
  "ingestionTime": "2026-07-15T21:42:15Z",
  "source": "github-enterprise",
  "sourceTenant": "corp",
  "sourceEntityId": "repo-a#1842/review/7781",
  "actor": {"type": "human", "id": "person-392"},
  "engineeringChangeId": "EC-01J8...",
  "changeLinkConfidence": 1.0,
  "correlationIds": ["ticket:PAY-431", "deploy:rel-991"],
  "classification": "internal",
  "payload": {}
}
```

Required properties include uniqueness, source identity, event time, schema version, actor type, provenance, classification,

and optional Engineering Change linkage.

12.4. Event time, ingestion time, and processing time

These timestamps answer different questions:

- **event time** — when the source activity occurred;
- **ingestion time** — when the platform received it;
- **processing time** — when a computation used it.

Late events are normal. A deployment system may publish after completion, an incident link may be added days later, and batch imports may arrive out of order.

Stream processors should use event-time windows and watermarks. Historical metrics must support correction and recomputation.

12.5. Delivery semantics and idempotency

At-least-once delivery is usually the practical default for enterprise event ingestion. Consumers must therefore be idempotent.

Use stable event identifiers, source sequence numbers where available, deduplication windows, deterministic upserts, and replay-safe side effects.

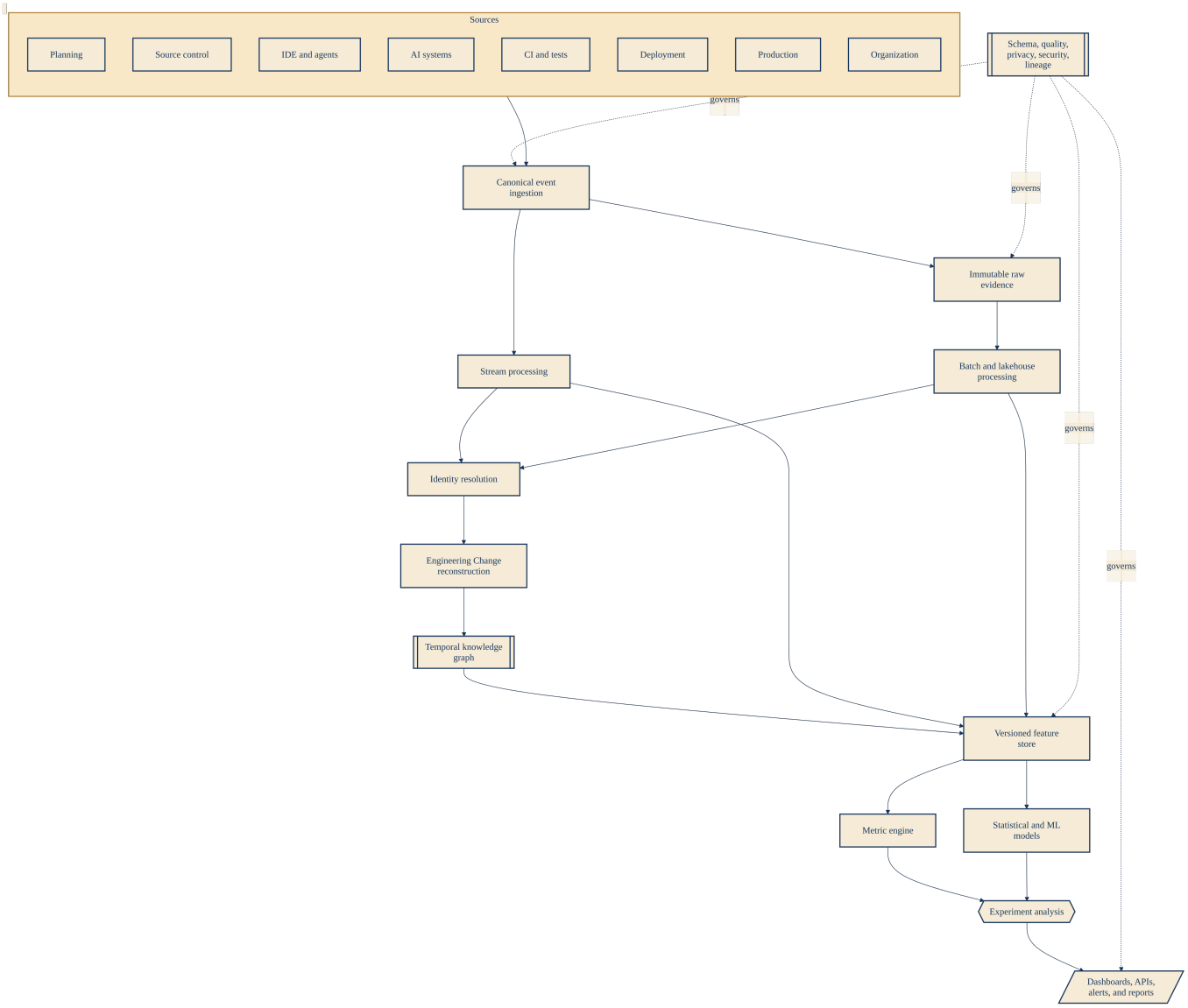
Exactly-once marketing claims do not remove the need for end-to-end idempotency across source APIs, brokers, processors, and sinks.

12.6. Immutable raw evidence

Raw source events should be retained in an immutable, partitioned store before transformation. This provides:

- auditability;
- replay;
- recovery from processing defects;
- metric recomputation;
- schema migration;
- forensic analysis;
- model retraining.

Store both the canonical envelope and original source payload, subject to retention and privacy rules.



12.7. Schema management

Event schemas are contracts. A registry should track name, version, owner, compatibility, classification, retention, example payloads, and quality rules.

Backward-compatible evolution is preferable. Breaking changes should create new versions and dual-read migration periods.

Semantic changes can be more dangerous than structural changes. If a source redefines "review submitted," the event may remain syntactically valid while corrupting historical comparisons.

12.8. Identity resolution

Engineering systems use different identifiers for the same person, repository, service, team, and environment.

Identity resolution creates canonical entities and time-bounded aliases:

$$alias(source, sourceId, canonicalId, [t_s, t_e], confidence)$$

Relevant entities include people, AI actors, repositories, services, teams, tools, environments, artifacts, and Engineering Changes.

Organizational membership changes over time. A pull request should be attributed using the team and ownership state that existed when the event occurred, not today's organization chart.

12.9. Human identity and privacy

Named identity is unnecessary for many system-level questions. Use pseudonymous identifiers, role or team aggregation, and minimum-access principles.

Identity mapping should be a controlled service with audit logs. Downstream datasets should receive only the level of identity required for the decision.

12.10. Stream processing responsibilities

Near-real-time processing is appropriate for:

- current Engineering Change state;
- review, CI, and approval queues;
- aging and stalled work;
- experiment assignment and exposure;
- AI interaction state;
- deployment progression;
- operational alerts;
- rapidly changing guardrails.

Stateful stream processors should support event-time ordering, deduplication, late-event correction, schema evolution, state snapshots, and replay.

12.11. Batch processing responsibilities

Batch or incremental lakehouse processing is appropriate for:

- repository and dependency topology;
- semantic code features;
- historical Engineering Change reconstruction;
- ownership and collaboration networks;
- code-complexity analysis;
- incident attribution;
- cost recomputation;
- model training and calibration;
- long-window experiments.

The distinction is not real-time versus accurate. Both paths must be accurate. It is operational latency versus computational breadth.

12.12. Lambda, kappa, and hybrid designs

A pure streaming architecture simplifies conceptual flow but can make large historical recomputation expensive. A pure batch architecture is simpler for history but cannot support current queues and experiment exposure.

Most enterprises need a hybrid architecture with immutable events, stream-derived operational state, batch-derived historical features, and reconciliation between them.

The same metric definition should produce equivalent results in both paths within declared tolerances.

12.13. Engineering Change reconstruction service

The reconstruction service consumes normalized events and identity mappings, applies deterministic links, computes probabilistic relationships, and maintains temporal Engineering Change graphs.

It should expose:

- create, merge, split, and close operations;
- membership and confidence;
- parent-child relationships;
- evidence and explanation;
- temporal state;
- correction history;
- versioned graph snapshots.

High-confidence explicit identifiers should override weak inferred links unless contradictory authoritative evidence exists.

12.14. Engineering knowledge graph

The knowledge graph connects Engineering Changes to people, AI actors, repositories, services, teams, dependencies, builds, deployments, incidents, and outcomes.

Useful graph queries include:

- find the critical path across repositories;
- identify services repeatedly involved in failed changes;
- locate scarce reviewers bridging many teams;
- trace an incident to contributing artifacts;
- discover repository boundaries causing coordination delay;
- identify shared dependencies with high blast radius.

The graph should preserve temporal state. Current ownership and historical ownership are distinct facts.

12.15. Feature store

A feature store provides versioned, reusable inputs for metrics, experiments, and models.

Feature groups may include:

- change complexity;

- repository topology;
- blast radius;
- author and reviewer familiarity;
- queue conditions;
- AI exposure;
- organizational boundaries;
- historical quality;
- cost assumptions.

Every feature requires definition, entity key, event-time semantics, freshness, owner, source lineage, and version.

Point-in-time correctness is essential. Experiment analysis must use features known before treatment, not information that arrived afterward.

12.16. Metric engine

The metric engine should separate semantic definitions from execution.

A metric specification includes:

- identifier and version;
- construct and decision;
- population and exclusions;
- formula;
- required features;
- aggregation and time window;
- uncertainty method;
- guardrails;
- lineage;
- owner.

The engine can compile specifications into stream, SQL, or batch execution plans while preserving one canonical definition.

12.17. Experiment data plane

Experiment infrastructure requires deterministic assignment, eligibility snapshots, exposure events, treatment versions, outcome joins, guardrails, external-factor annotations, and reproducible analysis datasets.

Assignment should be independent from outcome computation. Exposure must record what actually happened rather than assume that assignment caused use.

12.18. Data quality

Data quality should be measured as a product outcome.

Recommended measures include:

- event completeness;
- duplicate rate;
- late-event rate;
- schema-conformance rate;
- identity-resolution confidence;
- Engineering Change linkage coverage;
- contradictory-link rate;
- source freshness;
- event-to-metric latency;
- replay divergence;
- orphan entity rate.

A metric should display or internally enforce minimum quality thresholds before publication.

12.19. Lineage and reproducibility

Every reported value should be traceable to:

- metric version;
- feature versions;
- model version;
- source schemas;
- source partitions;
- computation code;
- execution time;
- correction history.

A decision made from a dashboard six months ago should be reproducible even if definitions have since changed.

12.20. Storage layers

A common layered design is:

Layer	Purpose
Raw	Immutable canonical envelopes and original payloads
Normalized	Resolved identities, standardized entities, validated event schemas
Graph and state	Engineering Change lifecycle, relationships, current operational state
Feature	Versioned point-in-time features for metrics and models
Metric	Governed aggregates, intervals, and experiment outcomes

Layer	Purpose
Decision product	Dashboards, alerts, APIs, reports, and recommendations

Layer names matter less than clear contracts and lineage.

12.21. Retention and deletion

Retention should be defined by purpose and sensitivity. Raw behavioral telemetry may require shorter retention than aggregated system metrics. Legal holds and incident evidence may require longer retention.

Deletion must propagate through raw data, indexes, graphs, features, models where applicable, and cached decision products. Retention policy should be testable.

12.22. Security model

Controls should include encryption, least privilege, tenant isolation, data classification, row and column security, audit logs, secrets management, export controls, and anomaly detection.

AI telemetry may contain prompts, source snippets, secrets, or sensitive business context. Treat it as potentially high sensitivity.

12.23. Operational reliability

The platform itself requires service-level objectives.

Examples include event-ingestion availability, maximum source lag, metric freshness, reconstruction latency, replay completion, data-quality alerting, and recovery-point objectives.

Measurement systems that silently lose events create misleading confidence. Fail visibly and publish freshness.

12.24. Build versus buy

Commercial engineering-intelligence products can accelerate source integration and dashboards. Internal platforms may be needed for custom Engineering Change reconstruction, proprietary AI telemetry, causal experimentation, and enterprise governance.

A hybrid strategy is common: purchase standard connectors and visualization while retaining canonical events, identity, metric definitions, and experiment data in organizational control.

12.25. Incremental adoption

A practical roadmap is:

1. choose a small set of decisions and metrics;
2. capture source-control, CI, and deployment events;
3. create canonical identities;

4. introduce Engineering Change IDs for selected workflows;
5. add AI interaction telemetry;
6. establish immutable storage and replay;
7. build a versioned metric catalogue;
8. add experiment assignment and exposure;
9. expand to production outcomes and economic models.

Instrumentation should create visible value at each stage.

12.26. Common failure modes

12.26.1. Dashboard-first architecture

Local queries become the unofficial data model and cannot support reproducibility.

12.26.2. Centralizing transformed data without raw evidence

Processing defects cannot be replayed or audited.

12.26.3. Ignoring temporal identity

Current teams and owners are incorrectly applied to historical work.

12.26.4. Mixing event and processing time

Late data creates unstable and unexplained metric changes.

12.26.5. Unversioned schemas and metrics

Historical comparisons become irreproducible.

12.26.6. Collecting excessive behavioral telemetry

Privacy risk increases while trust and validity decline.

12.26.7. Treating one source as universal truth

No tool contains complete intent, implementation, deployment, and outcome evidence.

12.27. Worked example: measuring identity-resolution accuracy



Runnable version: `companion/notebooks/15-identity-resolution.py`.

A platform team has resolved 41,290 distinct source identities across six systems into 4,177 canonical actors. The resolution model uses email matches, SSO federation records, and probabilistic name-and-history clustering.

The team validates the resolution against a hand-labeled sample of 500 identities. Four counts result:

- True positives: 462 (correctly merged into the same canonical actor)
- True negatives: 21 (correctly kept separate)
- False positives: 8 (wrongly merged; two people collapsed into one)
- False negatives: 9 (the same person left as two distinct canonical actors)

Precision is $462 / (462 + 8) = 98.3\%$; recall is $462 / (462 + 9) = 98.1\%$. Both are high, but the analytical consequences are asymmetric. A false positive attributes work from one person to another (a governance and fairness risk); a false negative merely under-attributes and inflates apparent headcount.

The team publishes both metrics on the identity service's SLO dashboard, sets an SLO of precision $\geq 99.0\%$ (higher stakes), and configures the reconstruction service to flag any Engineering Change whose accountability depends on a probabilistic identity match with confidence below 0.90. Manual review remains the resolution path for those cases; automated joins never enter the derived analytical dataset without an SLO-tracked identity-resolution version stamp. Two months later, the flagged-manual-review rate is 0.4% of joins — a workload that the platform team can absorb without shifting the accuracy-versus-throughput trade-off.

12.28. Key takeaways

- Enterprise instrumentation should be designed around evidence, entities, and decisions—not dashboards.
- Canonical event envelopes, immutable raw storage, and replay are foundational.
- Event time, identity, schema, metric, and model versions are required for reproducibility.
- Stream and batch processing serve different latency and breadth needs and must reconcile.
- Engineering Change reconstruction and a temporal knowledge graph connect fragmented enterprise evidence.
- Feature stores require point-in-time correctness for valid experimentation.
- Data quality, privacy, security, lineage, and reliability are first-class platform capabilities.

12.29. Open questions

- How much raw behavioral telemetry is necessary for useful cognitive models?
- Which graph storage and computation patterns scale best to very large repository estates?
- How should semantic schema changes be detected automatically?
- What is the right balance between centralized governance and team-owned instrumentation?
- How can feature and metric definitions remain portable across stream and batch engines?

12.30. Implementation checklist

1. Define a canonical event envelope and schema registry.
2. Store immutable source evidence with replay support.
3. Separate event, ingestion, and processing time.
4. Implement canonical temporal identity resolution.
5. Establish stream and batch responsibilities with reconciliation tests.
6. Build Engineering Change reconstruction with confidence and provenance.
7. Create a temporal engineering knowledge graph.

8. Introduce a versioned, point-in-time-correct feature store.
9. Create a governed metric catalogue and execution engine.
10. Add experiment assignment, exposure, and external-factor records.
11. Publish data-quality and freshness indicators.
12. Apply privacy, retention, security, and deletion policies end to end.

12.30.1. Research foundations and source notes

The enterprise-instrumentation framework in this chapter draws on provenance, event-time stream processing, durable logs, replayable data architecture, site reliability engineering, privacy engineering, and cybersecurity governance. It extends those foundations into an Engineering Intelligence evidence system centered on Engineering Changes, decision-grade metrics, and governed reconstruction.

Durable evidence and replay

Modern data-intensive architecture emphasizes durable logs, immutable or append-oriented evidence, and derived state that can be recomputed. See [kleppmann2017].

This foundation supports several requirements in the chapter:

- preserve source evidence before interpretation;
- separate event capture from derived state;
- retain schema and transformation versions;
- support replay and backfill;
- make corrections auditable;
- avoid irreversible mutation of historical evidence;
- reconstruct current views from durable records.

The purpose is not to require one physical log technology. It is to preserve the semantic property that evidence can be reprocessed when definitions, identity, ownership, or reconstruction logic changes.

Event time, processing time, and late data

Streaming-systems research distinguishes when an event occurred from when it was observed and processed. See [akidau2018].

Engineering Intelligence requires at least:

- source event time;
- ingestion time;
- processing time;
- effective time for corrected or temporal relationships;
- observation maturity time for delayed outcomes.

These distinctions matter because:

- deployment evidence can arrive after source-control events;
- incidents may be linked days later;

- ownership data may be corrected retrospectively;
- AI-provider events may be delayed or retried;
- feature exposure may mature after deployment;
- defect and value outcomes may require long windows.

A metric that uses ingestion time as if it were event time can create false sequence, latency, and attribution conclusions.

Watermarks and outcome maturity

Watermarks provide a practical way to reason about how complete a time-bounded result is likely to be in the presence of late data. See [akidau2018].

The chapter extends this idea into **outcome maturity**. A recently merged Engineering Change may not yet have mature incident, adoption, support, or value evidence.

A decision product should therefore distinguish:

- complete enough for operational state;
- provisional analytical result;
- immature downstream outcome;
- corrected or retracted result;
- final result under a declared maturity policy.

This is an Engineering Intelligence-specific extension of late-data semantics.

Provenance

W3C PROV provides a general model for entities, activities, agents, generation, use, derivation, attribution, and association. See [moreau2013].

The chapter applies provenance to:

- source events;
- identity resolution;
- Engineering Change reconstruction;
- metric computation;
- model inference;
- experiment assignment;
- generated explanation;
- organizational decision.

The central rule is that observed evidence, inferred relationship, model output, and decision must remain distinguishable.

A generated narrative should never become indistinguishable from the evidence that supports it.

Canonical events and source contracts

A canonical event model decouples downstream analytical semantics from vendor-specific APIs.

The chapter's source-contract model requires:

- source owner;
- semantic definition;
- identifier rules;
- timestamp meaning;
- delivery and duplication behavior;
- schema version;
- deletion and correction semantics;
- expected completeness and latency;
- privacy classification;
- replay and backfill behavior.

This is a domain-specific operating standard. It is informed by general data-contract, event-log, and schema-evolution principles described in [kleppmann2017] and [akidau2018].

Idempotency and duplicate handling

Enterprise connectors frequently deliver duplicate, retried, reordered, or partially updated events.

The chapter therefore requires stable event identifiers, source sequence where available, idempotent ingestion, and explicit correction semantics.

Exactly-once processing should not be claimed casually. In many systems, the practical objective is deterministic, idempotent effect from at-least-once delivery.

This treatment is consistent with the distributed-data tradeoffs described in [kleppmann2017] and stream-processing practice in [akidau2018].

Schema evolution

Source and canonical schemas change over time.

Safe evolution requires:

- versioned schemas;
- backward- and forward-compatibility rules where appropriate;
- shadow parsing;
- contract tests;
- lineage impact analysis;
- migration and replay plans;
- deprecation windows;
- explicit semantic-change review.

A technically compatible schema change can still alter meaning. Renaming a field is less dangerous than changing what a timestamp or status represents while retaining the same field name.

The chapter therefore treats semantic versioning and contract governance as separate from serialization compatibility.

Identity resolution

Engineering evidence contains people, service accounts, bots, assistants, autonomous agents, vendors, repositories, services, teams, and organizational units.

Identity resolution should retain:

- source identities;
- canonical identities;
- identity type;
- confidence;
- effective time;
- merge and split history;
- correction provenance;
- re-identification controls.

The chapter's temporal identity model is a domain-specific extension of entity-resolution and temporal-data practices.

It is essential for avoiding errors such as attributing historical work to current teams or treating bots as people.

Temporal ownership

Repository and service ownership changes through reorganization, migration, and operational transfer.

The chapter therefore resolves ownership as of event time rather than using present-day ownership for historical analysis.

This supports valid analysis of:

- team flow;
- incident responsibility;
- architecture boundaries;
- investment outcomes;
- intervention exposure;
- organizational change.

Current ownership remains useful for routing and action, but historical ownership is required for causal and retrospective interpretation.

Engineering Change reconstruction

Instrumentation is not complete when source events are merely collected. The platform must connect them into Engineering Changes.

The chapter's reconstruction layer uses:

- explicit identifiers;
- deterministic lineage;
- rule-based relationships;
- probabilistic linkage;

- temporal membership;
- confidence and explanations;
- human correction.

This architecture builds on provenance in [moreau2013] and replayable data principles in [kleppmann2017], while the Engineering Change semantics are original to this book.

Polyglot storage

No single storage representation is ideal for every Engineering Intelligence workload.

The chapter separates:

- immutable raw evidence;
- operational lifecycle state;
- analytical columnar data;
- graph relationships;
- search and document retrieval;
- feature and model-serving state;
- registries and governance records.

This polyglot approach follows the data-system tradeoff reasoning in [kleppmann2017].

The architectural goal is not maximum technology diversity. It is to preserve clear sources of truth and fit storage to access patterns while maintaining lineage.

Reliability engineering

The Engineering Intelligence platform is itself a production system.

Site reliability engineering provides the foundation for service-level objectives, error budgets, operational toil, monitoring, incident response, and reliable change. See [beyer2016] and [beyer2018].

Relevant platform SLOs include:

- source freshness;
- source completeness;
- ingestion success;
- schema-conformance rate;
- identity-resolution quality;
- Engineering Change linkage quality;
- metric publication latency;
- replay completion;
- correction propagation;
- serving availability.

A platform that publishes unreliable evidence can produce high-consequence organizational errors even when its infrastructure remains technically available.

Data-quality SLOs

Traditional infrastructure SLOs are necessary but insufficient.

The chapter introduces semantic and analytical SLOs such as:

- percentage of deployments linked to an Engineering Change;
- proportion of actors with resolved identity type;
- percentage of metric rows within freshness threshold;
- calibration quality of inferred links;
- rate of unresolved schema drift;
- proportion of outcomes with mature observation windows.

These are original Engineering Intelligence extensions of service-level thinking.

Quality incidents

A data or metric incident occurs when evidence becomes stale, incomplete, mislinked, semantically incorrect, or inappropriately exposed.

The chapter treats these as operational incidents requiring:

- detection;
- severity classification;
- containment;
- correction;
- replay;
- downstream impact analysis;
- affected-decision review;
- practitioner notification where appropriate;
- post-incident learning.

This extends SRE incident-management principles from system availability to evidence integrity.

Privacy engineering

The NIST Privacy Framework provides a risk-based foundation for identifying, governing, controlling, communicating, and protecting privacy-related data processing. See [nist2020-privacy].

The chapter applies these principles through:

- purpose limitation;
- data minimization;
- pseudonymization;
- identity isolation;
- aggregation thresholds;
- retention limits;
- restricted re-identification;

- correction and appeal;
- audit and access review.

The system should collect the least detailed evidence needed for the declared decision rather than all technically available behavioral data.

Security governance

The NIST Cybersecurity Framework provides a general foundation for governing, identifying, protecting, detecting, responding to, and recovering from cybersecurity risk. See [nist2024-csf].

Relevant controls include:

- source authentication;
- least-privilege connector credentials;
- secrets management;
- encryption;
- tenant and environment isolation;
- audit logging;
- supply-chain review;
- incident response;
- disaster recovery;
- vendor-risk management.

Engineering evidence can expose source code metadata, vulnerabilities, organizational structure, and sensitive work patterns. It should therefore be treated as high-value enterprise data.

Access by purpose

Role-based access alone is often too coarse for Engineering Intelligence.

The chapter therefore combines role, data classification, population, and declared purpose.

A platform engineer may need raw connector diagnostics but not identifiable practitioner behavior. A measurement scientist may need pseudonymous event sequences but not names. A manager may need team-level trends but not individual traces.

Purpose-bound access is an original operating extension aligned with the privacy-risk principles in [nist2020-privacy].

Retention and deletion

Retention periods should reflect purpose, legal obligation, analytical maturity, and risk.

The chapter distinguishes:

- immutable technical evidence needed for replay;
- identifiable mappings;
- detailed behavioral traces;
- aggregated metrics;

- experiment records;
- decisions and audit evidence.

These data classes may require different retention and deletion policies.

Deletion requests and source deletions must propagate through derived stores without silently corrupting reproducibility claims. Where exact reconstruction can no longer be retained, the limitation should be recorded explicitly.

Vendor and AI telemetry

Third-party tools may emit incomplete or provider-defined telemetry and may change semantics without notice.

The chapter therefore requires:

- vendor source contracts;
- version and configuration capture;
- raw response retention where permitted;
- reconciliation against billing or usage records;
- model and prompt provenance;
- data-use restrictions;
- export and exit capability;
- treatment of provider-generated labels as claims rather than facts.

AI risk-management principles are informed by [nist2023-airmf], [nist2024-genai], 001, and 894.

Observability versus surveillance

Engineering observability explains system behavior and supports improvement. Surveillance attempts to infer individual diligence, intent, or worth from detailed activity traces.

The chapter preserves this distinction through:

- decision-first instrumentation;
- aggregation by default;
- prohibited individual employment use;
- correction and appeal;
- transparent definitions;
- limited retention;
- independent governance review.

Measurement-validity concerns in [messick1995] and indicator-corruption concerns in [campbell1979] and [goodhart1984] reinforce these safeguards.

Explainability and generated narratives

Engineering Intelligence systems may generate summaries or explanations from evidence.

The chapter requires generated content to retain:

- cited source records;
- metric and model versions;
- uncertainty;
- distinction between observation and inference;
- generation timestamp;
- model provenance;
- human acceptance or correction.

Generated explanations should be treated as navigational and analytical aids, not as authoritative replacements for evidence.

Cost and capacity management

Large-scale ingestion, storage, replay, graph processing, and AI analysis can become expensive.

The chapter therefore tracks cost by:

- source;
- connector;
- pipeline;
- storage tier;
- metric or feature product;
- experiment;
- serving consumer;
- model and inference workload.

This allows platform cost to be evaluated against decision value using the economic methods in [boardman2018] and Chapter 2.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- purpose-bound Engineering Intelligence instrumentation;
- canonical Engineering Change event contracts;
- explicit separation of observed evidence, reconstruction, inference, generated explanation, and decision;
- temporal identity and ownership as mandatory analytical context;
- reconstruction confidence as a platform-quality dimension;
- outcome-maturity states derived from late-data semantics;
- data-quality and semantic SLOs;
- metric and evidence incidents with decision-impact review;
- purpose-aware serving and re-identification isolation;
- replay of metric, reconstruction, and decision products after correction;
- integrated source, metric, model, experiment, evidence, and decision registries.

These extensions should be validated through platform SLO performance, correction history, audit results, practitioner

trust, and the quality of decisions they support.

Chapter 13. Complexity, Blast Radius, and Repository Topology



***Engineering effort cannot be inferred from code size alone. A small change can require** advanced domain knowledge, coordinated deployment, or broad operational safeguards, while a large generated change may be routine. This chapter defines a multidimensional model of Engineering Change complexity, separates intrinsic reasoning difficulty from topological and organizational complexity, formalizes blast radius, and explains how repository structure, dependency graphs, release sequencing, and expertise requirements affect cost, latency, and risk.*

13.1. Why size is not complexity

Lines changed, files touched, and pull-request count are easy to observe. They are useful descriptive features, but they are weak stand-alone measures of difficulty.

Consider three changes:

- a two-line authorization change affecting every customer;
- a thousand-line generated API client update;
- a one-table migration requiring dual writes, backfill, cutover, and rollback.

The first is small but potentially high risk. The second is large but often routine. The third may be modest in code size yet operationally and organizationally complex.

Engineering Intelligence therefore models complexity as a vector rather than a single raw count.

13.2. Complexity dimensions

Let the complexity of Engineering Change e be:

$$\mathbf{C}(e) = [C_i, C_d, C_a, C_v, C_o, C_t, C_n, C_b]$$

where:

- C_i is implementation complexity;
- C_d is domain complexity;
- C_a is architectural complexity;
- C_v is verification complexity;
- C_o is organizational complexity;
- C_t is topological and deployment complexity;
- C_n is novelty;
- C_b is potential blast radius.

Each dimension may be represented by multiple features and uncertainty.

13.3. Implementation complexity

Implementation complexity includes algorithmic difficulty, control flow, concurrency, state management, performance constraints, language and framework complexity, and amount of handwritten logic.

Potential static features include cyclomatic complexity, nesting, data-flow complexity, concurrency primitives, state transitions, exception paths, and semantic change size.

Static analysis should not be used blindly. A complex function may be unchanged context, while a small modified expression may alter an invariant. Change-aware analysis is preferable.

13.4. Domain complexity

Domain complexity reflects specialized knowledge required to reason correctly.

Examples include:

- security and cryptography;
- accounting and taxation;
- distributed consistency;
- machine learning and statistics;
- advanced user interaction;
- privacy and regulation;
- database internals;
- real-time and embedded systems.

Domain complexity can be estimated from code ownership, historical reviewer specialization, topic classification, dependency use, security labels, and expert annotation.

It should not become a fixed ranking of technologies. Complexity depends on the organization’s familiarity and available expertise.

13.5. Architectural complexity

Architectural complexity arises from the number and structure of components affected.

Useful features include:

- services and libraries touched;
- dependency depth;
- fan-in and fan-out;
- API and schema contracts;
- compatibility windows;
- shared-state boundaries;
- synchronous and asynchronous interactions;
- circular dependencies;
- hidden runtime coupling.

A change affecting a high-fan-in shared library may have broader consequences than one affecting an isolated service.

13.6. Verification complexity

Verification complexity measures how difficult it is to establish sufficient confidence.

Factors include:

- nondeterministic behavior;
- production-only conditions;
- test-data requirements;
- environment scarcity;
- observability quality;
- manual validation;
- long-running migration states;
- difficult rollback;
- security or privacy review.

Verification complexity is often a stronger predictor of cycle time than implementation size.

13.7. Organizational complexity

Organizational complexity includes teams, ownership boundaries, approvals, handoffs, vendors, time zones, and decision rights.

A first-order measure may include:

$$C_o(e) = \alpha T_e + \beta H_e + \gamma A_e + \delta X_e$$

where T_e is teams involved, H_e handoffs, A_e approvals, and X_e external organizational boundaries.

The coefficients must be calibrated from observed delay and effort. The same number of teams can create very different friction depending on interface quality and prior collaboration.

13.8. Novelty

Novelty measures distance from what the author, reviewers, team, and organization already understand.

Useful novelty features include:

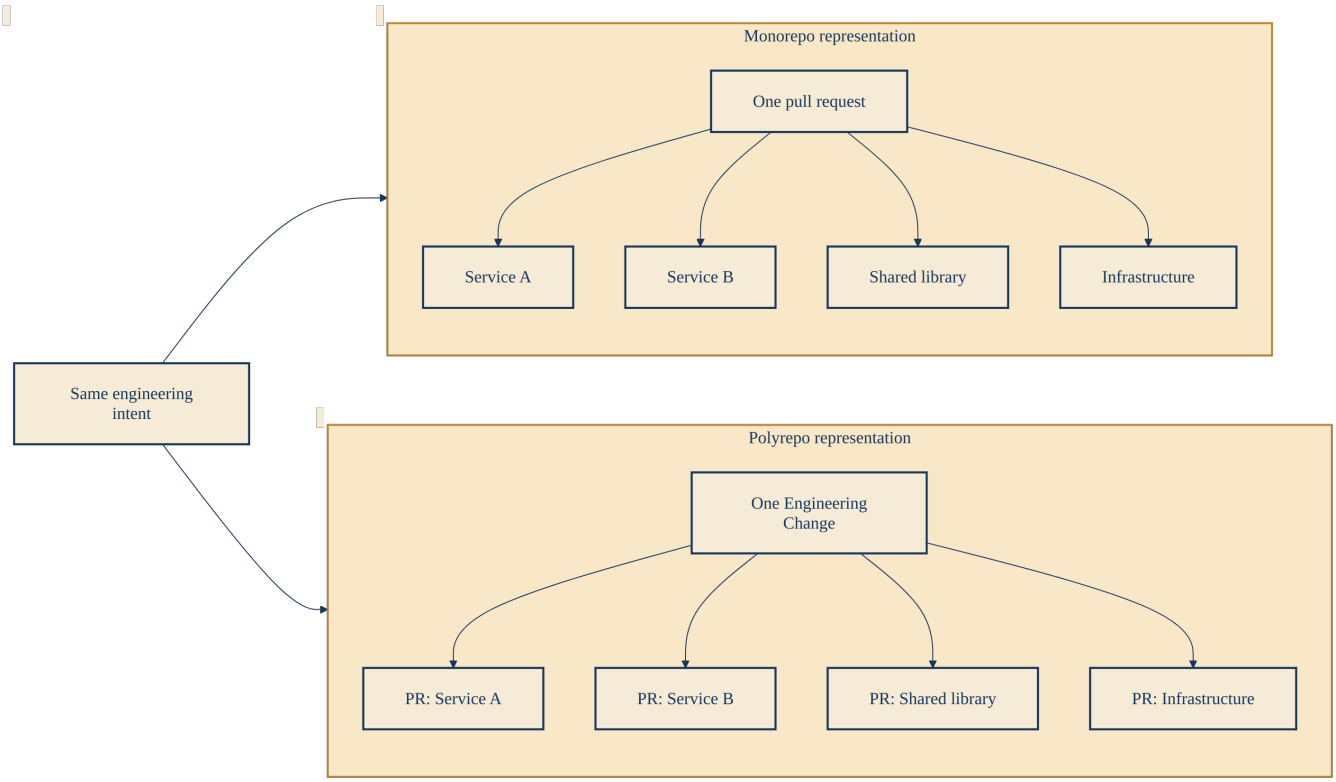
- first use of a framework or service;
- low author familiarity with the code region;
- low reviewer familiarity;
- new dependency or protocol;
- architectural pattern not previously used;
- sparse historical examples;
- high semantic distance from prior changes.

Novelty can increase both active reasoning and uncertainty. AI may reduce search and explanation cost while leaving accountability and verification requirements unchanged.

13.9. Repository topology

Repository structure changes how one Engineering Change appears in source-control data.

In a monorepo, one pull request may touch many services and owners. In a polyrepo environment, the same intent may require many linked pull requests. A repository count is therefore a topology feature, not an intrinsic measure of work.



Relevant topology features include repository count, repository dependency depth, cross-repository build relationships, artifact publication steps, version pinning, release order, and ownership transitions.

13.10. Multi-repository change graphs

Let the multi-repository implementation graph be $G_R = (V_R, E_R)$, where nodes are artifacts or repository-local changes and edges represent dependency, compatibility, or ordering constraints.

The critical path duration is:

$$T_{critical}(e) = \max_{p \in Paths(G_R)} \sum_{v \in p} d(v)$$

Parallel work reduces elapsed time only when dependencies permit it. The parallelism ratio can be approximated as:

$$P(e) = \frac{\sum_{v \in V_R} d(v)}{T_{critical}(e)}$$

A higher value indicates more potential concurrency, though coordination overhead may limit realizable parallelism.

13.11. Deployment topology

Code topology and deployment topology are not the same. Several repositories may deploy as one unit, while one repository may produce many independently deployed services.

Deployment complexity includes:

- number of environments and regions;
- rollout waves;
- ordering constraints;
- feature flags;
- schema compatibility;
- dual-run periods;
- rollback boundaries;
- irreversible steps;
- external dependencies.

A migration may remain operationally active long after all pull requests merge.

13.12. Blast radius



Blast radius is the set and severity of systems, users, data, business processes, and obligations that could be affected if an Engineering Change behaves incorrectly.

Potential blast radius is distinct from realized impact. A change can have high potential blast radius and no incident because controls succeed.

Let affected dimensions be represented as:

$$\mathbf{B}(e) = [U, S, D, F, A, C, R]$$

where:

- U is users or customers;
- S is services and systems;
- D is data volume and sensitivity;
- F is financial exposure;
- A is availability impact;
- C is compliance and legal exposure;
- R is reversibility and recovery difficulty.

A scalar score may be useful for routing but should preserve component drill-down.

13.13. A blast-radius score

One practical form is:

$$BRS(e) = \sum_{k=1}^{K} w_k g_k(B_k(e))$$

where g_k transforms each dimension and w_k reflects organizational priorities.

Because blast-radius dimensions can interact, a nonlinear model may be preferable. For example, sensitive data plus broad customer reach may require stronger controls than either dimension alone.

The score should be calibrated against incidents, expert assessment, and operational policy—not chosen only for numerical convenience.

13.14. Reversibility

Reversibility strongly changes risk. A flagged UI change can be disabled quickly. A destructive data migration may be difficult or impossible to reverse.

Reversibility features include rollback time, rollback completeness, data restoration, compatibility duration, feature-flag coverage, and whether side effects escape organizational control.

A high-blast-radius but highly reversible change may require different controls from a lower-reach irreversible change.

13.15. Change surface versus impact surface

The **change surface** is what was modified. The **impact surface** is what can be affected.

A tiny change in a shared authentication library has a small change surface and large impact surface. A large isolated test rewrite has a large change surface and small impact surface.

Engineering review and experimentation should use both.

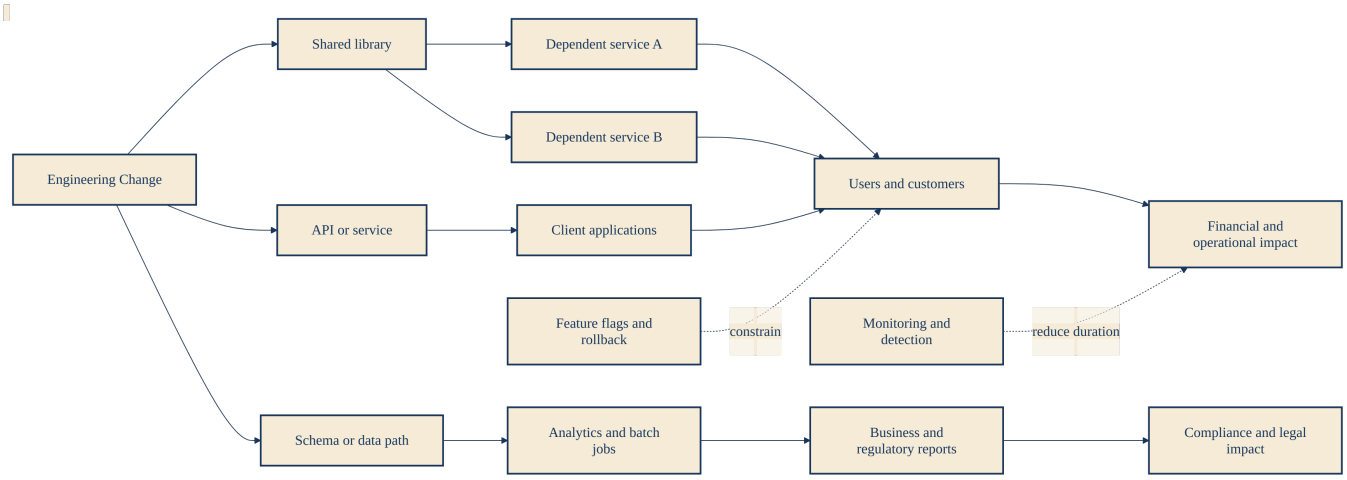
13.16. Measuring dependency reach

Static dependency graphs are useful but incomplete. Runtime traces, service calls, message topics, database access, feature flags, and deployment manifests reveal additional reach.

A combined reach graph may include:

- build-time dependencies;
- runtime calls;
- data flows;
- event subscriptions;
- deployment coupling;
- ownership relationships;
- customer and business-process mappings.

Temporal versions are required because dependencies change.



13.17. Complexity-adjusted comparisons

Raw cycle-time comparisons should control for complexity and blast radius.

A hierarchical model might be:

$$\log Y_e = \beta_0 + \beta^T \mathbf{C}(e) + u_{team(e)} + v_{repo(e)} + \epsilon_e$$

where Y_e is an outcome such as review effort or cycle time.

The residual can help identify unexpectedly high or low friction after accounting for work mix. It should not be used as an individual performance ranking.

13.18. Complexity and review routing

Complexity features can route changes to appropriate controls:

- low-risk routine changes receive automated checks and lightweight review;
- high-domain-complexity changes receive specialist review;
- high-blast-radius changes require stronger verification and rollout safeguards;
- high-novelty changes receive additional design and knowledge-sharing support;
- irreversible changes require explicit recovery planning.

Routing policies should be evaluated experimentally and monitored for false negatives.

13.19. AI and complexity

AI performance varies by complexity dimension. A model may be strong at common framework patterns and weak at novel architecture, distributed state, or organizational context.

Evaluation should therefore stratify by domain, novelty, dependency depth, blast radius, and verification complexity.

A global average can conceal unsafe performance in important subpopulations.

13.20. Estimation methods

Complexity features can be obtained from:

- static and semantic code analysis;
- repository and dependency graphs;
- service catalogues;
- runtime telemetry;
- ownership and review history;
- deployment manifests;
- issue and design-document classification;
- expert labels;
- incident history.

Use supervised models only after establishing interpretable baselines. Preserve feature explanations for routing and governance.

13.21. Validation

A complexity model should predict outcomes not used directly in its construction, such as active effort, review duration, defect probability, specialist involvement, or rollback difficulty.

Validate calibration by complexity band and subgroup. A score that predicts average effort but underestimates security changes is not acceptable for risk routing.

13.22. Common failure modes

13.22.1. Lines changed as complexity

Size contributes evidence but misses domain, architecture, verification, and impact.

13.22.2. Repository count as work count

Repository strategy determines artifact boundaries.

13.22.3. One universal complexity score

A scalar can hide why a change is difficult and which control is appropriate.

13.22.4. Static dependencies as complete reach

Runtime, data, deployment, and organizational coupling may be absent.

13.22.5. Realized incidents as blast radius

Potential impact exists even when controls prevent failure.

13.22.6. Ignoring reversibility

Reach alone does not determine operational risk.

13.23. Worked example: scoring the blast radius of a shared-auth library change



Runnable versions of this example live in the companion notebooks: * `companion/notebooks/06-blast-radius-scoring.py` * `companion/notebooks/16-dependency-blast-radius.py` * `companion/notebooks/17-critical-path.py`

A shared authentication library is updated to change how session tokens are refreshed. The team classifies the change as "small" — 38 lines of code, one repository, passing tests.

The blast-radius score developed in this chapter looks at three inputs. Reach: the service catalogue shows 47 downstream services import the library at the pinned version; 31 will pick up the change on their next deploy, and 8 have deploy cadences under 24 hours. Exposure: 5 of the downstream services are internet-facing and handle authenticated user requests. Reversibility: the change modifies a client-visible cookie attribute; rollback requires simultaneous coordinated redeployment across the fleet or a period of graceful mixed-version handling.

A simple weighted score, with each factor normalized to [0, 1]:

$$BR = 0.4 \times R_{reach} + 0.3 \times R_{exposure} + 0.3 \times R_{reversibility}$$

Assigning $R_{reach} = 0.66$ (31 of 47 immediate uptake), $R_{exposure} = 0.80$ (five customer-facing services), $R_{reversibility} = 0.85$ (coordinated rollback required), gives $BR \approx 0.76$ — the top quartile.

The team’s 38-line "small" change is, by this measure, a high-blast-radius change that warrants a coordinated rollout schedule, an explicit rollback rehearsal, and specialist review — none of which the pull request otherwise triggered. This is exactly the case the metric exists to catch. The corresponding policy is not "block all top-quartile changes" but "require the coordinated-rollout playbook for top-quartile changes"; the score sets the process, not the verdict.

13.24. Key takeaways

- Engineering Change complexity is multidimensional and cannot be inferred from code size alone.
- Domain, architecture, verification, organization, topology, novelty, and blast radius should be modeled separately.
- Repository and deployment topology shape how work appears and how it must be coordinated.
- Multi-repository work should be represented as a dependency graph with a critical path.
- Blast radius measures potential impact, not only realized incidents.
- Change surface and impact surface are distinct.
- Complexity models should support adjustment, routing, and experimentation—not individual ranking.

13.25. Open questions

- Which semantic features best capture reasoning difficulty beyond static complexity?
- How should business-process and regulatory reach be joined to technical dependency graphs?
- Can reversibility be estimated automatically from deployment and data-migration patterns?
- What complexity dimensions best predict AI failure modes?

- How should organizations calibrate risk-routing thresholds when severe failures are rare?

13.26. Implementation checklist

1. Define a multidimensional complexity schema.
2. Combine code, dependency, runtime, deployment, organizational, and business evidence.
3. Separate repository topology from intrinsic work complexity.
4. Build multi-repository Engineering Change graphs and critical-path measures.
5. Define blast-radius dimensions and preserve component drill-down.
6. Add reversibility and recovery features.
7. Validate models against effort, quality, and operational outcomes.
8. Stratify AI and workflow evaluations by complexity and blast radius.
9. Use complexity for routing and adjustment, not individual ranking.
10. Version all dependency and ownership graphs temporally.

13.26.1. Research foundations and source notes

The complexity and blast-radius framework in this chapter draws on modularity, information hiding, socio-technical congruence, organizational communication structure, team cognitive load, distributed coordination, reliability engineering, and resilience. It extends those foundations into an Engineering Change-level model of consequence, reversibility, and architectural risk.

Modularity and information hiding

Parnas's work on information hiding provides a foundational argument for organizing modules around design decisions likely to change rather than merely around execution order. See [parnas1972].

This supports the chapter's treatment of architecture as a mechanism that changes the cost and risk of future Engineering Changes.

A good boundary can reduce:

- the amount of code and behavior that must be understood;
- the number of teams involved;
- compatibility obligations;
- validation scope;
- deployment coordination;
- potential failure propagation.

A poor boundary can repeatedly impose these costs even when each individual code change is small.

Conway's Law

Conway observed that organizations tend to design systems whose structures mirror their communication structures. See [conway1968].

The chapter uses this as a starting point for analyzing the alignment among:

- service boundaries;
- repository boundaries;
- team ownership;
- communication paths;
- approval structures;
- deployment responsibility.

The presence of structural similarity does not prove that one side caused the other in a specific case. It does establish that organizational and technical structures should be studied together.

Socio-technical congruence

Socio-technical congruence research examines whether coordination relationships align with technical dependencies. See [cataldo2006].

When a change creates dependencies across teams that do not routinely coordinate, the organization may experience:

- slower owner discovery;
- more handoffs;
- inconsistent assumptions;
- duplicated validation;
- delayed decisions;
- higher integration risk.

This supports the chapter's distinction between purely technical coupling and coordination burden.

Distributed coordination

Research on distributed software development documents the difficulty created by distance, organizational boundaries, and coordination dependencies. See [herbsleb2007].

The chapter therefore treats time zones, vendor boundaries, organizational separation, and decision-right fragmentation as contributors to change complexity.

A dependency graph is not fully understood until the ownership and communication graph are considered as well.

Team cognitive load

Team Topologies emphasizes the importance of team cognitive load and defines interaction modes intended to support fast flow. See [skelton2019].

The chapter applies this perspective at Engineering Change level by asking:

- How many domains must a team understand?
- How many systems must be reasoned about simultaneously?
- How much external coordination is required?
- Which dependencies require specialist knowledge?
- How stable and documented are the interfaces?

The resulting **cognitive surface** is not directly observable and should be inferred through validated indicators rather than treated as a simple count.

Complexity is multidimensional

The chapter rejects a universal scalar complexity measure.

Relevant dimensions include:

- implementation complexity;
- domain complexity;
- dependency depth;
- ownership fragmentation;
- validation surface;
- deployment coupling;
- data consequence;
- security and compliance exposure;
- novelty;
- reversibility;
- uncertainty.

A change can be simple in one dimension and dangerous in another. Two lines of configuration can have a larger consequence than thousands of lines of generated code.

This multidimensional model is an Engineering Intelligence-specific synthesis built on the modularity and socio-technical foundations above.

Static complexity measures

Static code measures can provide useful local signals about control flow, nesting, size, dependency count, and maintainability.

They do not fully represent:

- organizational coordination;
- runtime coupling;
- data migration risk;
- deployment sequence;
- user or financial consequence;
- regulatory exposure;
- operational reversibility.

The chapter therefore treats static complexity as one evidence family within a broader Engineering Change model rather than as architecture quality itself.

Dependency graphs

Repository, package, service, runtime, deployment, data, and organizational graphs reveal different forms of coupling.

Useful measures include:

- fan-in and fan-out;
- path length;
- strongly connected components;
- centrality;
- dependency depth;
- bridge nodes;
- ownership boundaries;
- change co-occurrence;
- deployment synchronization;
- shared failure domains.

Graph measures are descriptive structural features. Their relationship to effort, incidents, or value should be validated empirically rather than assumed.

Change coupling

Files, modules, repositories, or services that frequently change together may reveal hidden coupling.

Change-coupling evidence can complement declared dependency graphs because actual engineering behavior may expose relationships absent from architecture documentation.

However, co-change can also arise from:

- broad formatting changes;
- generated code;
- shared release processes;
- ownership conventions;
- repeated migrations;
- one large initiative.

The chapter therefore requires contextual interpretation and Engineering Change reconstruction before treating co-change as architectural coupling.

Blast radius

Blast radius is the scope and consequence of plausible failure from an Engineering Change.

The chapter models it across several dimensions:

- users affected;
- services and regions affected;
- data affected;
- financial exposure;

- security boundaries crossed;
- compliance obligations;
- dependency propagation;
- recovery difficulty;
- duration of consequence.

This multidimensional blast-radius model is an original Engineering Intelligence construct.

It should be calibrated against incident, rollback, support, and production evidence rather than inferred solely from static topology.

Likelihood and consequence

Risk should distinguish the probability of failure from the impact if failure occurs.

A change with low failure probability can still require strong controls when consequence is catastrophic.

The chapter therefore separates:

- defect or failure likelihood;
- exposure;
- blast radius;
- detectability;
- reversibility;
- recovery time;
- tail consequence.

This structure is consistent with reliability and risk-management thinking, including the operational approaches described in [beyer2016] and [beyer2018].

Reversibility

Reversibility reduces the cost of being wrong.

Relevant properties include:

- tested rollback;
- feature-flag isolation;
- backward-compatible deployment;
- data reversibility;
- dual-write or shadow operation;
- regional or population-limited rollout;
- automated stop conditions;
- preserved prior artifacts and configuration.

The chapter treats reversibility as a primary architectural and economic variable rather than a deployment implementation detail.

This is an Engineering Intelligence-specific synthesis aligned with reliable change practices in [humble2010],

[beyer2016], and [kim2021-devops].

Observability and detection

The consequence of a failure depends partly on how quickly it is detected and understood.

A broad change with excellent detection, containment, and rollback can be safer than a smaller opaque change.

The chapter therefore includes:

- detection latency;
- diagnostic clarity;
- ownership reachability;
- telemetry coverage;
- automated containment;
- rollback readiness.

These ideas draw on SRE practices in [beyer2016] and [beyer2018].

Resilience engineering

Resilience concerns a system's ability to adapt, absorb disturbance, and continue functioning under unexpected conditions.

Woods's work on graceful extensibility provides a foundation for thinking about adaptive capacity beyond static robustness. See [woods2015].

The chapter applies this by examining whether architecture provides:

- spare response capacity;
- alternate operating paths;
- degraded modes;
- clear escalation;
- bounded failure domains;
- flexible rollback and isolation;
- accessible expert knowledge.

Resilience cannot be reduced to incident count. A system may appear stable until it encounters conditions outside its designed envelope.

High-reliability organizing

High-reliability research emphasizes sensitivity to operations, reluctance to simplify, attention to weak signals, and resilience. See [weick2007].

This supports the chapter's recommendation to combine structural risk scores with:

- incident narratives;
- near misses;

- practitioner concerns;
- override patterns;
- repeated manual safeguards;
- unusual coordination behavior.

Weak signals may reveal hidden architectural fragility before a major incident occurs.

Incident learning

Systems-oriented incident analysis warns against attributing complex failure to one visible human action. See [dekker2014].

The chapter therefore links incidents back to the complete Engineering Change and surrounding system:

- technical dependencies;
- decision conditions;
- review and validation;
- deployment context;
- organizational constraints;
- observability;
- recovery capability.

This supports learning about architecture rather than merely assigning blame.

Critical paths and synchronization

Multi-repository and multi-team Engineering Changes often contain parallel work plus synchronization points.

Critical-path length, waiting at dependency boundaries, and release sequencing can dominate elapsed time even when total implementation effort is moderate.

The chapter combines dependency-graph analysis with queueing concepts from [little1961], [kingman1961], and [hopp2011] to explain why constrained synchronization points create nonlinear delay.

Ownership entropy

The chapter introduces **ownership entropy** as an operational way to describe how responsibility for a change is distributed across teams or domains.

Higher fragmentation may imply:

- more handoffs;
- less complete local context;
- slower decision-making;
- greater coordination effort;
- more ambiguous accountability.

Ownership entropy is an original Engineering Intelligence construct. It should be calibrated against observed flow, coordination effort, and outcomes rather than treated as intrinsically harmful.

Some complex changes legitimately require diverse ownership.

Validation surface

The **validation surface** is the set of behaviors, environments, data conditions, interfaces, and risks that must be checked before an Engineering Change can be accepted safely.

It may include:

- unit and integration behavior;
- compatibility;
- migration state;
- security and privacy;
- performance and scale;
- regional behavior;
- rollback;
- production verification.

This concept is original to the book and extends ordinary test-count measures into a consequence-oriented model.

Architecture as an economic variable

Architecture affects the recurring cost of change.

A boundary that reduces coordination, validation, deployment coupling, and recovery effort can create durable capability value. A poorly designed dependency can tax every future change.

The chapter's economic treatment is informed by cost-benefit analysis in [boardman2018] and the productivity-realization literature in [brynjolfsson1993] and [brynjolfsson2017].

Architectural value should be measured through longitudinal change outcomes, not asserted solely from design intent.

Causal evaluation of architecture

Architecture changes are difficult to evaluate causally because they are rarely randomized and often coincide with broader organizational changes.

The chapter therefore recommends:

- phased migration;
- matched comparison populations;
- difference-in-differences;
- interrupted time series;
- synthetic controls;
- mechanism and guardrail measures;
- long follow-up windows.

The methodological foundations are described in [shadish2002], [angrist2009], [imbens2015], and [hernan2020].

Calibration and model drift

Blast-radius and complexity models should be calibrated against observed outcomes.

Relevant evidence includes:

- incidents;
- rollbacks;
- review effort;
- change latency;
- rework;
- support impact;
- security findings;
- failed or delayed migrations.

Architecture, ownership, and deployment topology evolve. A model trained on old structure may become invalid even when its computation continues successfully.

The chapter therefore requires model-versioning, drift detection, subgroup calibration, and correction.

Governance and prohibited use

Complexity and blast-radius scores can be misused to rank teams or penalize people assigned difficult work.

Measurement-validity principles in [messick1995] and indicator-corruption concerns in [campbell1979] and [goodhart1984] support the chapter's restrictions.

These measures should normally be used to:

- route review and validation;
- prioritize architecture investment;
- design rollout controls;
- identify hidden dependencies;
- improve resilience;
- estimate uncertainty.

They should not become simplistic performance scores.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- Engineering Change-level multidimensional complexity;
- multidimensional blast radius spanning technical, user, data, financial, security, and compliance consequence;
- ownership entropy;
- validation surface;
- cognitive surface;
- reversibility as a first-class metric;
- confidence-aware blast-radius prediction;

- connected repository, service, runtime, deployment, ownership, and incident graphs;
- architecture economics measured through longitudinal Engineering Change outcomes;
- integration of weak signals, near misses, and practitioner evidence into architecture intelligence.

These constructs should be treated as measurable hypotheses and calibrated against real engineering and operational outcomes.

Chapter 14. The Engineering Intelligence Platform



***The preceding chapters define the concepts, metrics, and evidence required to understand** engineering systems. This chapter turns those ideas into a concrete platform architecture. It describes the major services, storage systems, APIs, computation paths, governance controls, decision products, and operating model needed to build an Engineering Intelligence platform incrementally in a large enterprise.*

14.1. Platform objectives

An Engineering Intelligence platform should answer four classes of questions:

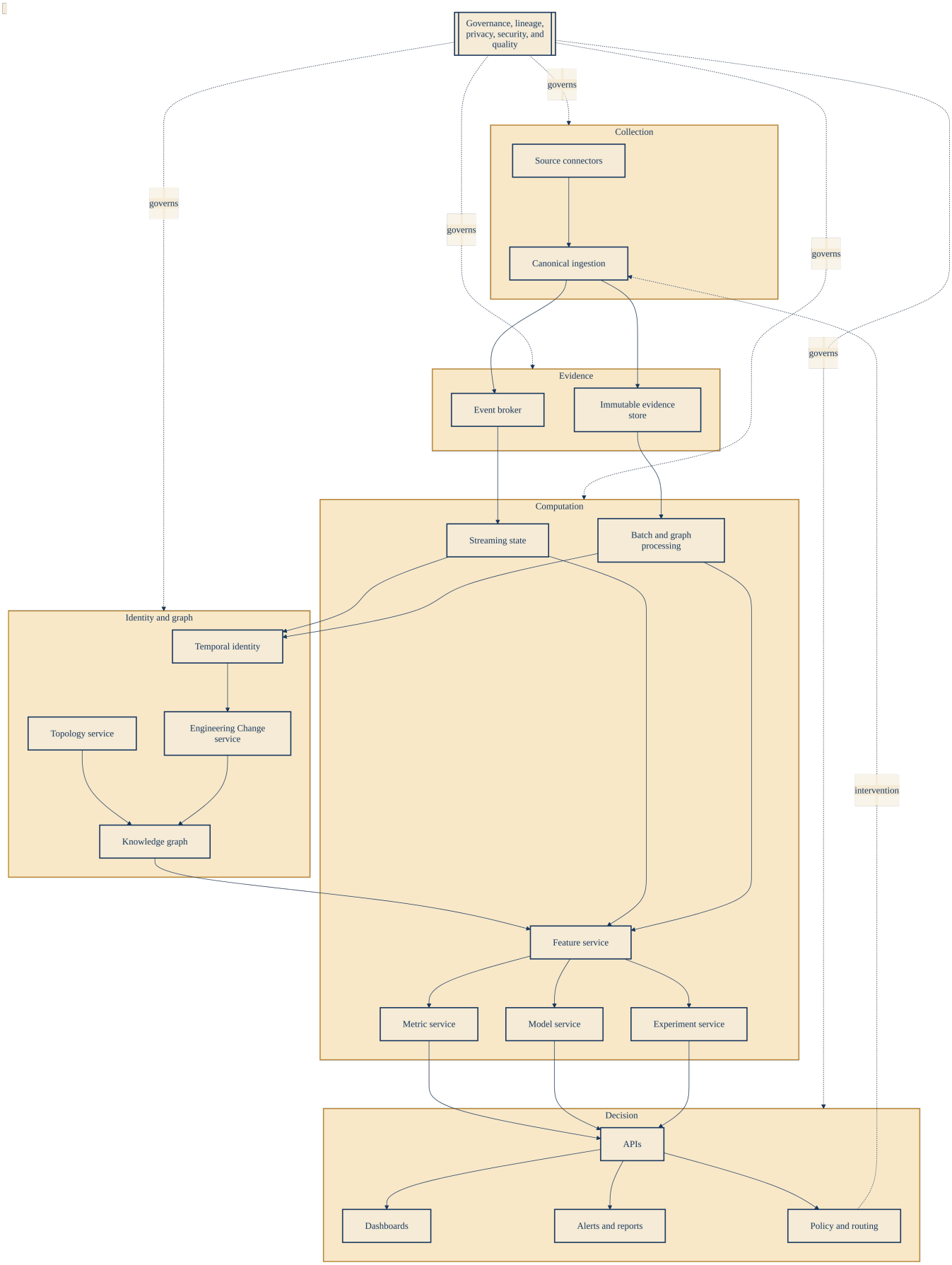
- **What happened?** — reconstruct Engineering Changes, queues, reviews, deployments, and outcomes.
- **Why did it happen?** — identify constraints, complexity, dependencies, and external factors.
- **What is likely to happen next?** — predict delay, risk, cost, and quality outcomes.
- **What should we change?** — support experiments, routing, prioritization, and investment decisions.

The platform is not merely a reporting warehouse. It is an evidence and decision system.

14.2. Core architectural principles

1. Preserve immutable raw evidence.
2. Separate canonical entities from source-specific identifiers.
3. Treat the Engineering Change as the primary unit of analysis.
4. Version events, features, metrics, models, and policies.
5. Support both near-real-time operations and retrospective recomputation.
6. Keep causal experiment assignment independent from metric computation.
7. Expose uncertainty, provenance, and data quality with every derived result.
8. Minimize behavioral surveillance and prohibit unsupported individual ranking.

14.3. Logical architecture


Governance, lineage, privacy, security, and quality
governs
governs
governs
governs
governs
intervention
Collection
Source connectors
Canonical ingestion
Evidence
Event broker
Immutable evidence store
Computation
Streaming state
Batch and graph processing
Identity and graph
Temporal identity
Topology service
Engineering Change service
Knowledge graph
Feature service
Metric service
Model service
Experiment service
Decision
APIs
Dashboards
Alerts and reports
Policy and routing

The architecture is divided into seven planes:

Plane	Responsibilities
Collection	Connectors, webhooks, CDC, polling, agent telemetry, schema validation
Evidence	Immutable event log, raw payload store, deduplication, replay
Identity and graph	Canonical identities, Engineering Change reconstruction, topology, provenance
Computation	Streaming state, batch features, metric engine, models, experiments
Governance	Schema registry, lineage, privacy, access, retention, audit
Decision	Dashboards, alerts, APIs, reports, recommendations
Control	Experiment assignment, policy delivery, model and metric versions

14.4. Service boundaries

A practical service decomposition includes:

14.4.1. Source connector service

Connects to Git hosting, issue tracking, CI/CD, AI tools, observability, service catalogues, and organizational systems. It translates source records into canonical envelopes while retaining original payloads.

14.4.2. Identity service

Resolves people, AI actors, repositories, services, teams, environments, and tools into stable temporal identities.

14.4.3. Engineering Change service

Creates, links, merges, splits, and closes Engineering Changes. It exposes lifecycle state, graph membership, confidence, and provenance.

14.4.4. Topology service

Maintains repository, service, data, deployment, ownership, and business dependency graphs.

14.4.5. Feature service

Publishes versioned point-in-time features for changes, repositories, teams, services, and experiments.

14.4.6. Metric service

Executes governed metric definitions and exposes current and historical values with lineage and uncertainty.

14.4.7. Experiment service

Manages eligibility, assignment, exposure, guardrails, external factors, and experiment status.

14.4.8. Model service

Hosts predictive, causal, ranking, and reconstruction models with versioned inputs and monitoring.

14.4.9. Decision product service

Provides APIs, dashboards, alerts, and reports tailored to operational, engineering, product, and executive users.

14.5. Canonical APIs

The Engineering Change API should expose operations such as:

```
GET /engineering-changes/{id}
GET /engineering-changes/{id}/graph
GET /engineering-changes/{id}/metrics
POST /engineering-changes/{id}/members
POST /engineering-changes/{id}:merge
POST /engineering-changes/{id}:split
```

A metric API should require explicit version and time context:

```
GET /metrics/review-friction/v2.1
  ?entityType=engineering_change
  &entityId=EC-01J8...
  &asOf=2026-07-15T22:00:00Z
```

A result should return value, interval, sample basis, quality state, and lineage.

14.6. Storage architecture

No single database is optimal for all workloads.

Store	Purpose
Object storage or lakehouse	Immutable raw events, normalized history, batch features, reproducible snapshots
Event broker	Low-latency transport, ordering, replay, stream processing
Relational store	Experiment registry, metric catalogue, governance metadata, operational state
Graph store	Engineering Change, repository, service, ownership, and dependency relationships
Search index	Interactive discovery across changes, artifacts, findings, and explanations

Store	Purpose
Feature store	Point-in-time online and offline model inputs
Analytical warehouse	Large-scale aggregation, ad hoc analysis, and reporting

The canonical source of truth should be explicit for every entity and fact.

14.7. Stream processing

Streaming computation maintains:

- current lifecycle state;
- queue depth and age;
- experiment exposure;
- AI interaction state;
- build and deployment progression;
- freshness and quality alerts;
- rapidly changing guardrails.

State transitions should be deterministic and replayable. Stream processors should emit changelogs rather than silently mutate opaque state.

14.8. Batch and graph computation

Batch computation handles:

- historical reconstruction;
- semantic code analysis;
- dependency graph generation;
- ownership and collaboration networks;
- complexity and blast-radius features;
- experiment datasets;
- cost and value recomputation;
- model training and calibration.

Incremental graph processing is preferable where possible, but complete recomputation must remain available for recovery and definition changes.

14.9. Metric specification language

A declarative metric definition reduces divergence between dashboards and experiments.

```
id: review_friction
version: 2.1.0
entity: engineering_change
```

```yaml
construct: review_process_friction
population: accepted_changes
features:
  - review_queue_minutes
  - review_response_cycles
  - context_recovery_events
  - validation_minutes
aggregation: weighted_sum
uncertainty: bootstrap_clustered_by_repository
guardrails:
  - severe_escape_rate
  - rollback_rate
owner: engineering-intelligence
```

The execution engine can compile the definition into SQL, batch, or streaming plans.

14.10. Policy and routing engine

Metrics and models become operational when they influence routing and policy.

Examples include:

- send high-blast-radius changes to specialist review;
- suppress low-confidence duplicate AI comments;
- prioritize aged changes with high cost of delay;
- require stronger rollout controls for irreversible migrations;
- allocate scarce reviewers to changes with highest expected risk reduction.

Every policy should be versioned, explainable, and linked to an experiment or governance decision.

14.11. Decision products

Different audiences require different views.

14.11.1. Engineer view

Shows current change state, unresolved dependencies, review queues, AI findings, expected bottlenecks, and production outcome.

14.11.2. Team view

Shows flow distributions, work in process, review constraints, quality, cognitive burden, and experiment status.

14.11.3. Platform view

Shows infrastructure latency, tool cost, data quality, model drift, and source freshness.

14.11.4. Executive view

Shows portfolio-level cost, value, risk, capacity constraints, and evidence from interventions.

A good decision product permits drill-down from aggregate conclusion to raw evidence without exposing unnecessary individual detail.

14.12. Data contracts and ownership

Each domain should own the quality of the events it emits. The platform team owns canonical contracts, identity, lineage, and shared computation.

A data contract includes schema, semantics, owner, SLO, classification, retention, compatibility, and quality checks.

Producers should not be able to change semantics silently.

14.13. Platform SLOs

Suggested service-level objectives include:

- event ingestion availability;
- source-to-platform lag;
- maximum duplicate and loss rate;
- Engineering Change reconstruction latency;
- metric freshness;
- experiment assignment availability;
- replay completion time;
- lineage completeness;
- data-quality alert time.

Freshness should be visible in every product. Stale metrics should not appear current.

14.14. Security and privacy architecture

The platform may contain source code, prompts, behavioral metadata, organizational relationships, and financial assumptions.

Required controls include:

- encryption and key management;
- least-privilege service identities;
- row and column access control;
- pseudonymous identity by default;
- classified event handling;
- retention and deletion workflows;
- export controls;
- audit trails;
- separation between system analysis and performance management.

14.15. Reference implementation roadmap

14.15.1. Phase 1: evidence foundation

Ingest source-control, CI, and deployment events. Establish canonical envelopes, raw storage, replay, and identity resolution.

14.15.2. Phase 2: Engineering Change graph

Introduce explicit IDs where practical and probabilistic reconstruction elsewhere. Expose graph and lifecycle APIs.

14.15.3. Phase 3: governed metrics

Create the metric catalogue, feature pipelines, lineage, and quality indicators. Deliver flow and quality views.

14.15.4. Phase 4: experimentation

Add deterministic assignment, exposure logging, external-factor registry, guardrails, and reproducible analysis datasets.

14.15.5. Phase 5: AI and cognitive instrumentation

Capture AI contributions, human actions, validation burden, trust, and quality-adjusted offload.

14.15.6. Phase 6: predictive and prescriptive intelligence

Add delay, risk, complexity, and routing models. Introduce policy decisions only after experimental validation.

14.16. Suggested technology-neutral components

The design can be implemented with different stacks, but common component classes include:

- event broker;
- object store or lakehouse;
- stream processor;
- SQL transformation engine;
- graph database or graph-processing layer;
- relational metadata store;
- feature store;
- search engine;
- model-serving platform;
- dashboard and notebook environment;
- policy engine.

Avoid coupling the conceptual architecture to one vendor.

14.17. Build-versus-buy decisions

Buy commodity capabilities when they do not define strategic differentiation: standard connectors, visualization, cataloguing, or infrastructure monitoring.

Retain control of canonical event history, identity, Engineering Change reconstruction, metric definitions, experiment assignments, proprietary AI telemetry, and policy logic.

Vendor products should integrate through contracts rather than become the only location where evidence exists.

14.18. Operating model

A successful platform requires more than software.

Recommended roles include:

- platform engineering;
- data engineering;
- applied statistics and causal inference;
- ML and information retrieval;
- developer experience;
- security and privacy;
- domain representatives;
- metric and experiment owners.

A governance council should approve high-impact metrics, experiments, and individual-level uses.

14.19. Common failure modes

14.19.1. Building a universal data lake without decisions

Data accumulates while operational value remains unclear.

14.19.2. Coupling metrics to dashboards

Definitions diverge across products and cannot be reproduced.

14.19.3. Hiding provenance

Users cannot challenge or correct derived conclusions.

14.19.4. Centralizing all ownership

The platform team cannot understand every source and domain without producer contracts.

14.19.5. Automating policy too early

Predictive models are deployed before validation and create self-reinforcing bias.

14.19.6. Ignoring operating model

Technology exists, but no one owns definitions, experiments, or action.

14.20. Worked example: sequencing platform delivery for first-year value

A newly formed engineering-intelligence team has a mandate to demonstrate value within twelve months. They have four candidate initial capabilities: an ingestion layer, Engineering Change reconstruction, a metric governance layer, and an experiment platform.

Ranking by decision-value-per-effort:

- Ingestion + descriptive review-latency dashboard: 3 engineer-quarters, unblocks the CTO's monthly reliability review, low uncertainty.
- Engineering Change reconstruction: 6 engineer-quarters, unblocks blast-radius scoring and coordination-cost measurement, medium uncertainty because linkage confidence depends on source-system semantics.
- Metric governance layer: 4 engineer-quarters, defers value but is a prerequisite for anything shown on an executive scorecard.
- Experiment platform: 8 engineer-quarters, high value but useless until reconstruction is in place.

The team has 18 engineer-quarters for the year. A naive plan would try to build all four in parallel; the resulting slippage delays every deliverable.

A sequenced plan takes ingestion + descriptive dashboard through governance in the first quarter (7 engineer-quarters), builds reconstruction against that governed ingest in Q2-Q3 (6 engineer-quarters, with governance work continuing in the background at 1 engineer-quarter), and starts the experiment platform in Q4 (4 engineer-quarters against the reconstructed graph). By month twelve, the team has produced two governed executive-scorecard metrics, one causal experiment on review routing, and a blast-radius measure the incident review committee has adopted — a small, credible portfolio that justifies year-two funding.

The alternative sequencing — experiment platform first — would have delivered no consumable output for eight months and made continued investment politically difficult.

14.21. Key takeaways

- The Engineering Intelligence platform is an evidence, experimentation, and decision system—not merely a dashboard stack.
- Clear service boundaries separate collection, identity, graph, features, metrics, experiments, models, and decision products.
- Immutable evidence, temporal identities, point-in-time features, and versioned definitions are foundational.
- Operational policies should be explainable, versioned, and experimentally validated.
- Different audiences require different views, but all should share one governed semantic layer.
- Incremental adoption should deliver value before advanced prediction or automation.

14.22. Open questions

- Which platform capabilities should be centralized versus federated to teams?
- How can metric specification languages remain expressive without becoming ungovernable?

- What is the best storage pattern for temporal engineering graphs at enterprise scale?
- How should policy engines incorporate uncertainty and human override?
- Which parts of the platform are likely to become standardized industry infrastructure?

14.23. Implementation checklist

1. Define platform objectives and decision products before selecting technology.
2. Establish canonical events, immutable evidence, and temporal identity.
3. Implement the Engineering Change and topology services.
4. Create versioned feature, metric, and experiment services.
5. Expose lineage, confidence, and data quality through APIs.
6. Separate assignment, exposure, and outcome computation.
7. Add policy routing only after validated experiments.
8. Publish platform SLOs and freshness indicators.
9. Establish privacy, security, retention, and governance controls.
10. Build incrementally from evidence capture to prescriptive intelligence.

14.23.1. Research foundations and source notes

The platform architecture in this chapter draws on data-intensive systems, event-time stream processing, provenance, site reliability engineering, privacy engineering, cybersecurity governance, and AI risk management. It extends those foundations into a decision-oriented evidence platform for software engineering.

Durable logs and derived state

The architectural distinction between durable evidence and derived views is grounded in modern data-intensive system design. See [kleppmann2017].

The chapter therefore separates:

- raw source evidence;
- canonical events;
- identity and ownership resolution;
- Engineering Change reconstruction;
- analytical projections;
- metric and model outputs;
- generated explanations;
- organizational decisions.

This separation supports replay, correction, reproducibility, and semantic evolution.

Event time and stream processing

Event-time semantics, late data, watermarks, and replayable stream processing are grounded in [akidau2018].

These principles support the platform’s treatment of:

- source event time;
- ingestion and processing time;
- delayed deployment and incident evidence;
- outcome maturity;
- retractions and recomputation;
- shadow processing during semantic changes.

A current dashboard value is not necessarily a final analytical result when important downstream evidence has not matured.

Provenance

The W3C PROV model provides a general foundation for representing entities, activities, agents, derivation, generation, attribution, and association. See [moreau2013].

The platform applies provenance across the complete evidence path, including source records, reconstruction, metrics, models, experiments, explanations, and decisions.

The central requirement is that observed evidence, inferred relationships, predictions, causal estimates, generated narratives, and human decisions remain distinguishable.

Polyglot analytical architecture

Different workloads require different storage and serving patterns.

The chapter's use of immutable evidence stores, operational state, analytical warehouses, graph projections, search indexes, feature stores, and governed registries follows the tradeoff-oriented approach in [kleppmann2017].

Polyglot storage should not produce polyglot truth. Authoritative semantics, identifiers, lineage, and versions must remain shared.

Service reliability

The platform is a production service whose failures can affect consequential organizational decisions.

SRE principles concerning service-level objectives, error budgets, operational toil, monitoring, incident response, and reliable change are grounded in [beyer2016] and [beyer2018].

Relevant objectives include:

- connector freshness and completeness;
- schema-conformance rate;
- identity-resolution quality;
- linkage quality;
- metric publication latency;
- replay and correction latency;
- serving availability;
- audit-log completeness.

The chapter extends SRE thinking from infrastructure availability to evidence integrity.

Privacy engineering

The NIST Privacy Framework provides the basis for identifying, governing, controlling, communicating, and protecting privacy risk. See [nist2020-privacy].

The platform applies these principles through:

- purpose limitation;
- data minimization;
- pseudonymization;
- re-identification isolation;
- aggregation thresholds;
- retention and deletion controls;
- practitioner correction and appeal;
- access auditing.

The platform should collect the least detailed evidence necessary for the declared decision.

Cybersecurity governance

The NIST Cybersecurity Framework provides a general foundation for governing, identifying, protecting, detecting, responding to, and recovering from cybersecurity risk. See [nist2024-csf].

Relevant controls include:

- least-privilege connectors;
- secrets management;
- encryption;
- workload and tenant isolation;
- supply-chain review;
- immutable audit evidence;
- incident response;
- tested backup and recovery;
- vendor-risk management.

Engineering Intelligence data can reveal organizational structure, vulnerabilities, code activity, and sensitive initiatives, making it a high-value security target.

AI governance

AI-enabled explanations, classifications, recommendations, and agents introduce additional risks.

The chapter's governance approach is informed by [nist2023-airmf], [nist2024-genai], 001, and 894.

These foundations support:

- model and prompt registries;
- treatment and configuration versioning;
- evaluation and monitoring;
- human accountability;
- bounded permissions;
- rollback;
- vendor and data-use controls;
- generated-content provenance.

The standards guide risk management; they do not establish that any individual model is effective or safe for a particular engineering decision.

Registries as institutional memory

The platform includes registries for metrics, models, experiments, evidence, decisions, and governed uses.

This design supports reproducibility, review, and organizational learning by preserving:

- definitions;
- owners;
- versions;
- assumptions;
- evidence;
- limitations;
- outcomes;
- review dates;
- correction history.

The broader organizational-learning foundations are discussed in [argyris1978] and [senge1990].

Decision products versus dashboards

The platform is designed to serve bounded decision products rather than unrestricted metric browsing.

A decision product should include:

- the decision question;
- eligible population;
- evidence and quality state;
- metrics and uncertainty;
- causal status;
- economic interpretation;
- guardrails;
- assumptions and limitations;
- recommended review date.

This use-specific approach follows the validity principles in [messick1995] and reduces the indicator misuse described in

[campbell1979] and [goodhart1984].

Purpose-bound serving

Role-based access is combined with declared purpose, population, data classification, and use restrictions.

This is an original Engineering Intelligence extension aligned with the privacy-risk principles in [nist2020-privacy].

Examples include:

- pseudonymous analysis for measurement scientists;
- aggregated team views for managers;
- raw connector diagnostics for platform operators;
- restricted re-identification for correction or authorized investigation;
- prohibition of unsupported individual employment use.

Reproducibility and correction

A decision-grade platform must reproduce the evidence path behind a result.

This requires:

- source and schema versions;
- transformation lineage;
- identity and reconstruction versions;
- metric and model versions;
- experiment assignment and exposure;
- correction history;
- generated artifacts and publication manifests.

The chapter's correction model requires replay and downstream impact analysis when evidence or semantics change.

This follows the provenance and replay foundations in [moreau2013], [kleppmann2017], and [akidau2018].

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- a platform centered on Engineering Change reconstruction rather than source-specific dashboards;
- explicit separation of evidence, reconstruction, inference, prediction, causal estimate, explanation, and decision;
- connected metric, model, experiment, evidence, decision, and governed-use registries;
- purpose-bound serving products;
- semantic and evidence-quality SLOs;
- correction-triggered decision-impact review;
- outcome-maturity state;
- confidence-aware graph relationships;
- machine-participation and delegation provenance;

- a decision-first platform roadmap tied to realized value rather than connector count.

These contributions should be evaluated through reliability, correction accuracy, practitioner trust, reproducibility, governance outcomes, and the quality of decisions the platform improves.

Chapter 15. The Engineering Metrics Catalogue



Engineering Intelligence requires a governed catalogue of metrics whose definitions, purposes, *formulas, instrumentation, biases, and actions are explicit. This chapter provides a practical reference model for defining and operating metrics across flow, quality, economics, human attention, AI participation, architecture, and experimentation. The objective is not to maximize metric count, but to create a coherent set of measures that support decisions and remain reproducible over time.*

15.1. Why a catalogue matters

Metrics are often created as local queries, embedded dashboard formulas, or spreadsheet calculations. Over time, several definitions of the same concept emerge. "Cycle time" may mean commit to merge, pull-request open to merge, work start to deployment, or intent to accepted outcome. All can be valid, but they are not interchangeable.

A metric catalogue establishes one governed location for:

- semantic definition;
- decision purpose;
- unit of analysis;
- formula and exclusions;
- source events and lineage;

- model and metric version;
- uncertainty and data quality;
- known biases and gaming risks;
- intended actions and guardrails.

Without a catalogue, arguments about performance become arguments about definitions, and the same word travels between teams carrying incompatible meanings. Catalogue governance is therefore not an infrastructure concern but a decision-quality concern: it fixes the vocabulary in which engineering claims are made, disputed, and settled. The remainder of this chapter treats each metric as an entry in that catalogue rather than a chart on a dashboard.

15.2. Metric specification template

Every metric should contain the following fields:

Field	Description
Identifier	Stable machine-readable name
Title	Human-readable name
Version	Semantic version of definition and computation
Construct	Concept the metric is intended to measure
Decision	Decision or question the metric supports
Unit of analysis	Engineering Change, pull request, repository, service, team, portfolio, or time interval
Population	Eligible entities and exclusion rules
Formula	Exact mathematical or algorithmic definition
Sources	Events, entities, features, and models required
Latency	Expected time from source event to availability
Uncertainty	Confidence, credible interval, or quality state
Interpretation	What an increase or decrease may mean
Failure modes	Biases, confounders, gaming, and invalid uses
Guardrails	Outcomes that must remain within tolerance
Owner	Accountable group for maintenance and review

15.3. Flow metrics

15.3.1. Engineering Change lead time

Definition: elapsed time from accepted intent to accepted outcome.

$$LT(e) = t_{accepted\ outcome}(e) - t_{accepted\ intent}(e)$$

Purpose: portfolio planning, delay analysis, and end-to-end flow optimization.

Instrumentation: planning events, Engineering Change lifecycle, deployment and outcome evidence.

Interpretation: lower values may indicate faster delivery, simpler work, smaller scope, or weaker verification. Always stratify by complexity and outcome class.

15.3.2. Active work time

Definition: inferred or observed duration during which human or AI participants actively worked on the change.

Purpose: distinguish execution effort from waiting.

Failure modes: passive telemetry may undercount offline reasoning and overcount open-but-idle sessions.

15.3.3. Queue time

Definition: duration in a state where progress awaits a constrained resource or policy.

$$QT(e) = \sum_{q \in Q_e} duration(q)$$

Useful subtypes include review queue, CI queue, approval queue, environment queue, and deployment queue.

15.3.4. Flow efficiency

$$FE(e) = \frac{ActiveTime(e)}{LeadTime(e)}$$

Flow efficiency is diagnostic, not an objective to maximize mechanically. Safety-critical work may rationally include waiting or observation windows.

15.3.5. Work in process

Definition: number of active Engineering Changes at a point in time or over a period.

Weighted WIP should incorporate complexity and unresolved involvement rather than treating every change as equal.

15.3.6. Aging work

Definition: age distribution of open Engineering Changes relative to expected duration for their class.

Purpose: detect stalled work before aggregate lead time worsens.

15.4. Review metrics

15.4.1. Time to first substantive review

Bots, assignment acknowledgements, and non-substantive comments should be excluded unless explicitly part of the construct.

15.4.2. Active review effort

Definition: estimated human and AI effort spent understanding, verifying, and responding to a change.

15.4.3. Review response cycles

Definition: number of meaningful author-reviewer iteration loops before acceptance.

A high value may indicate poor initial quality, unclear expectations, complex work, or effective defect discovery.

15.4.4. Review queue age

Definition: age of changes awaiting review, preferably severity- or cost-of-delay-weighted.

15.4.5. Review Friction Index

A composite diagnostic metric:

$$RFI(e) = \sum_{k} w_k z_k(e)$$

Potential components include queue age, response cycles, context recovery, unresolved discussions, and validation effort.

The index must expose component values and should be calibrated against delay and rework.

15.5. Quality metrics

15.5.1. Change failure rate

Definition: proportion of accepted outcomes that require rollback, hotfix, incident response, or material remediation within a declared window.

The remediation threshold and attribution window must be explicit.

15.5.2. Escaped defect rate

Definition: defects attributable to a change and discovered after the designated release boundary.

Use severity weighting and confidence of attribution.

15.5.3. Severity-weighted defect cost

$$DWC = \sum_i w_{severity(i)} \times count_i$$

Weights should reflect expected impact or policy, and remain versioned.

15.5.4. Rework ratio

$$RR(e) = \frac{PostAcceptanceRework(e)}{TotalEffort(e)}$$

Rework includes corrective changes linked to the original Engineering Change. Do not count planned follow-on development as rework.

15.5.5. Detection stage distribution

Shows where defects are found: authoring, AI review, human review, CI, staging, rollout, production, or customer report.

Earlier detection is often cheaper, but stage alone does not establish value without severity and effort.

15.6. Economic metrics

15.6.1. Total Engineering Change cost

$$C_{total} = C_{human} + C_{AI} + C_{tool} + C_{infra} + C_{coord} + C_{delay} + C_{quality} + C_{risk}$$

This is a model with explicit assumptions, not a directly observed number.

15.6.2. Cost per accepted outcome

$$CPO = \frac{\sum_e C_{total}(e)}{N_{accepted\ outcomes}}$$

Use only within comparable outcome classes.

15.6.3. Cost of delay

Definition: value lost because an outcome arrives later than its relevant value curve.

Report ranges and scenario assumptions rather than false precision.

15.6.4. Scarce-attention consumption

Definition: attention-weighted effort from constrained roles or expertise pools.

Purpose: identify organizational bottlenecks and evaluate routing policies.

15.6.5. Tool net value

$$TNV = Benefit_{labor} + Benefit_{quality} + Benefit_{delay} - Cost_{license} - Cost_{infra} - Cost_{validation}$$

Benefits should be causal estimates whenever possible.

15.7. Human and cognitive metrics

15.7.1. Context recovery time

Definition: time required to resume productive work after a material interruption or delay.

15.7.2. Concurrent cognitive inventory

$$CI_p(t) = \sum_{e \in Active_p(t)} w_e u_{p,e}(t)$$

Use for system research and workload design, not individual performance ranking.

15.7.3. Cognitive Offload Index

$$COI = \frac{\hat{W}_{control} - \hat{W}_{treatment}}{\hat{W}_{control}}$$

Must be paired with quality, learning, and trust guardrails.

15.7.4. Cognitive reload

Definition: additional human work caused by inspecting, correcting, rejecting, or recovering from tool or AI output.

15.7.5. Attention fragmentation

Possible measure: number and weighted severity of task transitions within active work windows.

The metric requires careful validation because transitions can also reflect healthy collaboration or deliberate batching.

15.8. AI metrics

15.8.1. Finding precision and recall

Use labeled findings and severity weighting.

15.8.2. Precision at k

Measures the proportion of relevant findings among the top k positions shown to a human.

Purpose: evaluate whether limited attention is directed to useful findings.

15.8.3. NDCG

Normalized Discounted Cumulative Gain evaluates ranking when findings have graded relevance or severity.

15.8.4. Duplicate finding rate

Definition: proportion of AI findings duplicating another AI or human finding without additional value.

15.8.5. Human validation burden

Measures inspection, investigation, correction, and recovery effort attributable to AI output.

15.8.6. Action rate

Tracks direct application, modified application, equivalent independent fix, follow-up issue, acknowledgement, rejection, and no response.

15.8.7. AI contribution latency

Time from change state to useful AI contribution. Late findings may have low value even if correct.

15.8.8. AI Leverage Factor

A decision model relating value and avoided human or quality cost to AI and validation cost.

15.9. Architecture and topology metrics

15.9.1. Repository spread

Number of repositories participating in one Engineering Change.

Interpret as topology, not intrinsic effort.

15.9.2. Dependency depth

Longest dependency path among affected components.

15.9.3. Critical-path duration

Longest duration-weighted path through the Engineering Change dependency graph.

15.9.4. Ownership boundary count

Number of distinct ownership or approval boundaries crossed.

15.9.5. Blast Radius Score

Composite of users, systems, data, financial exposure, availability, compliance, and reversibility.

15.9.6. Change surface to impact surface ratio

$$SIR(e) = \frac{ImpactSurface(e)}{ChangeSurface(e)}$$

A high ratio can identify small changes with disproportionate reach.

15.10. Experiment metrics

15.10.1. Treatment exposure rate

$$ExposureRate = \frac{ExposedEligibleUnits}{AssignedTreatmentUnits}$$

15.10.2. Compliance rate

Measures adherence to the assigned workflow or policy.

15.10.3. Average treatment effect

$$ATE = E[Y(1) - Y(0)]$$

15.10.4. Minimum practical effect attainment

Probability or confidence that the effect exceeds a predeclared practical threshold.

15.10.5. Guardrail breach rate

Frequency and severity with which experiment guardrails exceed tolerance.

15.10.6. Experiment decision latency

Time from experiment registration to evidence-backed decision.

15.11. Data quality metrics

15.11.1. Event completeness

Proportion of expected source events received.

15.11.2. Linkage coverage

Proportion of artifacts linked to Engineering Changes above a confidence threshold.

15.11.3. Identity ambiguity rate

Proportion of events whose actor, repository, service, or team identity remains unresolved or multiply matched.

15.11.4. Freshness

Age of the newest complete data available to a metric.

15.11.5. Replay divergence

Difference between streaming results and authoritative recomputation.

15.12. Metric bundles

Metrics should often be consumed as bundles rather than isolated values.

15.12.1. AI review bundle

- human review effort;
- AI validation burden;
- severe finding recall;
- precision at k;
- duplicate rate;
- cycle time;
- escaped defects;
- tool and inference cost.

15.12.2. Flow bundle

- lead time;
- active time;
- queue time by stage;
- WIP;
- aging distribution;
- flow efficiency;
- accepted outcome rate.

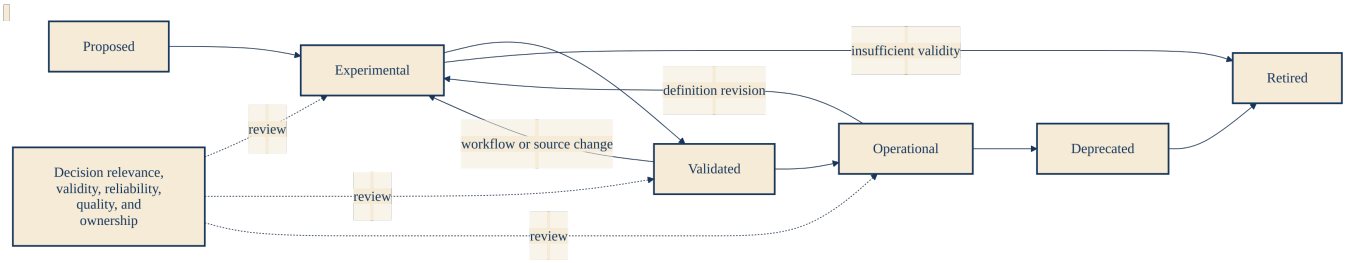
15.12.3. Platform bundle

- event freshness;
- linkage coverage;
- metric latency;
- replay divergence;
- service availability;
- source failure rate.

15.13. Metric lifecycle

A metric progresses through proposed, experimental, validated, operational, deprecated, and retired states.

A proposed metric should not appear in executive scorecards. A validated metric has evidence of construct validity and reliability. An operational metric has an owner, SLO, lineage, and action process.



15.14. Review and retirement

Metrics should be reviewed periodically for:

- continuing decision relevance;
- changed workflows or source semantics;
- gaming or behavioral adaptation;
- data quality;
- overlap with other metrics;
- evidence of action;
- cost of maintenance.

A metric that no longer informs a decision should be retired, not preserved indefinitely because it once appeared in a dashboard.

15.15. Common failure modes

15.15.1. Name without definition

Two teams report "cycle time" using different start and end points.

15.15.2. Formula without construct

A number is precisely computed but does not validly represent the concept claimed.

15.15.3. Metric without decision

Dashboards accumulate measures with no operational action.

15.15.4. Score without components

Composite metrics hide the reason for movement and invite gaming.

15.15.5. Dollar value without assumptions

Derived economics appear exact despite uncertain rates, attribution, and counterfactuals.

15.16. Worked example: reading the flow bundle without misreading it



Runnable version: `companion/notebooks/07-flow-bundle-interpretation.py`.

A platform lead is asked whether a workflow change three months ago helped delivery. She opens the flow bundle for the affected teams.

- Lead time: 6.2 days, down from 7.9 days.
- Active time: 3.1 days, down from 3.2 days.
- Queue time by stage: review queue 1.4 days (down from 2.6); CI queue 0.7 days (unchanged); staging queue 1.0 days (up from 0.9).
- WIP: 42 changes, down from 51.
- Aging distribution: 95th percentile 18 days, down from 27.
- Flow efficiency: 50% active, up from 41%.
- Accepted outcome rate: 87%, down from 91%.

Read as isolated numbers, this looks like an unambiguous improvement — every flow metric moved in the desired direction. Read as a bundle, the accepted-outcome rate falling four points is the loudest signal. Faster flow with fewer accepted outcomes is not the same intervention as faster flow with steady outcomes; the difference is whether "faster" has come from reducing wait time or from accepting lower-quality work.

Drill-down on the accepted-outcome delta shows a specific category of change — customer-facing UI updates — is now being reverted twice as often as before. The workflow change routed those changes past a specialist reviewer who had previously caught interaction-design issues. The bundle exposed the trade-off that any single flow number would have hidden. The intervention is not reversed but is amended: the routing rule adds an exception for customer-facing UI changes, restoring specialist review while preserving the queueing improvements elsewhere.

15.17. Key takeaways

- A governed metric catalogue is necessary for reproducible Engineering Intelligence.
- Every metric requires a construct, decision, unit, formula, lineage, uncertainty, owner, and failure modes.
- Flow, quality, economics, cognition, AI, architecture, experiments, and data quality should be represented together.
- Composite metrics must preserve drill-down into components.
- Derived measures require explicit models and validation.
- Metrics should have lifecycle states and retirement criteria.

15.18. Open questions

- Which metric bundles best support executive decisions without oversimplification?
- How should metric definitions be standardized across companies while preserving local context?
- What evidence threshold should move a derived metric from experimental to operational?
- How can metric catalogues encode causal assumptions explicitly?

- Which measures remain robust after becoming organizational targets?

15.19. Implementation checklist

1. Create a versioned metric specification schema.
2. Require decision, construct, unit, formula, sources, uncertainty, guardrails, and owner.
3. Build initial bundles for flow, quality, AI review, and platform health.
4. Separate experimental metrics from operational scorecards.
5. Expose component drill-down for every composite measure.
6. Publish data quality and freshness with metric values.
7. Establish periodic review and retirement.
8. Prohibit unsupported individual ranking.
9. Preserve historical versions for reproducibility.
10. Link every operational metric to an explicit action process.

15.19.1. Research foundations and source notes

The metric catalogue in this chapter draws on software-delivery research, multidimensional productivity frameworks, queueing theory, information retrieval, measurement validity, and indicator-governance research. It extends those foundations into a governed system of Engineering Intelligence measures tied to explicit decisions and Engineering Change evidence.

Delivery performance

The catalogue's delivery and reliability measures build on the research synthesized in [forsgren2018].

Measures such as deployment frequency, lead time, restoration behavior, and change-failure outcomes are useful because they describe system-level delivery capability rather than individual activity.

Their interpretation still depends on:

- service and repository boundaries;
- release and deployment definitions;
- work mix;
- risk class;
- observation window;
- outcome maturity;
- data quality.

The catalogue therefore treats delivery measures as governed metric products rather than universally comparable raw numbers.

Multidimensional productivity

The SPACE framework argues that productivity should be considered across satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow. See [forsgren2021-space].

The DevEx framework emphasizes feedback loops, cognitive load, and flow state. See [noda2023-devex].

These foundations support the catalogue's refusal to collapse engineering performance into one score.

Instead, the catalogue groups measures across:

- flow;
- quality;
- attention;
- coordination;
- architecture;
- AI participation;
- economics;
- experimentation;
- governance;
- evidence quality.

Queueing and flow

Little's Law relates average work in progress, throughput, and average flow time. See [little1961].

Kingman's work and the operations literature explain why utilization and variability can create nonlinear waiting. See [kingman1961] and [hopp2011].

These foundations support measures such as:

- work in progress;
- queue depth;
- arrival rate;
- service rate;
- service time;
- wait time;
- blocked time;
- tail latency;
- utilization of constrained resources;
- flow efficiency.

The catalogue distinguishes these measures because elapsed time alone cannot explain whether work is active, queued, blocked, or waiting for scarce attention.

Quality-adjusted flow

Faster delivery can be harmful when it increases rework, incidents, security findings, or operational burden.

The catalogue therefore pairs speed measures with guardrails such as:

- escaped defect rate;
- rollback rate;

- incident consequence;
- rework share;
- support burden;
- security and privacy findings;
- outcome durability.

This approach is consistent with the system-level delivery perspective in [forsgren2018] and the reliability principles in [beyer2016] and [beyer2018].

Precision and recall

AI review tools, static analyzers, classifiers, and retrieval systems often produce findings that can be evaluated through precision and recall. See [manning2008].

The catalogue requires explicit definitions for:

- what constitutes a finding;
- duplicate handling;
- relevance;
- severity;
- eligible population;
- adjudication;
- mature outcome window.

Without these definitions, precision and recall from different evaluations are not comparable.

Ranking metrics

When findings are ordered by importance, rank-sensitive measures become relevant.

Discounted cumulative gain and normalized discounted cumulative gain account for graded relevance and position. See [jarvelin2002].

The catalogue uses ranking metrics where reviewer attention is scarce and high-value findings should appear early.

NDCG should not be interpreted as defect prevention by itself. It measures ranking quality against a defined relevance judgment.

Validity

Messick's use-specific view of validity provides the foundation for requiring every metric to identify its intended interpretation and use. See [messick1995].

Hand's broader treatment of measurement reinforces the point that definitions and representation choices shape what can be concluded. See [hand2004].

The catalogue therefore records:

- construct;

- decision purpose;
- eligible population;
- unit of analysis;
- scale and unit;
- computation;
- quality requirements;
- limitations;
- permitted and prohibited uses.

Reliability

A metric must be sufficiently consistent for its intended use.

Reliability concerns include:

- stable source semantics;
- reproducible transformation;
- labeler agreement;
- reconstruction consistency;
- metric-version determinism;
- temporal stability where expected.

Cronbach's reliability work provides one important psychometric foundation. See [cronbach1951].

Reliability alone is insufficient: a consistently biased metric can remain reliable while invalid.

Indicator gaming

Campbell's Law and Goodhart's Law describe the risk that indicators become corrupted when used as targets. See [campbell1979] and [goodhart1984].

Catalogue measures can be gamed through:

- change slicing;
- superficial approvals;
- suppressed incident reporting;
- altered eligibility;
- trivial AI suggestions;
- exclusion of blocked work;
- shifted work outside the measured boundary.

Each high-consequence metric should therefore include a gaming analysis and balancing measures.

Denominators and exposure

Rates depend on governed denominators.

For example, an AI acceptance rate can be computed over all generated suggestions, displayed suggestions, actionable suggestions, unique suggestions, or suggestions with mature outcomes.

The catalogue requires explicit eligibility, exposure, and denominator contracts because each version answers a different question.

Distributions and tails

Engineering measures are often skewed or heavy-tailed.

The catalogue therefore favors:

- medians;
- quantiles;
- tail probabilities;
- distributions;
- subgroup views;
- uncertainty intervals;
- sample counts.

Average incident cost, review latency, or change size can be dominated by a small number of extreme observations.

Attention measures

The catalogue's attention measures draw on cognitive-load theory and interruption research. See [sweller1988] and [mark2008].

Relevant measures include:

- review and validation effort;
- context-recovery proxies;
- fragmentation;
- scarce-specialist demand;
- queue delay;
- interruption frequency;
- attention yield.

These measures are latent or model-based and should be reported with uncertainty and privacy controls.

Coordination measures

Coordination metrics are informed by Conway's Law, socio-technical congruence, and distributed-development research. See [conway1968], [cataldo2006], and [herbsleb2007].

Relevant measures include:

- team and ownership boundaries;
- handoffs;

- approval boundaries;
- synchronization points;
- owner-discovery time;
- dependency-related waiting;
- ownership entropy.

These measures should describe system structure rather than serve as team rankings.

Architecture measures

Architecture metrics are informed by information hiding and modularity in [parnas1972] and team cognitive-load concepts in [skelton2019].

The catalogue includes measures for:

- dependency depth;
- fan-in and fan-out;
- change coupling;
- cycles;
- validation surface;
- reversibility;
- blast radius;
- cognitive surface.

Structural measures should be calibrated against observed flow, effort, incidents, and rework.

AI measures

The AI metric family is informed by coding-assistant studies in [peng2023], [vaithilingam2022], and [barke2023], and repository-level agent benchmarks in [jimenez2024].

The catalogue distinguishes:

- generation;
- exposure;
- interaction;
- acceptance;
- contribution;
- validation burden;
- downstream quality;
- net attention effect;
- economic realization.

Acceptance rate alone is treated as insufficient evidence of value.

Economic measures

Economic measures draw on cost-benefit analysis in [boardman2018] and productivity-realization research in [brynjolfsson1993] and [brynjolfsson2017].

The catalogue distinguishes:

- cash savings;
- avoided cost;
- released capacity;
- modeled outcome value;
- risk reduction;
- option value;
- realized value.

This separation reduces the risk of presenting theoretical time savings as actual financial savings.

Causal status

A metric movement is not automatically a causal effect.

The catalogue labels values as:

- observed;
- reconstructed;
- inferred;
- predicted;
- experimentally estimated;
- causally estimated under declared assumptions.

The methodological foundation comes from [rubin1974], [holland1986], [pearl2009], [imbens2015], and [hernan2020].

Data-quality metrics

Metric quality depends on source and linkage quality.

The catalogue therefore includes:

- source freshness;
- completeness;
- schema conformance;
- identity-resolution rate;
- Engineering Change linkage coverage;
- confidence calibration;
- outcome maturity;
- correction rate;
- drift.

This follows the validity principle that evidence quality is part of interpretation rather than a separate implementation

concern.

Governance metrics

Governance measures track whether the measurement system is being used responsibly.

Examples include:

- percentage of metrics with owners;
- percentage with prohibited-use declarations;
- access-review completion;
- correction and appeal resolution;
- metric incidents;
- stale or deprecated definitions;
- unsupported individual-level uses;
- decision follow-up completion.

Privacy and professional principles are informed by [nist2020-privacy] and [acm2018-ethics].

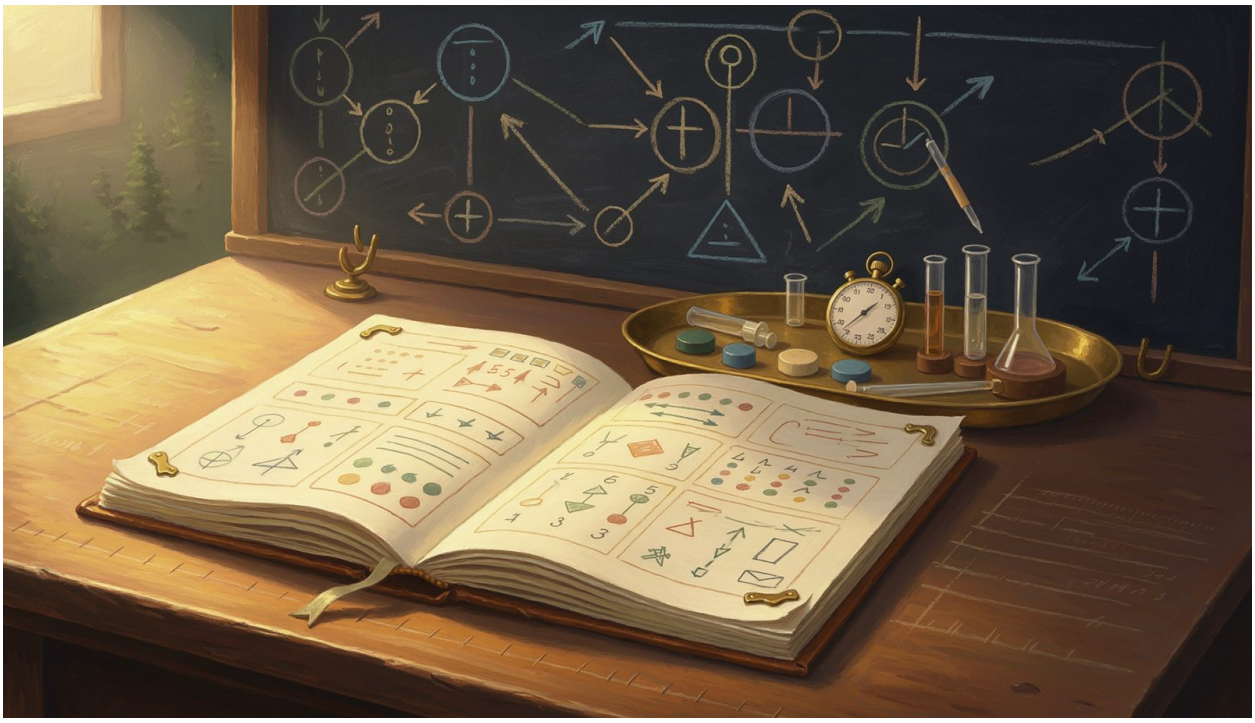
Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- the unified metric catalogue spanning flow, quality, attention, economics, AI, architecture, governance, and evidence quality;
- Engineering Change as the default aggregation unit;
- net attention effect;
- cognitive offload and reload;
- accepted-contribution share;
- validation surface and cognitive surface;
- ownership entropy;
- outcome-maturity metrics;
- reconstruction-quality metrics;
- explicit causal-status labels;
- value-realization measures linked to decision records;
- prohibited-use and quality-state fields as part of every metric definition.

These measures should be calibrated, versioned, and validated in each organizational context rather than treated as universal constants.

Chapter 16. Experiment Playbooks



The previous chapters describe how to measure engineering systems and estimate causal effects. *This chapter turns those methods into reusable experiment playbooks. Each playbook defines the intervention, mechanism, unit of assignment, outcomes, guardrails, instrumentation, external factors, analysis method, and decision rule. The examples cover AI review, merge queues, CI acceleration, review routing, pull-request slicing, repository topology, context restoration, and autonomous agents.*

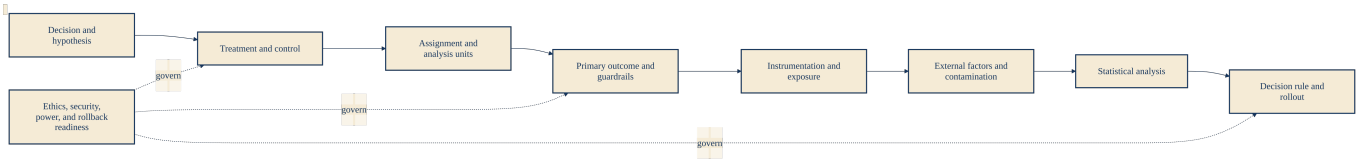
16.1. How to use a playbook

A playbook is not a fixed recipe. It is a structured starting point that forces the experiment owner to make assumptions explicit.

Every playbook contains:

- decision and hypothesis;
- treatment and control;
- assignment and analysis units;
- eligibility and exclusions;
- primary outcome;
- guardrails;

- treatment exposure;
- external factors;
- statistical design;
- minimum practical effect;
- stopping and decision rules;
- interpretation risks.



16.2. Playbook 1: AI pull-request reviewer

16.2.1. Decision

Should the organization adopt, replace, or expand an AI pull-request reviewer?

16.2.2. Mechanism

The reviewer may reduce human search and verification effort, find defects earlier, and improve consistency. It may also create false positives, duplication, validation burden, and automation bias.

16.2.3. Treatment

A precisely versioned model, prompt, repository index, severity policy, comment-ranking policy, and user interface.

16.2.4. Assignment

Engineering Change or repository-cluster assignment is preferable when reviewers and codebases are shared. Pull-request assignment is possible for low-contamination environments.

16.2.5. Primary outcomes

Recommended primary outcome:

$$QualityAdjustedReviewEffort = HumanReviewMinutes + ValidationMinutes + ExpectedFailureCostRate$$

Alternative primary outcomes include severe-finding recall at a fixed validation budget or cost per accepted Engineering Change.

16.2.6. Guardrails

- severe escaped defects;
- rollback rate;
- security findings;
- cognitive reload;

- author understanding;
- reviewer learning;
- comment latency;
- trust calibration.

16.2.7. Instrumentation

Capture AI findings, ranking, confidence, files inspected, human views, action type, duplicate status, validation effort, code changes, and downstream outcomes.

16.2.8. Analysis

Use cluster-randomized or staggered rollout with hierarchical effects by language, complexity, repository type, and finding category.

16.2.9. Decision rule

Adopt only if the minimum practical reduction in quality-adjusted effort is achieved and severe-quality guardrails remain within tolerance.

16.3. Playbook 2: ranked versus unranked AI findings

16.3.1. Hypothesis

Showing the most important findings first improves severe-finding action and reduces validation burden.

16.3.2. Treatment

Ranked top-k findings with expandable remainder.

16.3.3. Control

Chronological or unranked full finding list.

16.3.4. Primary outcomes

- human validation minutes;
- severe-finding action rate;
- NDCG or precision at k;
- time to first high-value action.

16.3.5. Risks

Ranking can hide valuable low-confidence findings. Users may inspect only visible items. Track severe recall across the full set and interface expansion behavior.

16.4. Playbook 3: merge queue introduction

16.4.1. Decision

Should a repository or portfolio adopt a merge queue?

16.4.2. Mechanism

A merge queue may reduce broken-main events, integration rework, and redundant CI, but can increase queueing latency and reduce author control.

16.4.3. Assignment

Repository cluster or switchback by time block. Avoid individual assignment because the queue is shared infrastructure.

16.4.4. Primary outcome

Quality-adjusted time from approval to integrated mainline state.

16.4.5. Guardrails

- failed mainline builds;
- rollback and revert;
- queue age;
- urgent-change latency;
- CI cost;
- author context recovery.

16.4.6. External factors

Release freezes, high-volume launches, CI outages, and repository migrations should be recorded.

16.4.7. Analysis

Switchback analysis is useful when the queue can be enabled and disabled safely. Otherwise use staggered rollout with repository-level matching.

16.5. Playbook 4: CI acceleration

16.5.1. Decision

Does a faster CI configuration create sufficient engineering value to justify infrastructure cost?

16.5.2. Treatments

Examples include larger runners, test sharding, improved caching, selective testing, speculative execution, or remote build execution.

16.5.3. Primary outcome

Quality-adjusted critical-path CI latency.

16.5.4. Economic model

$$NetValue = Value_{delay\ reduced} + HumanContextCost_{avoided} - InfrastructureCost_{added} - QualityRisk_{change}$$

16.5.5. Guardrails

- flaky-test rate;
- missed defect rate;
- cache correctness;
- resource saturation;
- queue spillover;
- downstream rework.

16.5.6. Assignment

Job, pull-request, repository, or time-block assignment depending on shared-cache and capacity interference.

16.5.7. Interpretation risk

Reducing job duration may not reduce Engineering Change lead time if review or deployment remains the true constraint.

16.6. Playbook 5: risk-based specialist review

16.6.1. Decision

Can scarce specialist attention be routed only to changes likely to benefit from it?

16.6.2. Treatment

A risk model routes high-risk changes to specialist review and lower-risk changes to standard review plus automated controls.

16.6.3. Control

Universal specialist review or the current policy.

16.6.4. Primary outcome

Quality-adjusted release time and specialist attention consumed.

16.6.5. Guardrails

- severe false negatives;

- migration, security, or compliance incidents;
- rollback;
- model calibration by subgroup;
- unresolved author uncertainty.

16.6.6. Design

Randomize within a safe risk band. For policy thresholds, use regression discontinuity around the cutoff. Retain mandatory review for categories where experimentation is inappropriate.

16.7. Playbook 6: pull-request size guidance

16.7.1. Decision

Does encouraging smaller pull requests improve flow and review quality?

16.7.2. Mechanism

Smaller changes may reduce review burden and integration risk, but excessive fragmentation can increase coordination, sequencing, and context switching.

16.7.3. Treatment

Guidance, tooling, or workflow that encourages a target size or stacked changes.

16.7.4. Unit of analysis

Engineering Change, not individual pull request. Otherwise one intent split into ten pull requests can appear artificially productive.

16.7.5. Primary outcomes

- Engineering Change lead time;
- total human review effort;
- coordination and handoff count;
- rework and defect outcomes;
- critical-path duration.

16.7.6. Heterogeneous effects

Estimate by repository topology, deployment coupling, team familiarity, and change complexity.

16.7.7. Decision rule

Do not adopt a universal line-count threshold. Identify ranges and contexts where smaller slices improve total outcomes.

16.8. Playbook 7: monorepo or repository consolidation

16.8.1. Decision

Would consolidating repositories reduce coordination and build friction enough to justify migration cost and new coupling?

16.8.2. Intervention

A selected portfolio is migrated or reorganized into a shared repository, build system, or release structure.

16.8.3. Design

Randomization is rarely feasible. Use matched comparison groups, synthetic controls, and interrupted time series with long pre- and post-periods.

16.8.4. Primary outcomes

- Engineering Change repository spread;
- coordination cost;
- critical-path duration;
- build latency;
- ownership boundary count;
- defect and rollback outcomes.

16.8.5. Guardrails

- build-system complexity;
- access-control limitations;
- repository-scale performance;
- blast radius;
- team autonomy;
- migration disruption.

16.8.6. External factors

Reorganizations and platform migrations are likely confounders and must be recorded explicitly.

16.9. Playbook 8: automated context restoration

16.9.1. Hypothesis

An AI-generated resumption package reduces the cognitive recovery required after a change has been inactive.

16.9.2. Treatment

On return, the author receives a structured summary of intent, files changed, unresolved comments, decisions, CI status, dependent changes, and modifications since last activity.

16.9.3. Control

Standard pull-request and issue interface.

16.9.4. Eligibility

Changes inactive beyond a declared period and later resumed by the same or a new engineer.

16.9.5. Primary outcome

Time to first substantive productive action after resumption.

16.9.6. Guardrails

- incorrect-summary rate;
- missed unresolved issue;
- downstream rework;
- over-reliance;
- validation time.

16.9.7. Instrumentation

Capture resumption trigger, summary generation, views, navigation, first productive action, and corrective behavior.

16.10. Playbook 9: human-first versus AI-first review

16.10.1. Decision

Should AI findings be shown before or after an independent human review?

16.10.2. Mechanisms

AI-first may reduce human effort and direct attention. Human-first may reduce automation bias and provide a cleaner measure of AI incremental value.

16.10.3. Treatment arms

1. AI-first review.
2. Human-first review with AI revealed afterward.
3. Parallel independent review.

16.10.4. Outcomes

- independent human defect discovery;
- total combined recall;
- duplication;
- validation effort;
- severe escapes;
- review latency;
- learning effects.

16.10.5. Design

Randomize Engineering Changes, stratified by complexity and reviewer expertise. Measure carryover as reviewers learn from repeated AI exposure.

16.11. Playbook 10: autonomous maintenance agent

16.11.1. Decision

Can an agent safely complete bounded maintenance tasks with less human effort?

16.11.2. Eligible tasks

Dependency upgrades, mechanical migrations, formatting, test generation, or configuration updates with deterministic validation and easy rollback.

16.11.3. Treatment

Agent plans, edits, tests, and opens a pull request under constrained permissions.

16.11.4. Control

Human-executed task with equivalent acceptance criteria.

16.11.5. Primary outcome

Accepted tasks per unit of quality-adjusted human attention.

16.11.6. Guardrails

- unrelated-file modifications;
- failed or incomplete tests;
- security or license violations;
- rollback;
- hidden side effects;
- escalation rate;

- reviewer validation burden.

16.11.7. Attribution

Measure full agent runtime and infrastructure cost, human supervision, corrections, and downstream rework.

16.12. Playbook 11: branch-protection policy

16.12.1. Decision

Does requiring additional approvals or checks reduce expected failure loss enough to justify delay?

16.12.2. Intervention

Add or remove one policy element at a time where feasible: approval count, code-owner review, test suite, security scan, or signed commit.

16.12.3. Primary outcome

$$NetPolicyValue = ExpectedFailureLoss_{avoided} - HumanCost - DelayCost - InfrastructureCost$$

16.12.4. Design

Repository-cluster randomization for low-risk populations, regression discontinuity for risk-based policy, or matched staggered rollout.

16.12.5. Guardrails

Severe incidents and compliance constraints may impose non-negotiable minima.

16.13. Playbook 12: review-queue prioritization

16.13.1. Hypothesis

Prioritizing review requests by expected cost of delay and aging improves portfolio value without starving routine work.

16.13.2. Treatment

A priority queue using cost-of-delay class, age, risk, and specialist requirements.

16.13.3. Control

FIFO, manual selection, or current practice.

16.13.4. Outcomes

- cost-of-delay-weighted queue time;
- tail latency;
- starvation rate;
- reviewer utilization;
- severe-quality outcomes;
- cognitive switching.

16.13.5. Design

Switchback by reviewer pool or time block. Account for strategic behavior in priority labels.

16.14. External-factor control checklist

For every playbook, record:

- release and planning cycles;
- holidays and staffing changes;
- incidents and outages;
- reorganizations;
- new tools or model releases;
- repository migrations;
- policy and compliance changes;
- changes in work mix;
- vendor or platform failures.

External factors should be scoped to affected entities and time windows rather than recorded as global notes.

16.15. Pre-experiment readiness

Do not launch until:

1. treatment and control are technically stable;
2. assignment and exposure are instrumented;
3. primary outcomes and guardrails are available;
4. baseline variance and power are understood;
5. security, legal, and ethical reviews are complete;
6. stopping and rollback rules are defined;
7. concurrent interventions are registered.

Skipping any of these produces a result that reads like evidence but decays under review. Treatment instability confounds effects with rollout artifacts; missing exposure instrumentation forces an intent-to-treat interpretation onto a study intended to measure engagement; ambiguous stopping rules invite peeking. The cost of a delayed launch is almost always smaller than the cost of a study that has to be redone once these gaps are discovered mid-analysis.

16.16. Post-experiment decision record

A decision record should include:

- estimated causal effects and intervals;
- practical significance;
- guardrail status;
- heterogeneous effects;
- external-factor sensitivity;
- economic interpretation;
- limitations;
- adopt, revise, reject, or extend decision;
- rollout and monitoring plan.

The result of a good experiment is organizational knowledge, even when the intervention fails.

16.17. Common failure modes

16.17.1. Evaluating the wrong unit

Pull-request metrics are used for interventions whose effects occur across the Engineering Change.

16.17.2. Changing several mechanisms at once

A new model, prompt, UI, and policy are introduced together, preventing attribution.

16.17.3. Missing exposure

Assignment is recorded, but actual use is unknown.

16.17.4. Short observation windows

Immediate speed is measured while quality, learning, and operational effects arrive later.

16.17.5. Ignoring negative or subgroup effects

A positive average hides harm in high-risk repositories or novice populations.

16.17.6. No decision threshold

The experiment reaches significance but the improvement is too small to justify migration.

16.18. Worked example: applying the AI-review playbook



Runnable versions of this example live in the companion notebooks: * `companion/notebooks/04-causal-did-cace.py` * `companion/notebooks/05-power-and-mde.py`

A company is considering rolling out an AI code reviewer across its 180 backend teams. Rather than negotiate the rollout ad-hoc, the platform team applies the AI-review playbook from this chapter.

The playbook fixes the intervention: vendor *V*, model *M* at version 2024-08, prompt template *P* at revision 12, findings are advisory and appear within 90 seconds of a pull request being opened. The unit of assignment is the team; the unit of analysis is the Engineering Change. Randomisation is stratified by baseline review latency and repository language to reduce baseline imbalance.

The primary outcome is quality-adjusted human review effort per Engineering Change: measured review-time minutes weighted by a downstream-defect-cost multiplier over the following 45 days. Guardrails include escaped-defect rate, rollback rate, cognitive-reload complaints, and reviewer disengagement.

The playbook's power-planning stage sets the minimum detectable effect at 8% of baseline effort. Baseline standard deviation from historical data is $\sigma = 42$ minutes per change; the required sample size per arm is:

$$n = 2 \times \left(\frac{(1.96 + 0.84)\sigma}{0.08 \bar{Y}} \right)^2 \approx 1{,}950 \text{ changes per arm}$$

At the team's observed change velocity that requires roughly 11 weeks. Stopping rules use the O'Brien-Fleming boundary at three interim looks; results are analyzed by intention-to-treat with a secondary complier-average estimate.

Outcome: the intervention reduces effort by 6.4% (95% CI: 2.1% to 10.7%) with escaped-defect rate unchanged (+0.02pp, non-significant). The point estimate is below the pre-declared 8% MDE; the decision record notes the effect is real but too small to justify the licensing cost at current model prices, and the playbook is re-run six months later after a model version change.

16.19. Key takeaways

- Experiment playbooks make mechanisms, assumptions, and decision rules explicit.
- The Engineering Change is usually the correct analysis unit for workflow interventions.
- Primary outcomes should represent quality-adjusted system value, not local activity.
- Treatment exposure, external factors, guardrails, and heterogeneous effects are essential.
- Every experiment should end with a durable decision and learning record.

16.20. Open questions

- Which engineering interventions are most amenable to safe randomization?
- How should long-term learning and skill outcomes be incorporated into short operational experiments?
- What methods best evaluate rare severe failures during limited rollouts?
- How can experiment playbooks be standardized without encouraging mechanical use?
- Which interventions produce the largest value per unit of experimentation effort?

16.21. Implementation checklist

1. Select experiments tied to material decisions and plausible mechanisms.
2. Use the playbook template to register treatment, units, outcomes, guardrails, and external factors.
3. Prefer quality-adjusted Engineering Change outcomes.

4. Instrument assignment and actual exposure independently.
5. Predeclare practical thresholds and stopping rules.
6. Analyze heterogeneous effects and contamination.
7. Preserve economic assumptions and uncertainty.
8. Record decisions and rollout actions.
9. Reuse successful designs while recalibrating for local context.
10. Treat failed interventions as valuable evidence.

16.21.1. Research foundations and source notes

The experiment playbooks in this chapter operationalize the causal-inference foundations introduced earlier. They draw on randomized experiments, quasi-experimental design, online experimentation, preregistration, information retrieval, and reproducibility.

Randomized evaluation

The potential-outcomes tradition provides the core framework for estimating causal effects. See [rubin1974], [holland1986], [imbens2015], and [hernan2020].

The playbooks therefore require:

- a declared estimand;
- an explicit assignment unit;
- a credible comparator;
- auditable randomization;
- exposure and compliance tracking;
- analysis consistent with the assignment design.

Randomization improves comparability in expectation, but it does not remove problems caused by missing outcomes, spillover, treatment drift, or poor measurement.

Practical online experimentation

The operating discipline for trustworthy experiments is informed by [kohavi2020].

This supports playbook requirements for:

- sample-ratio and assignment checks;
- primary outcomes and guardrails;
- data-quality monitoring;
- predeclared stopping rules;
- treatment-version control;
- reproducible analysis;
- decision follow-up.

Engineering experiments should not rely on a dashboard trend and a post hoc explanation.

Quasi-experimental alternatives

When randomization is infeasible, the playbooks use methods such as:

- difference-in-differences;
- regression discontinuity;
- interrupted time series;
- matched comparison groups;
- synthetic controls;
- instrumental variables.

The methodological foundations are described in [shadish2002], [angrist2009], [imbens2015], and [hernan2020].

Each method relies on assumptions that must be made explicit and tested where possible. A before-and-after comparison alone is not a causal design.

Preregistration

Preregistration helps distinguish confirmatory analysis from exploration. See [nosek2018].

The playbooks therefore record before analysis:

- hypotheses;
- estimands;
- outcomes and guardrails;
- eligibility;
- exclusions;
- covariates;
- missing-data handling;
- subgroup analysis;
- stopping and decision rules.

Exploratory work remains valuable, but it should be labeled as such.

Power and decision relevance

Power analysis should be based on the smallest effect that would change the decision.

Engineering studies often require simulation because outcomes are clustered, skewed, rare, delayed, or heterogeneous.

A study that is underpowered for a decision-relevant effect should not interpret a nonsignificant result as proof of no effect.

Cluster and switchback designs

Teams, repositories, reviewers, and time windows may be more appropriate assignment units than individuals.

Clustered designs must account for within-cluster correlation. Switchback designs must account for:

- time effects;
- carryover;
- washout;
- shared capacity;
- assignment-window integrity.

These design concerns follow the general experimental principles in [shadish2002], [imbens2015], and [kohavi2020].

Noninferiority and equivalence

Some playbooks ask whether a cheaper or faster process is not unacceptably worse than the current process.

That question requires a predeclared noninferiority margin rather than a failed superiority test.

Equivalence designs are appropriate when the decision depends on demonstrating that differences lie within acceptable positive and negative bounds.

AI review and finding evaluation

AI review experiments use precision and recall as defined in standard information retrieval. See [manning2008].

Rank-sensitive evaluation can use NDCG when high-value findings should appear early. See [jarvelin2002].

The playbooks also require:

- duplicate handling;
- severity and partial relevance;
- adjudication procedures;
- reviewer disagreement analysis;
- false-positive handling effort;
- downstream defect and rework evidence.

Finding quality is not equivalent to product value, so these metrics remain mechanisms within a broader outcome bundle.

AI coding-assistant playbooks

The coding-assistant playbooks are informed by empirical studies in [peng2023], [vaithilingam2022], and [barke2023].

These studies support measuring more than completion speed. Relevant outcomes include:

- validation effort;
- correctness;
- comprehension;
- rework;
- downstream quality;
- net attention effect;
- long-term maintainability.

Voluntary adoption should be treated as a source of selection bias.

Repository-level agent evaluation

Agent playbooks use repository-level benchmark evidence such as [jimenez2024] as one capability layer.

Production readiness requires additional evidence for:

- permission boundaries;
- tool-call behavior;
- human oversight;
- rollback;
- operational consequence;
- cost;
- long-term maintainability.

Benchmark success is not sufficient evidence for autonomous production authority.

Architecture and workflow interventions

Architecture, review-policy, CI, and platform interventions are often evaluated through phased or quasi-experimental designs.

Relevant methods include:

- staggered rollout;
- matched repositories;
- interrupted time series;
- difference-in-differences;
- synthetic controls.

The playbooks require mechanism measures and guardrails because an intervention can improve one stage while moving delay or risk elsewhere.

Reproducibility

The reproducibility concerns illustrated by [open-science2015] and the FAIR principles in [wilkinson2016] support retaining:

- study protocol;
- assignment table;
- treatment configuration;
- source snapshots or durable references;
- metric versions;
- analysis code;
- generated outputs;
- deviations and corrections.

A study result should be reproducible even after dashboards, models, or organizational structures change.

Governance and ethics

Experiments involving detailed practitioner data or AI systems require purpose limitation, privacy controls, and professional accountability.

Relevant foundations include [nist2020-privacy], [nist2023-airmf], [nist2024-genai], and [acm2018-ethics].

The playbooks therefore require risk review, minimum necessary data, transparent treatment, and explicit restrictions on unsupported individual employment use.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- reusable experiment playbooks organized around Engineering Change outcomes;
- explicit linkage among assignment, exposure, human disposition, and downstream consequence;
- machine-treatment provenance across model, prompt, retrieval, tools, and configuration;
- experiment-readiness gates tied to data-quality and governance floors;
- attention and economic outcomes as first-class experimental measures;
- treatment-version drift controls;
- value-realization follow-up after study completion;
- linked study, evidence, metric, and decision records;
- replication across Engineering Change classes before enterprise-wide policy.

These playbooks should evolve as evidence accumulates and should preserve null, negative, and contradictory findings rather than only successful interventions.

Chapter 17. Governance, Ethics, and Organizational Adoption



Engineering Intelligence can improve decisions, reduce wasted effort, and make hidden system *constraints visible. It can also become a surveillance system, distort incentives, and undermine trust if metrics are misused. This chapter defines the governance, ethical, privacy, and organizational practices required for responsible adoption. It explains purpose limitation, metric rights, acceptable and prohibited uses, incentive effects, model governance, transparency, employee participation, and the operating model needed to sustain legitimate measurement.*

17.1. Measurement changes behavior

The act of measurement is not neutral. Once engineers know that a metric exists, they may consciously or unconsciously alter behavior.

Examples include:

- splitting pull requests to improve apparent throughput;
- delaying issue creation to shorten measured lead time;
- avoiding complex work that harms personal metrics;
- accepting AI suggestions to improve an adoption score;
- reducing review comments to appear efficient;
- moving coordination into uninstrumented channels;

- optimizing for deployment count rather than customer value.

This is not evidence that engineers are acting improperly. It is a predictable response to incentives and measurement pressure.

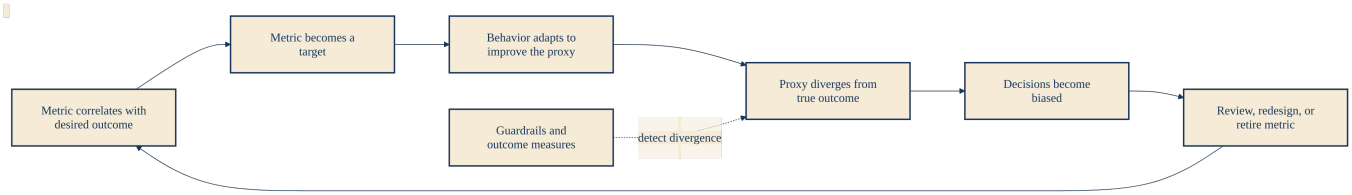
Engineering Intelligence must therefore treat behavioral adaptation as part of the system being measured.

17.2. Goodhart’s Law and metric corruption

A familiar principle states that when a measure becomes a target, it often ceases to be a good measure.

The mechanism is straightforward:

1. A metric is correlated with a desired outcome.
2. The organization rewards the metric directly.
3. People optimize the observable proxy.
4. The relationship between the proxy and outcome weakens.
5. The metric becomes misleading.



The appropriate response is not to abandon measurement. It is to design metrics for decisions, use bundles and guardrails, rotate or retire corrupted measures, and avoid turning diagnostic telemetry into individual quotas.

17.3. Foundational governance principles

A responsible Engineering Intelligence program should adopt explicit principles.

17.3.1. Purpose limitation

Data should be collected and used only for declared engineering decisions, research, safety, or operational purposes.

17.3.2. Data minimization

Collect the least granular data capable of answering the question. Prefer workflow and artifact events over keystrokes, screen capture, or continuous activity monitoring.

17.3.3. Proportionality

The intrusiveness, retention, and precision of measurement should be proportional to the decision’s importance and risk.

17.3.4. Transparency

People affected by measurement should know what is collected, how it is derived, who can access it, and what decisions it

may influence.

17.3.5. Contestability

Users should be able to challenge incorrect identities, inferred links, classifications, and derived conclusions.

17.3.6. Separation of purposes

System-improvement telemetry should not silently become individual performance data.

17.3.7. Human accountability

Models may inform decisions, but accountable human roles remain responsible for policy, employment, safety, and accepted engineering outcomes.

17.4. Acceptable and prohibited uses

Use class	Examples
Generally appropriate	Finding queue bottlenecks, evaluating tools, measuring CI reliability, identifying coordination boundaries, routing specialist review, estimating portfolio cost
Requires heightened governance	Team-level comparisons, workload and cognitive-burden studies, risk prediction, access to named collaboration graphs, AI trust and behavior analysis
Generally inappropriate	Ranking engineers by commits, lines changed, pull requests, review speed, AI acceptance, or inferred cognitive load
Prohibited without exceptional legal and ethical basis	Covert surveillance, screen or keystroke monitoring for productivity scoring, automated employment decisions from engineering telemetry, punitive use of experimental metrics

The governance body should publish these boundaries before broad instrumentation begins.

17.5. Individual-level measurement

Some events must retain actor identity to reconstruct Engineering Changes, permissions, handoffs, and specialist constraints. That does not justify exposing named individual scorecards.

Recommended controls include:

- pseudonymous identifiers in analytical datasets;
- team or role aggregation by default;
- minimum group-size thresholds;
- restricted named access for operational need;
- audit logs for identity lookups;

- prohibition of direct individual productivity ranking;
- explicit review before any new individual-level use.

Individual data may be appropriate for an engineer's private self-view or for correcting attribution, provided it is not silently repurposed.

17.6. Metric rights

Organizations should define practical rights for people represented in the data:

- the right to know what is measured;
- the right to understand metric definitions;
- the right to see relevant personal or team data where legally and operationally appropriate;
- the right to correct factual errors;
- the right to challenge derived classifications;
- the right to know whether data affects consequential decisions;
- the right to understand retention and access rules.

These rights improve trust and data quality.

17.7. Privacy by architecture

Privacy should be implemented structurally rather than through policy alone.

Design patterns include:

- pseudonymization before analytical processing;
- separate identity vaults;
- row- and column-level authorization;
- aggregation before broad access;
- short retention for detailed behavioral events;
- irreversible aggregation for long-term trends;
- differential access by purpose;
- deletion propagation across raw, graph, feature, and metric stores;
- privacy-preserving experiment analysis where practical.

A dashboard cannot leak data that the underlying service never exposes.

17.8. Data retention

Retention should reflect purpose and sensitivity.

Detailed review-interaction events may be useful for a limited experiment but unnecessary after aggregated results are validated. Incident and compliance evidence may require longer retention.

Every data class should specify:

- purpose;

- retention period;
- legal basis where applicable;
- deletion process;
- archival rules;
- access level;
- downstream derived artifacts.

17.9. Model governance

Models used for reconstruction, risk, complexity, cognitive burden, or routing require lifecycle governance.

A model record should include:

- owner and accountable approver;
- intended and prohibited uses;
- training and evaluation data;
- features and sensitive attributes;
- performance by relevant subgroup;
- calibration and uncertainty;
- drift monitoring;
- human override;
- rollback and retirement criteria;
- version history.

Models that affect review requirements, permissions, or workload should receive greater scrutiny than models used only for exploratory analysis.

17.10. Fairness and disparate impact

Engineering populations differ by role, tenure, location, repository assignment, accessibility needs, and work type. Metrics can create unequal effects even without using protected attributes directly.

Examples include:

- teams owning legacy systems appearing slower;
- new engineers receiving lower familiarity scores;
- specialists appearing to create delays because they serve as organizational bottlenecks;
- accessibility-related workflows producing different interaction patterns;
- globally distributed teams showing longer calendar latency.

Fairness analysis should examine error rates, policy effects, and burden distribution across relevant populations. The goal is not equal raw metrics; it is avoiding unjustified harm from invalid comparisons or routing.

17.11. Incentive-aware metric design

Before operationalizing a metric, ask:

- How could a rational person improve this number without improving the outcome?
- Which work becomes invisible?
- Who bears the cost of optimization?
- What behavior will be discouraged?
- What guardrail detects distortion?
- When should the metric be retired?

A metric's gaming surface is part of its specification.

17.12. Governance bodies

A mature program usually requires several forms of oversight.

17.12.1. Metric council

Approves high-impact metric definitions, versions, scorecard use, and retirement.

17.12.2. Experiment review

Reviews assignment, ethics, guardrails, security, and decision criteria for material interventions.

17.12.3. Data governance

Owns classification, retention, identity access, lineage, and deletion.

17.12.4. AI and model governance

Approves models, prompts, autonomy tiers, and high-risk routing policies.

17.12.5. Employee or practitioner representation

Includes engineers and affected practitioners in policy design and review.

These bodies may be combined, but accountability should remain explicit.

17.13. Organizational adoption

A technically strong platform can fail if introduced as a central surveillance program.

A healthier adoption sequence is:

1. begin with visible system problems engineers already experience;
2. publish principles and prohibited uses;
3. deliver local value such as queue visibility and CI diagnosis;
4. involve teams in metric definitions;
5. show uncertainty and data quality openly;
6. allow corrections;

7. evaluate interventions rather than rank teams;
8. expand only after trust and usefulness are demonstrated.

17.14. Trust as a system dependency

Trust is not a communications layer added after implementation. It determines whether people use tools, correct data, participate in experiments, and believe conclusions.

Trust increases when the system is accurate, transparent, useful, corrigible, and restrained. It decreases when metrics are surprising, opaque, punitive, or obviously incomplete.

A useful trust model includes:

$$Trust = f(Accuracy, Transparency, Benefit, Control, Fairness, Restraint)$$

This is a conceptual model rather than a score to optimize mechanically.

17.15. Communication patterns

Good communication should explain:

- why the system exists;
- what decisions it supports;
- what it does not measure;
- what uses are prohibited;
- how data is protected;
- how people can correct errors;
- what evidence led to a decision;
- how metrics and models change over time.

Avoid claims that the platform measures "developer productivity" as one objective quantity.

17.16. Governance for experiments

Experiments involving engineering workflows should consider:

- informed organizational notice;
- whether individual consent is required or practical;
- risk and reversibility;
- unequal burden across treatment groups;
- collection of new behavioral telemetry;
- employment or performance implications;
- security and production impact;
- post-experiment data retention.

Low-risk interface experiments differ materially from experiments that alter security review or collect detailed cognitive proxies.

17.17. Governance for AI agents

Agentic systems introduce additional questions:

- What permissions can the agent receive?
- Which actions require human approval?
- Who is accountable for side effects?
- What evidence must the agent provide?
- How is untrusted repository content handled?
- What triggers automatic shutdown or escalation?
- Can the organization reproduce the agent's actions?

Autonomy should increase only with evidence, bounded scope, and reversible controls.

17.18. Handling metric disputes

Disagreement should be expected. A dispute process should support:

- definition review;
- source and lineage inspection;
- data correction;
- model explanation;
- alternative interpretation;
- documented resolution;
- temporary suspension when validity is uncertain.

Disputes are valuable evidence that the construct, data, or communication may be weak.

17.19. Change management

Metric or policy changes should be managed like software changes:

- proposal;
- review;
- test and validation;
- versioned release;
- migration plan;
- communication;
- monitoring;
- rollback.

Silent dashboard changes are governance failures.

17.20. Auditability

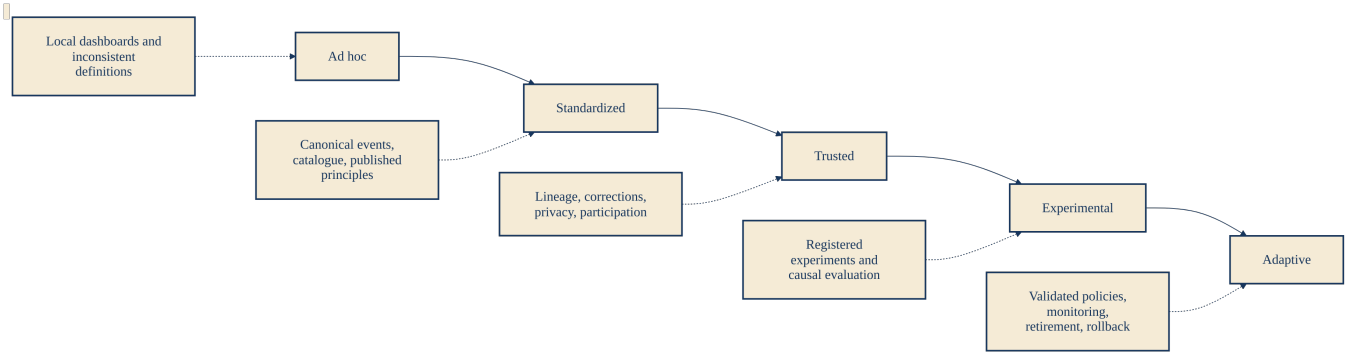
High-impact decisions should be reconstructable from:

- data and metric versions;
- model and policy versions;
- experiment record;
- access history;
- approvals;
- external factors;
- final decision rationale.

Auditability protects both the organization and affected individuals.

17.21. Adoption maturity model

Stage	Characteristics
Ad hoc	Local dashboards, inconsistent definitions, limited governance
Standardized	Canonical events, metric catalogue, published principles
Trusted	Transparent lineage, corrections, privacy controls, practitioner participation
Experimental	Registered experiments, guardrails, causal evaluation, external-factor tracking
Adaptive	Validated models and policies, continuous monitoring, formal retirement and rollback



17.22. Common failure modes

17.22.1. Starting with executive ranking dashboards

This creates defensive behavior before the platform demonstrates value.

17.22.2. Policy without architecture

Rules prohibit misuse, but broad raw access makes misuse easy.

17.22.3. Consent theater

People are nominally informed but have no meaningful understanding, correction path, or protection.

17.22.4. Treating fairness as equal averages

Different work populations should not be expected to have identical raw metrics.

17.22.5. Permanent experimental telemetry

Detailed data collected for one study is retained and reused indefinitely.

17.22.6. Governance only after harm

Oversight is created reactively after a metric or model has already shaped behavior.

17.23. Worked example: responding to a metric-misuse incident



Runnable version: `companion/notebooks/18-governance-classification.py`.

A vice president of engineering asks the intelligence team for "the top 20 lowest-performing engineers" ranked by review-latency contribution. The registered use of the review-latency metric is Class 2 (organizational decision support); individual-level ranking would be Class 4 and is currently prohibited without a separate formal process.

The governance operating model triggers three responses within 48 hours. First, the metric owner declines the request in writing, citing the registered purpose and the acceptable-use standard. Second, the request is logged as a governance incident and routed to the governance board for review. Third, the platform confirms that the metric API's purpose-bound access already blocked the underlying query at the interface layer, so no ad-hoc query produced the ranking.

The board review, one week later, considers two questions. Was the requester properly informed of the classification, and does the review-latency metric have any legitimate purpose at the individual level? The board concludes the requester's expectation reflected a widespread misunderstanding, updates the leadership onboarding materials to clarify Class 4 boundaries, and reaffirms the prohibition. It does not sanction the requester; the point of the process is to make legitimate boundaries visible, not to punish the first person who bumps against them.

The incident is closed with three artefacts: the decline letter, the board minute, and the updated onboarding note. Twelve months later, an audit of governance incidents shows this pattern of request occurring three more times but declining after the onboarding change was rolled out — the governance loop demonstrably shifted behavior without any punitive step.

17.24. Key takeaways

- Engineering measurement changes behavior and incentives.
- Purpose limitation, minimization, transparency, contestability, and separation of uses are foundational.
- System telemetry should not silently become individual performance scoring.
- Privacy and governance should be enforced through architecture, not policy alone.
- Models and routing policies require lifecycle governance, fairness analysis, and rollback.
- Trust is a functional dependency of valid Engineering Intelligence.
- Adoption should begin with local value and practitioner participation.

17.25. Open questions

- What governance structures best balance enterprise consistency and team autonomy?
- How can organizations evaluate cognitive burden without crossing into surveillance?
- Which metric uses should require employee or works-council approval in different jurisdictions?
- How should fairness be defined for teams owning fundamentally different systems?
- What forms of transparency improve trust without exposing sensitive security or business data?

17.26. Implementation checklist

1. Publish principles, acceptable uses, and prohibited uses before broad rollout.
2. Apply pseudonymization and aggregation by default.
3. Establish metric, experiment, data, and model governance ownership.
4. Define correction, dispute, and suspension processes.
5. Record intended and prohibited uses for every metric and model.
6. Review gaming surfaces and incentive effects before operationalization.
7. Set retention and deletion policies for every data class.
8. Include practitioner representation in governance.
9. Separate system improvement from individual performance management.
10. Audit high-impact decisions and maintain rollback capability.

17.26.1. Research foundations and source notes

The governance, ethics, and adoption framework in this chapter draws on measurement validity, indicator corruption, privacy engineering, professional ethics, cybersecurity governance, AI risk management, organizational learning, and socio-technical change. It extends those foundations into a practical operating model for legitimate Engineering Intelligence.

Measurement changes behavior

Campbell's Law describes how indicators used for consequential decision-making become vulnerable to corruption and can distort the process they are intended to monitor. See [campbell1979].

Goodhart's Law expresses a related principle: when a measure becomes a target, it can cease to function as a useful measure. See [goodhart1984].

These foundations support the chapter's treatment of behavioral adaptation as part of the measured system rather than as misconduct by default.

Examples include:

- splitting work to improve visible throughput;
- suppressing incident declaration;
- optimizing AI acceptance rather than contribution;
- avoiding difficult work;
- moving coordination outside instrumented channels;

- reducing review depth to improve latency.

The governance response is to design incentive-aware metric bundles, guardrails, periodic validity review, and retirement mechanisms.

Use-specific validity

Messick's validity framework emphasizes that validity concerns the evidence supporting a score's intended interpretation and use, including consequences. See [messick1995].

The chapter therefore treats governance as part of measurement validity rather than as an external compliance layer.

A metric may be appropriate for diagnosing a queue and invalid for ranking individuals. The same computation can have different validity depending on population, decision, and consequence.

Purpose limitation and data minimization

The NIST Privacy Framework provides a risk-based foundation for identifying, governing, controlling, communicating, and protecting privacy risk. See [nist2020-privacy].

The chapter applies these principles through:

- declared purposes;
- collection minimization;
- proportional granularity;
- retention limits;
- pseudonymization;
- restricted re-identification;
- correction and appeal;
- purpose-bound access.

The preferred design is to collect the least detailed data capable of supporting the declared decision.

Privacy by architecture

Policy alone is insufficient when broad raw access makes misuse easy.

The chapter therefore requires structural controls such as:

- separate identity vaults;
- pseudonymous analytical data;
- aggregation by default;
- minimum group-size thresholds;
- row- and column-level authorization;
- audited identity lookup;
- deletion propagation;
- short retention for detailed behavioral telemetry.

This is an Engineering Intelligence-specific implementation of privacy engineering principles informed by [nist2020-

privacy].

Professional ethics

The ACM Code of Ethics emphasizes contributing to society, avoiding harm, honesty, fairness, privacy, competence, and accountability. See [acm2018-ethics].

These principles support the chapter's restrictions on:

- covert surveillance;
- unsupported individual productivity scoring;
- automated employment decisions from engineering telemetry;
- punitive use of experimental metrics;
- opaque or unchallengeable classifications;
- use beyond the original declared purpose.

Professional responsibility applies to the designers, operators, analysts, managers, and decision-makers using the system.

Cybersecurity governance

The NIST Cybersecurity Framework provides a foundation for governing, identifying, protecting, detecting, responding to, and recovering from cybersecurity risk. See [nist2024-csf].

Engineering Intelligence systems can reveal sensitive source metadata, organizational structure, vulnerabilities, and strategic work.

The chapter therefore treats security controls as governance controls, including:

- least privilege;
- secrets management;
- encryption;
- audit logging;
- environment isolation;
- incident response;
- vendor-risk review;
- tested recovery.

AI and model governance

The NIST AI Risk Management Framework and its generative-AI profile provide foundations for governing, mapping, measuring, and managing AI risk. See [nist2023-airmf] and [nist2024-genai].

Management-system and risk-process standards are represented by 001 and 894.

The chapter operationalizes these principles through model records containing:

- accountable owner;
- intended and prohibited uses;

- training and evaluation evidence;
- feature and sensitive-attribute review;
- subgroup performance;
- calibration and uncertainty;
- drift monitoring;
- human override;
- rollback and retirement criteria;
- version history.

These standards guide governance but do not prove that a model is suitable for a particular decision.

Human accountability

AI systems, classifiers, risk scores, and generated explanations may inform decisions, but responsibility remains with accountable human roles.

The chapter distinguishes between:

- evidence generation;
- automated recommendation;
- delegated action;
- policy approval;
- consequential decision.

The person or body making the decision should be able to inspect evidence, understand uncertainty, override the system, and document rationale.

Contestability and correction

A legitimate measurement system must permit challenge and correction.

The chapter's metric-rights model includes the ability to:

- know what is measured;
- understand definitions;
- inspect relevant data where appropriate;
- correct factual errors;
- challenge inferred links or classifications;
- understand retention and access;
- know whether data affects consequential decisions.

Contestability improves both fairness and data quality because affected practitioners often possess context unavailable to centralized instrumentation.

Fairness and disparate impact

Fairness cannot be reduced to equal raw metric values across populations performing different work.

Potential sources of disparate impact include:

- legacy-system ownership;
- role and tenure;
- accessibility-related workflows;
- distributed time zones;
- specialist responsibilities;
- repository and domain complexity;
- unequal access to tools or support.

The chapter therefore evaluates error rates, burden distribution, policy effects, and decision consequence by relevant subgroup.

The goal is not identical averages; it is avoidance of unjustified harm from invalid comparison or routing.

Individual-level measurement

Some actor identity is needed for provenance, permissions, handoffs, and correction.

That need does not justify named productivity scorecards.

The chapter's default controls—pseudonymization, aggregation, restricted lookup, minimum group sizes, and explicit review of new uses—follow the use-specific validity and privacy principles in [messick1995] and [nist2020-privacy].

Individual data may support private self-reflection or factual correction without becoming managerial ranking data.

Experiment ethics

Experiments involving practitioners can alter workload, security, review quality, or access to tools.

The causal-design foundations in [shadish2002], [imbens2015], [hernan2020], and [kohavi2020] support the chapter's requirements for:

- risk and reversibility review;
- guardrails;
- treatment transparency appropriate to the design;
- burden analysis;
- post-study retention decisions;
- separation from punitive employment action.

Low-risk interface experiments are ethically different from experiments that alter security review or collect detailed behavioral traces.

Agent governance

Agentic systems introduce questions of permission, delegation, evidence, shutdown, and side-effect accountability.

The chapter therefore requires explicit autonomy tiers and distinguishes authority to:

- inspect;

- recommend;
- modify;
- execute tests;
- merge;
- deploy;
- alter infrastructure or data;
- delegate to other agents.

This is an Engineering Intelligence-specific extension of provenance and AI risk-governance principles in [moreau2013], [nist2023-airmf], and [nist2024-genai].

Trust as a system dependency

Trust affects whether practitioners correct data, participate in experiments, disclose problems, and accept conclusions.

The chapter's trust model is conceptual rather than a mechanically optimized score.

Trust is strengthened by:

- accuracy;
- transparency;
- visible local benefit;
- correction mechanisms;
- fair process;
- restraint;
- reliable enforcement of prohibited uses.

Trust is weakened by opaque, punitive, surprising, or obviously incomplete measurement.

Organizational learning

Argyris and Schön distinguish single-loop learning from double-loop learning, where organizations reconsider the assumptions and governing values behind their actions. See [argyris1978].

Senge's learning-organization work emphasizes systems thinking, shared learning, and collective capability. See [senge1990].

The chapter therefore treats disputes, null results, corrections, and metric retirement as learning mechanisms rather than governance failures.

Practitioner participation

Engineering Intelligence is a socio-technical system. Practitioner participation improves construct definition, acceptable-use policy, error correction, and adoption.

The broader socio-technical context is informed by [conway1968], [cataldo2006], and [herbsleb2007].

The chapter recommends including affected engineers in metric councils, experiment review, model governance, and policy design.

This reduces the risk that centrally designed measures misunderstand local workflow or redistribute burden invisibly.

Adoption sequencing

The chapter recommends beginning with visible system problems that already matter to practitioners, such as unreliable CI, review queues, ownership ambiguity, or incident linkage.

This sequencing is consistent with developer-experience research emphasizing feedback loops, cognitive load, and flow. See [forsgren2021-space], [noda2023-devex], and [fagerholm2012].

Adoption should expand only after the system demonstrates usefulness, accuracy, transparency, and restraint.

Governance as versioned change

Metric, model, and policy changes should follow software-like lifecycle discipline:

- proposal;
- review;
- validation;
- versioned release;
- migration;
- communication;
- monitoring;
- rollback;
- retirement.

This is an original Engineering Intelligence operating principle informed by reliable-change practices in [humble2010], [beyer2016], and [beyer2018].

Silent dashboard or policy changes undermine auditability and trust.

Auditability

High-impact decisions should be reconstructable from:

- evidence and metric versions;
- model and policy versions;
- experiment records;
- access history;
- approvals;
- external factors;
- final rationale;
- later correction or appeal.

The provenance foundation is provided by [moreau2013].

Auditability protects affected practitioners and the organization by making the decision path inspectable.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- metric rights for engineering practitioners;
- explicit acceptable, heightened-governance, inappropriate, and prohibited use classes;
- a required gaming-surface analysis for consequential metrics;
- purpose-bound access combining role, population, classification, and declared use;
- correction, dispute, and temporary-suspension procedures;
- governance maturity tied to evidence quality and practitioner trust;
- versioned metric and policy releases with rollback;
- autonomy tiers and delegation records for engineering agents;
- decision audit trails spanning evidence, model, experiment, policy, and rationale;
- trust treated as a functional dependency of valid measurement;
- adoption beginning with local practitioner value rather than executive ranking.

These constructs should be evaluated through audit outcomes, dispute resolution, data corrections, practitioner participation, trust evidence, observed incentive effects, and the consequences of decisions made with the system.

Chapter 18. Portfolio and Executive Decision-Making



Engineering Intelligence ultimately exists to improve choices about where the organization *invests money, attention, time, and risk capacity. This chapter explains how change-level evidence can support portfolio decisions without collapsing engineering into a single score. It covers capacity allocation, opportunity cost, option value, scenario analysis, uncertainty, dependency-aware prioritization, constrained optimization, executive scorecards, and decision records.*

18.1. From measurement to allocation

Engineering metrics become strategically useful when they inform allocation decisions such as:

- which initiatives should receive scarce engineering capacity;
- where specialist review capacity should be added;
- which platform investment will reduce the largest recurring cost;
- whether to buy, build, replace, or retire an engineering tool;
- which migrations should be accelerated because delay is expensive;
- where risk should be accepted, reduced, transferred, or avoided.

The objective is not to maximize throughput in isolation. It is to maximize expected organizational value subject to constraints on capacity, quality, safety, regulation, and strategic direction.

18.2. Portfolio value model

For Engineering Change or initiative e, define expected net value:

$$NV(e) = E[B(e)] - E[C(e)]$$

where $B(e)$ includes direct business benefit, risk reduction, learning, capability, and option value, while $C(e)$ includes human, tool, infrastructure, coordination, delay, quality, and risk costs.

A portfolio decision selects a set of changes:

$$\max_{x_e \in \{0,1\}} \sum_e x_e NV(e)$$

subject to constraints such as:

$$\sum_e x_e r_{e,k} \leq Capacity_k$$

for each constrained resource k.

This formulation is intentionally simplified. Real portfolios include dependencies, mandatory work, correlated risks, staged decisions, and uncertain outcomes.

18.3. Capacity is multidimensional

Headcount is not one interchangeable pool. Capacity includes:

- general implementation effort;
- specialist domain expertise;
- review and approval bandwidth;
- platform and infrastructure capacity;
- deployment windows;
- test environments;
- leadership attention;
- change absorption capacity in customers and operations.

A plan can be fully funded in nominal engineer-months and still be infeasible because one scarce expertise pool is overcommitted.

18.4. Opportunity cost

The economic cost of choosing one initiative includes the value of the best alternative displaced.

If a security migration consumes the only database specialists for a quarter, the relevant comparison is not just its direct cost. It includes the value of other work delayed by that constraint.

Engineering Intelligence makes opportunity cost visible through queueing, dependency, cost-of-delay, and scarce-attention models.

18.5. Mandatory and strategic work

Not all work is selected by return on investment.

Mandatory classes include:

- regulatory commitments;
- critical security remediation;
- contractual obligations;
- safety and financial-integrity controls;
- end-of-life migrations;
- incident recovery.

These should be represented as constraints or minimum commitments rather than assigned artificial financial benefits to force them into a ranking.

Strategic work may also have option value that exceeds its immediate cash return.

18.6. Option value

An engineering investment can create the right, but not the obligation, to pursue future opportunities.

Examples include:

- establishing a reusable platform;
- removing vendor lock-in;
- creating a clean data boundary;
- adding observability before a major migration;
- building a prototype to resolve uncertainty;
- introducing an abstraction that enables multiple products.

Option value increases when future uncertainty is high, the decision can be staged, and the investment preserves flexibility.

A small experiment may be valuable because it prevents a much larger mistaken commitment.

18.7. Value of information

The expected value of information is the improvement in decision quality enabled by new evidence.

$$EVI = E[V\ after\ information] - E[V\ before\ information] - Cost_{information}$$

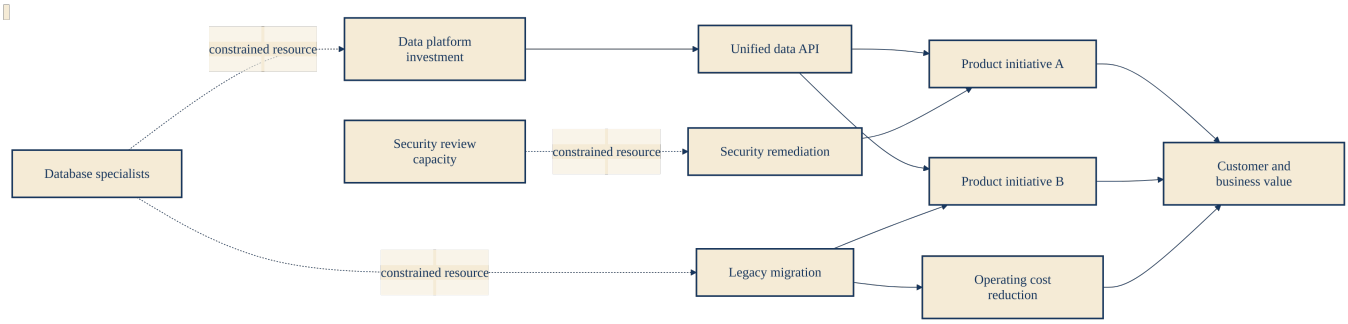
This framework supports investment in prototypes, instrumentation, experiments, and technical discovery.

Research work should not be judged only by immediate shipped output when its purpose is to reduce uncertainty.

18.8. Dependency-aware portfolios

Initiatives are not independent. One migration may enable several products. A shared platform may reduce future cost across a portfolio. Two changes may compete for the same deployment window.

Represent the portfolio as a directed graph with edges such as `ENABLES`, `BLOCKS`, `REQUIRES`, `COMPETES_FOR`, and `REDUCES_RISK_OF`.



Useful measures include:

- prerequisite depth;
- number and value of enabled changes;
- shared constrained resources;
- critical portfolio path;
- dependency concentration;
- value unlocked per enabling investment.

18.9. Cost of delay at portfolio level

Portfolio prioritization should account for the shape of delay cost.

A fixed-date regulatory change, an incident remediation, a seasonal launch, and an internal refactor have different value curves.

A useful prioritization input is weighted shortest job first:

$$WSJF(e) = \frac{CostOfDelay(e)}{EstimatedDuration(e)}$$

It is a heuristic, not a universal truth. It should be adjusted for dependencies, uncertainty, risk, and scarce-resource constraints.

18.10. Uncertainty and distributions

Executive decisions should not rely only on expected values.

For each initiative, report:

- median and expected value;
- credible interval;
- downside and upside scenarios;
- probability of negative net value;
- probability of missing a deadline;
- key uncertainty drivers;

- reversibility and exit options.

Two initiatives with the same expected value may have very different risk profiles.

18.11. Scenario planning

Scenario planning is useful when external conditions dominate precise forecasting.

Scenarios might vary:

- demand and revenue;
- regulatory timing;
- vendor pricing;
- staffing and attrition;
- AI model capability and cost;
- infrastructure constraints;
- acquisition or reorganization;
- security and reliability environment.

The goal is not to predict one future. It is to identify decisions that remain robust across several plausible futures.

18.12. Monte Carlo portfolio simulation

When duration, cost, benefit, and dependency outcomes are uncertain, simulation can estimate portfolio distributions.

For each run:

1. sample initiative cost and duration;
2. sample benefit and failure outcomes;
3. apply dependencies and resource constraints;
4. compute delivery dates and cost of delay;
5. aggregate portfolio value and risk.

Outputs include probability of meeting commitments, distribution of net value, resource bottleneck frequency, and sensitivity to assumptions.

18.13. Risk appetite and constraints

A risk-neutral portfolio maximizes expected value. Real organizations impose constraints on catastrophic loss, compliance, liquidity, operational capacity, and reputation.

Risk policies may include:

- maximum aggregate security exposure;
- minimum reliability investment;
- limits on simultaneous irreversible migrations;
- required contingency capacity;
- concentration limits for critical vendors or expertise;

- minimum experimental evidence before scale-up.

These are governance constraints, not metric weights to be traded away silently.

18.14. Portfolio balance

A healthy portfolio often balances:

- customer and growth work;
- reliability and security;
- regulatory and mandatory commitments;
- platform and developer experience;
- architecture and debt reduction;
- learning and experimentation;
- operational maintenance.

The correct balance is contextual. Engineering Intelligence provides evidence about consequences rather than one prescribed percentage.

18.15. Platform investment economics

Platform investments distribute benefit across many future Engineering Changes.

Evaluation should estimate:

- population of affected changes;
- reduction in active effort, queueing, infrastructure, or failure cost;
- adoption and migration cost;
- time until benefit begins;
- maintenance burden;
- heterogeneous effects across teams;
- option value and lock-in.

A platform can have positive average value while harming specialized teams. Preserve subgroup analysis.

18.16. Tool portfolio decisions

Engineering organizations often accumulate overlapping tools.

A tool portfolio review should compare:

- total and marginal cost;
- actual exposure and adoption;
- causal effect on effort, quality, and delay;
- integration and governance cost;
- vendor risk;
- switching cost;

- overlapping capability;
- long-term strategic fit.

License utilization alone is insufficient. A lightly used specialist tool can be highly valuable, while a widely used tool can create little net benefit.

18.17. Executive scorecards

An executive scorecard should present a compact set of decision-oriented views rather than a universal productivity score.

Recommended views include:

View	Question
Portfolio flow	Where is value delayed and why?
Capacity constraints	Which expertise, review, platform, or deployment resources are binding?
Quality and risk	Where is expected failure loss or blast radius concentrated?
Engineering economics	Where are money and attention consumed, and what value is created?
Experiment evidence	Which interventions have demonstrated causal benefit?
Platform health	Can the measurement system itself be trusted?

Each view should expose uncertainty, work mix, and drill-down.

18.18. Avoiding the one-score trap

Combining flow, cost, quality, cognition, and value into one enterprise score creates fragile assumptions and gaming opportunities.

A multi-objective decision view is preferable. Leaders should see trade-offs and constraints rather than a fabricated single ranking.

Composite scores may be useful for narrow routing decisions when components, weights, and policy are explicit.

18.19. Decision records

Material engineering investment decisions should preserve:

- decision and alternatives;
- expected value and cost distributions;
- constraints and dependencies;
- evidence and experiment results;
- assumptions and external scenarios;
- dissenting views;

- selected option;
- review date and trigger conditions.

A decision record enables later learning without judging past choices solely by realized outcome.

18.20. Forecast calibration

Organizations should compare forecasts with outcomes.

Calibration measures include:

- estimated versus actual duration;
- predicted versus realized cost;
- forecast probability versus event frequency;
- expected versus realized adoption;
- anticipated versus observed quality effects.

The purpose is to improve forecasting models, not punish uncertainty.

18.21. Portfolio review cadence

Different decisions require different cadences:

- near-real-time for incidents and urgent risk;
- weekly for operational queues and constrained capacity;
- monthly for tool and platform economics;
- quarterly for portfolio allocation and strategic scenarios;
- event-driven for major external changes.

Do not force all measures into one executive meeting rhythm.

18.22. Common failure modes

18.22.1. Ranking by ROI alone

Mandatory work, option value, risk constraints, and dependencies are ignored.

18.22.2. Treating capacity as fungible

Plans overcommit scarce expertise while nominal headcount appears sufficient.

18.22.3. Double counting value

Time savings, throughput, and delay reduction may represent overlapping benefit.

18.22.4. Ignoring uncertainty

Single forecasts create false confidence and brittle commitments.

18.22.5. Portfolio theater

Scores are produced after strategic choices have already been made and do not affect allocation.

18.22.6. Punishing forecast error

Teams hide uncertainty and produce politically safe estimates.

18.23. Worked example: allocating capacity across three initiatives



Runnable version: `companion/notebooks/19-portfolio-allocation.py`.

An engineering organization has 42 engineers of general capacity and 6 engineers of scarce security-specialist capacity available for the next quarter. Three initiatives compete for that capacity:

Initiative A — payments migration. Expected value $8M realised over 18 months (revenue enablement + risk reduction), estimated cost 24 general engineer-quarters and 3 specialist-quarters, dependency on a shared identity service, cost of delay $180k per month, downside if incomplete: minor.

Initiative B — internal developer platform upgrade. Expected value $4M over 24 months (recovered productivity across 900 engineers), cost 30 general engineer-quarters and 0 specialist-quarters, no external dependency, cost of delay $60k per month, downside if incomplete: capability continues to work as-is.

Initiative C — regulatory compliance program. Expected value not primarily monetary but mandatory; failing to complete on schedule triggers a $12M regulatory penalty and reputational risk. Cost 18 general engineer-quarters and 5 specialist-quarters, hard deadline 30 September, cost of delay approaches infinity as the deadline nears.

A simplistic ROI ranking would prefer A ($8M / 27 quarters = $296k/quarter) over B ($4M / 30 = $133k/quarter). But C's mandatory constraint locks in 18 general + 5 specialist quarters, leaving only 24 general + 1 specialist quarter free. C is not a portfolio choice; it is a floor. Within the remaining capacity, A is still short one specialist-quarter and B has 30 general-quarters of demand against 24 available.

The portfolio decision becomes: fund C fully, fund B partially with sequenced quarters (accepting a modest slip in cost of delay), and defer A by one quarter — or negotiate specialist capacity from another portfolio to unlock A now. The trade-off is explicit and traceable: A's $180k monthly cost of delay is the price of unlocking C's mandatory constraint. That price is not hidden inside a composite score.

18.24. Key takeaways

- Portfolio decisions allocate multiple constrained resources, not just headcount or budget.
- Opportunity cost and cost of delay connect engineering flow to strategic value.
- Mandatory constraints, option value, and learning value should not be forced into simplistic ROI rankings.
- Dependencies and shared bottlenecks require graph-aware portfolio analysis.
- Decisions should use distributions, scenarios, and risk constraints rather than point forecasts alone.
- Executive scorecards should preserve trade-offs and avoid one universal score.

- Decision records and forecast calibration turn portfolio choices into organizational learning.

18.25. Open questions

- How should option value be estimated consistently for platform and architecture work?
- Which portfolio optimization methods remain understandable enough for executive use?
- How should correlated technical and business risks be represented?
- What evidence is sufficient to scale an intervention from experiment to enterprise policy?
- How can organizations reward honest uncertainty in planning?

18.26. Implementation checklist

1. Define value, cost, risk, and mandatory-work classes.
2. Model scarce capacity by expertise and operational constraint.
3. Build a temporal dependency graph across initiatives and Engineering Changes.
4. Use cost-of-delay curves and scenario ranges.
5. Represent forecasts as distributions with key sensitivity drivers.
6. Add option and information value for staged investments.
7. Maintain executive views for flow, constraints, risk, economics, experiments, and data quality.
8. Avoid universal productivity or portfolio scores.
9. Record material decisions and review triggers.
10. Calibrate forecasts against realized outcomes.

18.26.1. Research foundations and source notes

The portfolio and executive decision-making framework in this chapter draws on cost-benefit analysis, capital allocation, real-options reasoning, productivity economics, decision analysis, causal inference, and organizational learning. It extends those foundations into an Engineering Intelligence operating model for prioritizing investments under uncertainty.

Cost-benefit analysis

The chapter's economic framing is grounded in standard cost-benefit analysis. See [boardman2018].

This supports explicit treatment of:

- implementation and operating cost;
- avoided cost;
- released capacity;
- quality and risk effects;
- timing of benefits;
- discounting;
- uncertainty;
- distribution of costs and benefits;
- opportunity cost.

The relevant question is not whether an initiative produces activity or local improvement, but whether it creates net value relative to credible alternatives.

Opportunity cost

Every engineering investment consumes scarce capital, specialist attention, organizational capacity, and calendar time.

The chapter therefore treats the cost of an initiative as including the value of the best feasible alternative that cannot be pursued simultaneously.

Examples include:

- platform work displacing a product initiative;
- migration work delaying reliability improvements;
- AI-tool integration consuming security and procurement capacity;
- specialist review assigned to low-risk changes rather than high-risk work.

Opportunity cost is often omitted from engineering business cases because it is not recorded in a purchase order. It remains economically real.

Net present value

Benefits and costs occurring at different times should not be treated as equivalent.

The chapter uses discounted cash-flow logic to compare future costs and benefits in present-value terms.

A simplified net-present-value expression is:

$$NPV = \sum_{t=0}^{T} \frac{Benefits_t - Costs_t}{(1+r)^t}$$

where r is the selected discount rate.

The choice of horizon and discount rate should be visible because long-lived architecture, capability, and risk investments can appear unattractive under short horizons.

Real options and staged commitment

Real-options reasoning treats flexibility as valuable under uncertainty.

An organization may create value by preserving the option to expand, defer, switch, stop, or reverse an investment after learning more.

The chapter applies this through:

- pilots;
- phased rollout;
- reversible architecture;
- vendor exit paths;
- shadow evaluation;

- modular contracts;
- stop conditions;
- staged funding.

The option value is not identical to expected cash flow. It reflects the value of learning before making an irreversible commitment.

This treatment is an Engineering Intelligence-specific synthesis of real-options principles and reliable-change practices.

Productivity realization

Information-technology investments often require complementary organizational changes before productivity benefits appear. See [brynjolfsson1993] and [brynjolfsson2017].

This supports the chapter's distinction among:

- technical capability delivered;
- adoption;
- workflow change;
- capacity released;
- capacity redeployed;
- financial or customer value realized.

A tool can be deployed successfully without creating measurable enterprise value.

Released capacity versus savings

The chapter distinguishes released engineering time from realized cash savings.

Released capacity may produce value when it is used for:

- additional customer outcomes;
- reliability work;
- reduced backlog;
- faster learning;
- reduced contractor or hiring demand;
- avoided overtime;
- lower operational risk.

It should not be reported automatically as payroll savings when staffing cost remains unchanged.

This distinction is an original Engineering Intelligence operating rule grounded in cost-benefit principles from [boardman2018].

Risk-adjusted value

Expected value combines possible outcomes with their probabilities, but executives should also examine tail consequence, reversibility, and uncertainty.

A risk-adjusted decision should consider:

- expected benefit;
- probability of failure;
- downside magnitude;
- detectability;
- reversibility;
- correlated risk;
- strategic dependence;
- uncertainty in the estimate itself.

A high expected return may still be inappropriate when downside is catastrophic and recovery is weak.

Portfolio diversification

Engineering portfolios can become concentrated by vendor, platform, architecture, skill, or model provider.

Diversification can reduce correlated failure and preserve strategic flexibility, but excessive diversity increases integration and operating cost.

The chapter therefore treats diversification as a tradeoff among:

- correlated operational risk;
- vendor concentration;
- model monoculture;
- skill concentration;
- platform complexity;
- purchasing leverage;
- exit cost.

The optimal portfolio is not necessarily the one with the fewest technologies or vendors.

Decision analysis under uncertainty

Executive decisions frequently involve incomplete evidence and multiple plausible futures.

The chapter uses:

- scenario analysis;
- sensitivity analysis;
- break-even analysis;
- expected-value ranges;
- confidence intervals;
- decision thresholds;
- value-of-information analysis.

The purpose is not to manufacture precision. It is to expose which assumptions drive the decision and where additional evidence could change the choice.

Value of information

Additional research or experimentation is valuable when it has a meaningful chance of changing a consequential decision.

The expected value of information depends on:

- current uncertainty;
- cost of obtaining evidence;
- reversibility of the decision;
- consequence of choosing incorrectly;
- time available;
- decision sensitivity.

This supports investing in better evidence for large, irreversible, high-risk decisions while avoiding endless analysis for small reversible choices.

Causal evidence

A portfolio decision should distinguish correlation, prediction, and causal effect.

The methodological foundations are described in [rubin1974], [holland1986], [pearl2009], [imbens2015], and [hernan2020].

The chapter therefore labels evidence according to causal status and avoids treating a metric improvement after rollout as proof that the intervention caused it.

Randomized or quasi-experimental evidence receives greater weight when the decision depends on attributable impact.

Evidence maturity

Evidence varies in maturity.

The chapter distinguishes:

- hypothesis;
- benchmark evidence;
- pilot evidence;
- controlled causal evidence;
- replicated production evidence;
- realized financial or operational outcome.

This maturity ladder is an original Engineering Intelligence construct.

It prevents early benchmark or vendor evidence from being presented as proven enterprise value.

Delivery and capability outcomes

Portfolio decisions should consider system-level delivery performance and developer experience.

Relevant foundations include [forsgren2018], [forsgren2021-space], and [noda2023-devex].

The chapter therefore includes outcomes related to:

- delivery responsiveness;
- reliability;
- quality;
- feedback loops;
- cognitive load;
- flow;
- collaboration;
- practitioner experience.

These measures should remain multidimensional rather than collapsed into a single productivity index.

Architecture as capital

Architecture can create durable capability by reducing the recurring cost and risk of future change.

The chapter treats architecture investment as a form of productive capital whose return may appear through:

- lower coordination cost;
- smaller validation surface;
- reduced blast radius;
- faster deployment;
- easier substitution;
- improved resilience;
- greater strategic optionality.

The modularity foundation is provided by [parnas1972], while socio-technical implications are informed by [conway1968], [cataldo2006], and [skelton2019].

Architectural benefit should be evaluated longitudinally through Engineering Change outcomes rather than assumed from design intent.

Reliability and resilience investment

Reliability work often creates value by reducing expected loss and preserving customer trust rather than by increasing visible feature output.

SRE and resilience foundations are described in [beyer2016], [beyer2018], and [woods2015].

Portfolio evaluation should therefore include:

- expected incident loss;
- recovery capability;
- tail-risk reduction;
- detection and rollback improvement;
- reduced operational toil;
- graceful degradation;

- preserved adaptive capacity.

A low incident count does not prove that resilience investment lacks value.

AI investment decisions

AI investment should include more than license or inference cost.

The chapter includes:

- integration;
- security and governance;
- retrieval and indexing;
- human validation;
- false-positive handling;
- rework;
- switching and exit;
- model drift;
- skill and dependency effects.

Empirical foundations include [peng2023], [vaithilingam2022], [barke2023], and [jimenez2024], while AI risk governance is informed by [nist2023-airmf] and [nist2024-genai].

The decision should be based on quality-adjusted net outcomes in the organization's own workflow.

Portfolio comparability

Initiatives differ in scale, maturity, risk, and evidence quality.

The chapter therefore recommends a common decision record rather than a universal score.

Comparable fields include:

- decision objective;
- baseline;
- cost range;
- benefit mechanism;
- evidence maturity;
- uncertainty;
- risk and reversibility;
- dependencies;
- time to value;
- owner;
- follow-up date.

The common structure improves comparison without pretending that all investments can be reduced credibly to one number.

Sensitivity and break-even analysis

Business cases often depend on uncertain assumptions such as adoption, time saved, defect reduction, or value per unit of capacity.

The chapter therefore requires sensitivity analysis and break-even thresholds.

For example:

- What adoption level makes the investment worthwhile?
- How much validation effort can be added before an AI tool loses value?
- What incident reduction justifies a reliability investment?
- How long can a migration take before its NPV becomes negative?

This makes assumptions testable and supports post-decision learning.

Decision records

A high-consequence decision should preserve:

- alternatives considered;
- evidence and versions;
- assumptions;
- uncertainty;
- expected mechanisms;
- economic model;
- risks and guardrails;
- decision authority;
- review date;
- eventual outcome.

The provenance foundation is provided by [moreau2013].

Decision records create institutional memory and allow later evaluation of both outcomes and reasoning quality.

Organizational learning

Argyris and Schön's distinction between single-loop and double-loop learning supports reviewing not only whether an initiative succeeded but whether the organization's assumptions and decision process were sound. See [argyris1978].

Senge's systems-thinking perspective supports learning across interconnected technical and organizational effects. See [senge1990].

The chapter therefore recommends post-investment review of:

- realized outcomes;
- invalid assumptions;
- unanticipated consequences;
- evidence gaps;

- decision-process quality;
- reusable lessons.

Governance and incentive effects

Executive scorecards can distort behavior when portfolio measures become targets.

Campbell's Law and Goodhart's Law provide the relevant foundation. See [campbell1979] and [goodhart1984].

The chapter therefore recommends:

- metric bundles;
- narrative context;
- uncertainty;
- independent guardrails;
- periodic retirement;
- prohibition of unsupported individual ranking;
- explicit review of gaming surfaces.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- portfolio decisions organized around Engineering Change outcomes;
- explicit separation of released capacity, avoided cost, cash savings, modeled value, and realized value;
- an evidence-maturity ladder from hypothesis to realized outcome;
- value-realization plans linked to investment decisions;
- attention-adjusted and quality-adjusted economic cases;
- architecture and reversibility treated as strategic option value;
- portfolio concentration analysis for vendors, models, platforms, and skills;
- decision records connected to metrics, experiments, models, and evidence;
- post-investment review of both outcomes and decision reasoning;
- executive decision products that preserve uncertainty rather than hide it.

These constructs should be evaluated through realized outcomes, forecast accuracy, decision reversals, portfolio resilience, and the organization's ability to learn from both successful and unsuccessful investments.

Chapter 19. Implementation and Rollout



Engineering Intelligence should not begin as a multi-year data-platform program. It should *begin with a small number of consequential decisions, a narrow evidence path, and a governance model that earns trust. This chapter presents a phased rollout strategy for large organizations. It covers pilot selection, team structure, technical sequencing, data quality, change management, value proof, operating cadence, scaling criteria, and failure recovery.*

19.1. Start with decisions, not completeness

A common failure pattern is to attempt universal instrumentation before demonstrating value. The program spends months connecting tools, designing schemas, and building dashboards while practitioners see little improvement.

A better starting point is a decision that is important, repeated, measurable, and currently made with weak evidence.

Examples include:

- whether to adopt an AI pull-request reviewer;
- where review queues create the greatest delay;
- whether CI acceleration justifies its infrastructure cost;
- which specialist approvals should be risk-routed;
- whether a merge queue improves quality-adjusted integration time.

The initial platform should collect only the evidence needed to answer the selected decision credibly.

19.2. Pilot selection criteria

A good pilot domain has:

- a meaningful operational or economic problem;
- sufficient event volume for analysis;
- willing engineering partners;
- stable enough workflows to define treatment and outcomes;
- manageable security and privacy risk;
- accessible source-control, CI, and deployment evidence;
- a plausible intervention within one or two quarters.

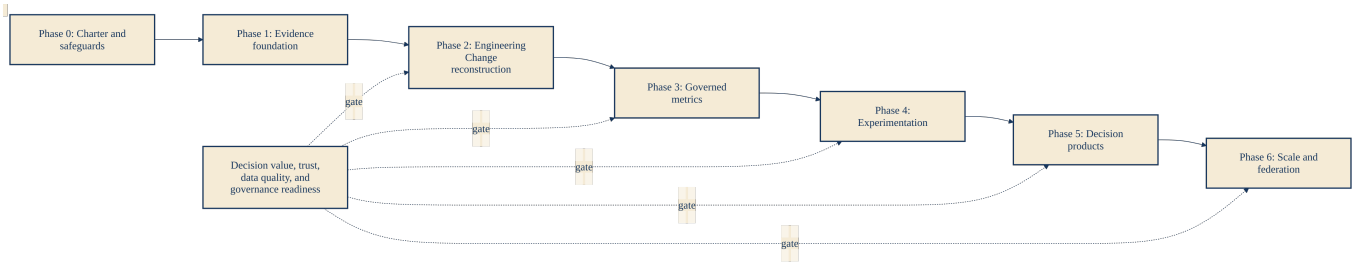
Avoid pilots chosen only because their data is easy. A technically convenient pilot with no decision consequence will not establish legitimacy.

19.3. The minimum viable evidence path

For many pilots, the minimum viable path includes:

1. source-control events;
2. CI and test outcomes;
3. deployment or accepted-outcome evidence;
4. canonical actor, repository, and team identities;
5. a basic Engineering Change linkage strategy;
6. a small metric catalogue;
7. experiment assignment and exposure when applicable.

This is enough to answer substantial questions without first building every graph, model, and dashboard described in the reference architecture.



19.4. Phase 0: charter and safeguards

Before instrumentation, define:

- the decisions the program will support;
- acceptable and prohibited uses;
- privacy and retention principles;
- pilot owners and stakeholders;
- success and stopping criteria;

- data-access boundaries;
- correction and dispute processes.

The charter should explicitly state that system telemetry will not be used for unsupported individual productivity ranking.

19.5. Phase 1: evidence foundation

The first technical phase should establish:

- canonical event envelopes;
- immutable raw storage;
- schema ownership;
- source freshness monitoring;
- deduplication and replay;
- temporal identity resolution;
- lineage from source event to metric.

The goal is not broad coverage. It is trustworthy evidence for the pilot.

19.6. Phase 2: Engineering Change reconstruction

Introduce explicit Engineering Change identifiers where teams can adopt them with low friction. Use deterministic and probabilistic reconstruction for legacy workflows.

Measure:

- explicit-ID coverage;
- inferred-link coverage;
- linkage confidence;
- correction rate;
- orphan artifact rate;
- event-to-change latency.

Do not hide weak coverage behind polished dashboards.

19.7. Phase 3: governed metrics

Implement a small metric bundle tied to the pilot decision.

For an AI review pilot, the first bundle might include:

- human review effort;
- AI validation burden;
- finding precision and ranking;
- cycle time;
- severe quality outcomes;
- tool and inference cost;

- data freshness and linkage coverage.

Each metric requires a version, owner, formula, population, uncertainty method, and guardrails.

19.8. Phase 4: experimentation

Add:

- eligibility service;
- deterministic assignment;
- treatment configuration;
- exposure logging;
- external-factor registry;
- predeclared analysis plan;
- sequential monitoring;
- decision record.

An experiment should not launch until exposure and guardrails are observable.

19.9. Phase 5: operational decision products

Deliver views tailored to the pilot users:

- engineers need current change and queue state;
- team leads need bottlenecks and work-mix context;
- platform owners need source freshness and tool economics;
- leaders need causal evidence and rollout decisions.

Avoid one dashboard attempting to serve every audience.

19.10. Phase 6: scale and federation

Scale only after the pilot demonstrates:

- credible decision impact;
- acceptable data quality;
- practitioner trust;
- stable metric definitions;
- manageable operating cost;
- repeatable onboarding;
- governance capacity.

Federation becomes important at scale. Central teams should own canonical contracts, identity, lineage, and experimentation foundations, while domain teams own source semantics, local metrics, and interpretation.

19.11. Team structure

A cross-functional implementation team typically needs:

- platform engineering;
- data engineering;
- applied statistics or causal inference;
- ML or information retrieval;
- developer experience;
- security and privacy;
- domain representatives;
- product or program leadership.

A small team with these capabilities is often more effective than a large dashboard organization lacking statistical or domain depth.

19.12. Product management for measurement

Engineering Intelligence should be managed as a product.

The product backlog should include:

- decisions to improve;
- user workflows;
- evidence gaps;
- metric quality;
- experiments;
- source reliability;
- governance debt;
- adoption and trust.

Usage is not the primary success metric. Decision quality and realized outcome improvement are.

19.13. Proving value

Early value should be demonstrated through concrete cases:

- a tool purchase avoided or justified;
- a queue bottleneck removed;
- specialist attention redirected;
- CI spend reduced without quality loss;
- an ineffective policy retired;
- an experiment prevented a costly enterprise rollout.

Value claims should distinguish realized savings, avoided future cost, released capacity, expected quality benefit, and theoretical model estimates.

19.14. Baseline before intervention

Before changing a workflow, establish:

- current metric distributions;
- work-mix and complexity features;
- source coverage and missingness;
- seasonality and release cycles;
- existing tool exposure;
- quality and economic baselines;
- external events.

A rushed intervention without baseline evidence may make causal interpretation impossible.

19.15. Data quality gates

High-level metrics should not publish when evidence falls below declared thresholds.

Possible gates include:

- minimum event completeness;
- maximum source lag;
- minimum Engineering Change linkage confidence;
- maximum unresolved identity rate;
- sufficient sample size;
- stable metric version;
- no known processing incident.

A visible unavailable state is better than a precise-looking but invalid number.

19.16. Onboarding new domains

A repeatable onboarding process should include:

1. identify domain owners and decisions;
2. inventory source systems and semantics;
3. map canonical identities;
4. establish Engineering Change boundaries;
5. validate event completeness;
6. calibrate work-mix and complexity features;
7. review privacy and access;
8. run metrics in shadow mode;
9. compare with domain expert understanding;
10. publish only after validation.

19.17. Shadow mode

New metrics and policies should initially operate without affecting decisions.

Shadow mode allows teams to inspect:

- reconstructed changes;
- predicted risk;
- routing recommendations;
- metric values;
- false positives and omissions;
- likely behavioral effects.

Corrections gathered during shadow mode improve both data and trust.

19.18. Rollout patterns

Useful rollout patterns include:

- opt-in pilot;
- selected representative teams;
- staged repository cohorts;
- risk-bounded populations;
- shadow recommendations;
- advisory before blocking policy;
- regional or organizational waves.

The rollout pattern should support both operational safety and causal learning.

19.19. Change communication

Communication should explain:

- the decision and problem;
- what data is collected;
- what is not collected;
- metric definitions and uncertainty;
- prohibited uses;
- pilot duration and review points;
- correction paths;
- how success will be judged.

Avoid framing the initiative as a way to measure which engineers are productive.

19.20. Operating cadence

A sustainable cadence may include:

19.20.1. Weekly operational review

Source failures, queue anomalies, experiment exposure, and data-quality incidents.

19.20.2. Monthly metric review

Definition changes, quality, drift, dashboard use, and action outcomes.

19.20.3. Quarterly governance review

New uses, privacy, retention, model performance, fairness, and metric retirement.

19.20.4. Experiment decision reviews

Held when predeclared evidence or stopping conditions are met.

19.21. Scaling source integrations

Source integrations should use reusable connector contracts and centralized reliability patterns.

Connector responsibilities include:

- source authentication;
- pagination and rate limits;
- event deduplication;
- schema translation;
- source cursor management;
- backfill;
- retry and dead-letter handling;
- freshness and completeness reporting.

Avoid embedding business metric logic inside connectors.

19.22. Build economics

The platform itself should be economically evaluated.

Track:

- engineering and operating cost;
- infrastructure and vendor cost;
- source onboarding cost;
- metric maintenance cost;
- experiment cycle time;
- realized or expected decision value;
- unused platform capabilities.

A comprehensive platform that does not change decisions is not successful.

19.23. Failure recovery

The program should be prepared to pause or reverse:

- a metric that creates gaming;
- a model with poor subgroup calibration;
- a policy that increases quality risk;
- an invasive data source;
- a dashboard used for an unapproved purpose;
- an experiment with guardrail breach.

Rollback should include technical, policy, communication, and data-retention actions.

19.24. Anti-patterns

19.24.1. Enterprise-wide launch first

The organization attempts universal coverage before any decision is improved.

19.24.2. Data platform as the product

Infrastructure milestones replace user and decision outcomes.

19.24.3. Dashboard proliferation

Every stakeholder receives a custom scorecard with inconsistent definitions.

19.24.4. Hidden quality problems

Coverage and freshness are omitted from executive views.

19.24.5. No local ownership

Central analysts interpret domain-specific workflows without practitioners.

19.24.6. Scaling before trust

Mandatory use expands after technical success but before governance and legitimacy exist.

19.25. A ninety-day pilot outline

19.25.1. Days 1–30

- finalize charter and safeguards;
- select pilot and intervention;
- connect core source systems;

- establish baseline event quality;
- define initial Engineering Change linkage;
- register metrics and experiment design.

19.25.2. Days 31–60

- run reconstruction and metrics in shadow mode;
- validate with practitioners;
- correct identity and linkage errors;
- complete power and readiness review;
- deliver pilot decision products.

19.25.3. Days 61–90

- launch bounded experiment or intervention;
- monitor exposure and guardrails;
- publish data quality and preliminary operational findings;
- record adoption, issues, and next-stage criteria.

Many outcome effects will require longer follow-up, especially rare quality events.

19.26. Worked example: selecting the first pilot



Runnable version: `companion/notebooks/20-pilot-selection.py`.

A newly funded intelligence team must choose one of four candidate pilots for the first ninety days. Each is scored against the pilot-selection criteria in this chapter:

- *Reviewer capacity in the payments organization.* Sponsor: engaged VP. Decision at stake: whether to add three senior reviewers or invest in AI review tooling. Data reachability: high — payments has clean source systems. Political cost of failure: moderate.
- *Deployment latency across the whole company.* Sponsor: interested but not accountable. Decision at stake: unclear. Data reachability: fragmented across six deployment systems. Political cost of failure: high, because the pilot would be visible enterprise-wide.
- *Incident response quality for site reliability.* Sponsor: strong SRE lead. Decision at stake: whether to invest in on-call reduction tooling. Data reachability: medium — needs joining incident data to on-call schedules. Political cost of failure: low.
- *AI-review evaluation.* Sponsor: platform lead. Decision at stake: rollout expansion. Data reachability: medium. Political cost of failure: moderate — this is one of several ongoing AI evaluations.

Scored on the "narrow, consequential, evidence-reachable, sponsored" rubric, the reviewer-capacity pilot ranks first. It has an accountable sponsor with a real capital decision, source systems already produce the required signal, the population is bounded to one organization, and success or failure will be visible to a supportive audience.

The team picks the reviewer-capacity pilot as the ninety-day target and lists the whole-company deployment-latency work as a "not yet" — the decision at stake is unclear, the political cost of failure is high, and the data path is not ready. That candidate returns to the roadmap for reconsideration after the ingestion layer stabilises.

Two months in, the pilot has produced a defensible cost model that shifts the payments leadership from adding three reviewers to adopting AI-assisted review — a $600k annual capital re-allocation supported by a governed metric and a documented analysis path.

19.27. Key takeaways

- Begin with a consequential decision, not universal instrumentation.
- Build the minimum trustworthy evidence path for a bounded pilot.
- Establish safeguards, identity, lineage, and data quality before advanced models.
- Scale only after proving decision value, trust, and repeatable onboarding.
- Use shadow mode and staged rollout to improve safety and calibration.
- Treat Engineering Intelligence as a product with operating costs and measurable value.
- Maintain rollback paths for metrics, models, policies, and data collection.

19.28. Open questions

- What pilot characteristics best predict successful enterprise adoption?
- How much platform architecture should be standardized before local pilots begin?
- Which forms of practitioner participation most improve trust and metric validity?
- How should platform value be attributed when benefits are distributed across many decisions?
- When should an organization stop investing in Engineering Intelligence infrastructure?

19.29. Implementation checklist

1. Select one high-value decision and willing pilot domain.
2. Publish charter, safeguards, and prohibited uses.
3. Build canonical events, raw evidence, identity, and lineage.
4. Establish Engineering Change linkage and quality measures.
5. Implement one small governed metric bundle.
6. Run in shadow mode and validate with practitioners.
7. Launch only when exposure and guardrails are observable.
8. Record decision impact and realized value.
9. Scale through repeatable domain onboarding and federation.
10. Maintain operating cadence, governance, and rollback procedures.

19.29.1. Research foundations and source notes

The implementation and rollout framework in this chapter draws on reliable delivery, site reliability engineering, staged experimentation, organizational learning, developer experience, and governance. It extends those foundations into a decision-first adoption model for Engineering Intelligence.

Incremental delivery

Continuous-delivery practices emphasize small batches, automated validation, deployment safety, and rapid feedback. See [humble2010] and [kim2021-devops].

The chapter therefore recommends:

- narrow initial scope;
- reversible changes;
- automated validation;
- shadow operation;
- progressive exposure;
- explicit rollback;
- short learning cycles.

A large platform launch that attempts to solve every measurement problem at once increases technical, organizational, and governance risk.

Reliability engineering

SRE principles concerning service-level objectives, error budgets, incident response, and operational toil are grounded in [beyer2016] and [beyer2018].

The rollout treats the Engineering Intelligence platform as a production system with objectives for:

- freshness;
- completeness;
- linkage quality;
- metric latency;
- correction propagation;
- serving availability;
- auditability.

Evidence integrity is as important as infrastructure availability because incorrect metrics can influence consequential decisions.

Pilot design

A pilot should test both technical feasibility and decision usefulness.

The causal and experimental foundations are described in [shadish2002], [imbens2015], [hernan2020], and [kohavi2020].

A credible pilot includes:

- a declared decision;
- baseline and comparator;
- eligible population;
- primary outcomes and guardrails;
- instrumentation quality checks;

- treatment and metric versions;
- stopping and expansion criteria.

A pilot is not successful merely because connectors run or dashboards render.

Shadow mode

Shadow operation allows a new reconstruction, metric, model, or policy to run without controlling production decisions.

This supports comparison of:

- old and new definitions;
- missed and additional links;
- score calibration;
- operational cost;
- disagreement cases;
- unintended policy effects.

Shadow mode is an Engineering Intelligence-specific rollout control informed by reliable-change and experimentation practices.

Progressive rollout

Progressive exposure limits consequence while evidence accumulates.

Approaches include:

- selected repositories or teams;
- low-risk Engineering Change classes;
- advisory-only output;
- restricted populations;
- stepped-wedge introduction;
- feature flags;
- explicit autonomy tiers.

Expansion should depend on evidence quality, usefulness, governance readiness, and guardrail performance rather than calendar commitment alone.

Organizational adoption

Developer-experience research emphasizes feedback loops, cognitive load, flow, and practitioner perception. See [forsgren2021-space], [noda2023-devex], [fagerholm2012], and [meyer2014].

The chapter therefore begins adoption with problems practitioners already experience, such as unreliable CI, review queues, ownership ambiguity, or repeated manual reporting.

Local usefulness creates a stronger basis for trust than executive ranking dashboards.

Complementary organizational change

Technology investment often requires changes in workflow, roles, incentives, and skills before productivity benefits are realized. See [brynjolfsson1993] and [brynjolfsson2017].

Implementation planning therefore includes:

- process redesign;
- training;
- governance capacity;
- ownership and support;
- integration with decision forums;
- value-realization follow-up.

Installing the platform is not the same as changing organizational capability.

Organizational learning

Argyris and Schön's double-loop learning supports reconsidering assumptions and policies, not only correcting execution. See [argyris1978].

Senge's systems-thinking perspective supports learning across technical and organizational boundaries. See [senge1990].

The rollout therefore preserves:

- pilot findings;
- null and negative results;
- disputes and corrections;
- invalid assumptions;
- retired metrics;
- decision outcomes.

Governance before scale

Measurement validity and indicator corruption provide the basis for establishing governance before broad adoption. See [messick1995], [campbell1979], and [goodhart1984].

The chapter requires early publication of:

- intended purposes;
- acceptable and prohibited uses;
- metric ownership;
- privacy controls;
- correction and appeal;
- retention;
- model and AI governance.

Scaling instrumentation before these boundaries exist can make later trust recovery difficult.

Privacy and security readiness

Privacy and cybersecurity foundations are provided by [nist2020-privacy] and [nist2024-csf].

Rollout readiness includes:

- purpose limitation;
- minimization;
- pseudonymization;
- least privilege;
- secrets management;
- audit logging;
- retention and deletion;
- incident response;
- vendor-risk review.

Detailed behavioral telemetry should not be retained indefinitely merely because it was useful during a pilot.

AI rollout

AI-enabled features require additional lifecycle governance informed by [nist2023-airmf], [nist2024-genai], 001, and 894.

The chapter recommends progressing from:

1. offline evaluation;
2. shadow evaluation;
3. advisory use;
4. bounded automation;
5. higher autonomy only after repeated evidence.

Model, prompt, retrieval, tool, and policy versions should remain traceable throughout rollout.

Operating model

A sustainable implementation requires explicit roles for:

- platform ownership;
- source stewardship;
- metric governance;
- experiment review;
- privacy and security;
- model governance;
- practitioner representation;
- executive decision ownership.

These responsibilities may be combined in smaller organizations, but accountability should remain visible.

Value realization

The chapter distinguishes deployment from realized value.

A rollout should track:

- adoption;
- workflow change;
- capacity released;
- capacity redeployed;
- quality and risk outcomes;
- avoided cost;
- realized financial or customer value.

The economic foundation is provided by [boardman2018] and the productivity-realization literature above.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- rollout organized around bounded decision products rather than connector count;
- evidence-quality, governance, and trust gates for expansion;
- shadow comparison for reconstruction, metrics, models, and policies;
- progressive autonomy tiers for engineering agents;
- explicit transition from pilot evidence to value realization;
- correction and dispute rates used as rollout-learning signals;
- semantic SLOs and evidence incidents included in operational readiness;
- practitioner-local value required before executive-scale reporting;
- maturity based on reproducibility and governance, not dashboard breadth.

These practices should be adapted to organizational context and evaluated through reliability, adoption quality, practitioner trust, governance outcomes, and realized decision value.

Chapter 20. The Science of Engineering Intelligence



Engineering Intelligence becomes a discipline only when its claims can be tested, reproduced, *challenged, and improved. This chapter defines the scientific foundations of the field: its objects of study, theory-building practices, measurement standards, replication requirements, benchmark design, shared datasets, reporting conventions, and research agenda. It argues that engineering organizations should not merely consume metrics, but participate in a cumulative science of software delivery.*

20.1. From analytics practice to scientific discipline

Engineering analytics is often local and ephemeral. A team builds a dashboard, answers an immediate question, and then moves on. Definitions remain undocumented, causal assumptions remain implicit, and results cannot be compared across organizations.

A scientific discipline requires more.

It needs:

- explicit constructs;
- observable units of analysis;
- falsifiable hypotheses;
- reliable measurement;

- transparent uncertainty;
- reproducible methods;
- cumulative evidence;
- mechanisms for correction.

Engineering Intelligence should aspire to these properties while recognizing that software delivery is a changing socio-technical system rather than a controlled physical environment.

20.2. The object of study

The primary object of study is not the individual engineer. It is the production and operation of Engineering Changes within an organizational system.

An Engineering Change connects:

- an intended outcome;
- human and machine participants;
- code, configuration, data, and infrastructure artifacts;
- review and validation activity;
- deployment and release mechanisms;
- production consequences;
- economic cost and value;
- organizational and architectural context.

This unit permits analysis across tools and workflow implementations while reducing the temptation to equate visible activity with individual performance.

20.3. Levels of explanation

Engineering Intelligence requires explanations at multiple levels.

20.3.1. Artifact level

Properties of code, tests, configuration, models, schemas, and infrastructure definitions.

20.3.2. Change level

The complete sequence through which an intended modification is designed, implemented, reviewed, validated, deployed, and observed.

20.3.3. Queue and workflow level

How work competes for reviewers, CI capacity, environments, approvals, and release windows.

20.3.4. Team level

Coordination structure, expertise distribution, ownership, interruptions, and local operating practices.

20.3.5. Architecture level

Dependencies, coupling, interfaces, blast radius, and the constraints imposed by the software system.

20.3.6. Portfolio level

Allocation of engineering capacity across products, platforms, risk reduction, modernization, compliance, and exploration.

20.3.7. Enterprise level

Capital allocation, governance, talent systems, vendor strategy, and the aggregate relationship between engineering investment and organizational outcomes.

A useful theory should state the level at which it operates and the mechanisms that connect it to other levels.

20.4. Constructs and observables

Many important engineering concepts cannot be observed directly.

Examples include:

- cognitive load;
- change complexity;
- review quality;
- coordination cost;
- architectural risk;
- developer experience;
- organizational learning;
- engineering value.

These are constructs. They must be represented through observable indicators, and those indicators must be validated.

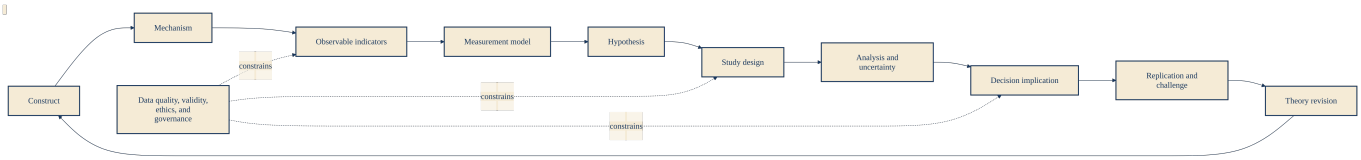
For example, cognitive load might be estimated through interruption frequency, concurrent work, context switching, review span, survey responses, and task-reconstruction studies. No single signal should automatically be treated as the construct itself.

20.5. A theory-to-evidence chain

Every important claim should be traceable through a chain:

1. construct;
2. proposed mechanism;
3. observable indicators;
4. measurement model;
5. hypothesis;
6. study design;
7. analysis;
8. decision implication;

9. replication status.



This chain discourages the common pattern in which a convenient telemetry field becomes a metric, the metric becomes a target, and the target becomes an unsupported claim about performance.

20.6. Falsifiable hypotheses

A hypothesis should describe an expected relationship that evidence could contradict.

Weak statement:

> AI code review improves engineering productivity.

Stronger statement:

> For eligible medium-risk pull requests, enabling the AI reviewer will reduce median human review effort by at least 15 percent without increasing escaped-severity-one-or-two defects, after controlling for repository and change complexity.

The stronger statement defines:

- population;
- treatment;
- expected effect;
- outcome;
- guardrail;
- comparison;
- adjustment variables.

Falsifiability forces clarity before data is examined.

20.7. Descriptive, predictive, causal, and prescriptive claims

Claims must be labeled by type.

20.7.1. Descriptive

What happened?

Example: review wait time increased during the quarter.

20.7.2. Predictive

What is likely to happen?

Example: this change has a high probability of requiring rollback.

20.7.3. Causal

What would happen under an intervention?

Example: introducing a merge queue reduced integration failures.

20.7.4. Prescriptive

What action should be taken given objectives and constraints?

Example: route this change to a security specialist and delay deployment until validation completes.

A predictive model does not establish that acting on its prediction will improve outcomes. A correlation does not establish an intervention effect. A prescriptive recommendation requires explicit objectives, costs, and guardrails.

20.8. Measurement validity

A scientifically useful metric requires several forms of validity.

20.8.1. Construct validity

Does the metric represent the intended concept?

20.8.2. Content validity

Does it cover the important dimensions of that concept?

20.8.3. Criterion validity

Does it agree with a credible external reference or outcome?

20.8.4. Convergent validity

Do different measures of the same construct move together where expected?

20.8.5. Discriminant validity

Can the metric distinguish the construct from related but different concepts?

20.8.6. Ecological validity

Does the measure remain meaningful in real engineering conditions?

A metric can be computed correctly and still be scientifically invalid.

20.9. Reliability and repeatability

Reliability asks whether measurement is stable under equivalent conditions.

Sources of unreliability include:

- inconsistent source events;
- identity-resolution errors;
- changing workflow semantics;
- annotation disagreement;
- unstable model inference;
- non-deterministic reconstruction;
- incomplete deployment linkage;
- hidden metric-version changes.

Reliability should be reported using suitable methods such as test-retest stability, inter-rater agreement, internal consistency, reconstruction agreement, and confidence intervals.

20.10. Measurement invariance

A metric may behave differently across languages, repositories, teams, business domains, or workflow types.

Before comparing groups, ask whether the measure has equivalent meaning.

For example, pull-request size may be a reasonable complexity indicator in one codebase and nearly meaningless in a generated-code repository. Review count may represent diligence in one workflow and mandatory approval policy in another.

Cross-group comparison requires evidence of measurement invariance or explicit qualification.

20.11. Calibration

Risk scores and probability estimates should be calibrated.

If a model assigns 0.30 defect probability to many changes, approximately 30 percent of comparable changes should exhibit the defined outcome over time.

Calibration should be assessed:

- globally;
- by repository and domain;
- by complexity band;
- by model version;
- by important subpopulation;
- under distribution shift.

A highly ranked model can still be operationally misleading when its probabilities are poorly calibrated.

20.12. Ground truth is constructed

Engineering datasets rarely contain perfect truth.

Defect labels may depend on:

- whether an incident was detected;
- whether it was linked to the correct change;
- how severity was assigned;
- whether a rollback represented failure or precaution;
- whether a reviewer recognized the issue;
- whether the organization records near misses.

Ground-truth creation should document adjudication rules, uncertainty, disagreement, missing outcomes, and label provenance.

20.13. Annotation science

Human labeling is itself a measurement process.

A rigorous annotation protocol should specify:

- label ontology;
- inclusion and exclusion rules;
- examples and counterexamples;
- annotator qualifications;
- blinded or non-blinded conditions;
- calibration exercises;
- disagreement resolution;
- confidence capture;
- periodic drift review.

For difficult judgments, preserve annotator distributions rather than collapsing all disagreement into a single authoritative label.

20.14. Benchmark design

Benchmarks can accelerate progress, but they can also distort it.

A useful Engineering Intelligence benchmark should include:

- a clearly defined task;
- representative populations;
- realistic class imbalance;
- temporal splits;
- repository separation where appropriate;
- leakage controls;
- uncertainty and annotation metadata;

- cost-sensitive evaluation;
- subgroup reporting;
- a protected holdout set.

Benchmarks should reward useful operational behavior, not merely leaderboard optimization.

20.15. Retrieval and ranking evaluation

Many Engineering Intelligence systems rank findings, changes, experts, dependencies, or interventions.

Evaluation should therefore include ranking measures such as:

- precision at k;
- recall at k;
- mean reciprocal rank;
- normalized discounted cumulative gain;
- average precision;
- expected utility at review budget;
- severe-finding recall;
- time-to-first-actionable result.

A system that finds the right issue but buries it below twenty low-value comments may be technically correct and operationally ineffective.

20.16. Economic evaluation of models

Model quality should be translated into consequences.

Relevant quantities include:

- expert minutes consumed;
- false-positive validation cost;
- defect cost avoided;
- latency introduced;
- infrastructure and inference cost;
- opportunity cost;
- adoption effects;
- risk concentration.

The best statistical model is not necessarily the best engineering intervention.

20.17. Internal replication

Before broad rollout, an organization should replicate important findings across:

- multiple teams;
- different repositories;

- time periods;
- workflow variants;
- complexity strata;
- organizational boundaries.

Internal replication tests whether a result is local, seasonal, policy-dependent, or robust enough to guide enterprise decisions.

20.18. External replication

Cross-organization replication is difficult because data and workflows differ, but it is essential for a cumulative field.

External studies should distinguish:

- exact replication of method;
- conceptual replication of mechanism;
- transportability to a new population;
- adaptation to a different workflow;
- failure to reproduce.

Non-replication should be treated as evidence about boundary conditions, not merely as embarrassment.

20.19. Reproducibility packages

A publishable Engineering Intelligence study should provide, where privacy and security permit:

- research question and hypotheses;
- analysis plan;
- metric definitions and versions;
- event and entity schemas;
- code or executable specifications;
- data-quality report;
- sample construction logic;
- statistical model details;
- robustness checks;
- synthetic or de-identified example data;
- decision record.

Where raw data cannot be shared, organizations can still publish transformation logic, schemas, synthetic fixtures, and reproducible aggregate analyzes.

20.20. Pre-registration

For consequential experiments, pre-register:

- hypotheses;
- primary and secondary outcomes;

- guardrails;
- eligibility;
- assignment;
- exposure definition;
- exclusion rules;
- sample-size rationale;
- stopping criteria;
- analysis model;
- subgroup analyzes.

Pre-registration does not prevent exploration. It distinguishes confirmatory evidence from discoveries made after inspecting the data.

20.21. Registered reports

A stronger practice is to review the study question and method before results are known.

Registered reports reduce publication bias and encourage organizations to value well-designed negative findings. The result becomes informative because the method was credible, not because the effect was exciting.

20.22. Negative results

Engineering organizations often suppress negative results:

- a tool produced no measurable benefit;
- an intervention helped only one workflow;
- a model failed under real operational load;
- a metric could not be validated;
- an expected effect disappeared after adjustment.

These findings prevent repeated waste. A mature discipline records them.

20.23. Meta-analysis

As studies accumulate, meta-analysis can estimate effect distributions rather than relying on one headline number.

For example, the effect of AI review on human effort may vary by:

- programming language;
- change complexity;
- reviewer expertise;
- repository maturity;
- model family;
- prompting strategy;
- organizational policy.

A random-effects view is often more realistic than assuming one universal effect.

20.24. Transportability

Decision makers must ask whether evidence from one setting applies to another.

Transportability depends on similarities in:

- population;
- architecture;
- workflow;
- incentives;
- tooling;
- risk tolerance;
- outcome definition;
- intervention fidelity.

The burden is not to prove that environments are identical, but to identify which differences could alter the causal mechanism.

20.25. Natural experiments

Engineering organizations continually produce quasi-experimental variation:

- phased tool rollouts;
- policy thresholds;
- infrastructure outages;
- team migrations;
- vendor changes;
- repository transfers;
- staggered platform adoption.

With careful assumptions, methods such as difference-in-differences, regression discontinuity, interrupted time series, and instrumental variables can extract useful evidence.

Natural experiments should not be presented as randomized trials. Their identifying assumptions must be explicit and tested where possible.

20.26. Longitudinal research

Many effects appear only over time.

Examples include:

- skill development or atrophy;
- architectural degradation;
- review-culture change;
- accumulated automation debt;
- model adaptation;
- organizational learning;

- delayed quality outcomes.

Longitudinal designs should distinguish short-term novelty effects from sustained change.

20.27. System adaptation

Engineering systems react to measurement and intervention.

People learn the metric. Teams alter workflow. Models change the distribution of work. Policies redirect attention. Vendors update products.

This means an intervention effect is not permanently fixed. Engineering Intelligence requires continuous revalidation and explicit study of adaptation.

20.28. Multi-method research

Telemetry alone cannot explain every mechanism.

Strong studies may combine:

- event data;
- controlled experiments;
- repository mining;
- surveys;
- interviews;
- cognitive walkthroughs;
- diary studies;
- observational fieldwork;
- incident review;
- economic modeling.

Qualitative evidence is especially valuable for discovering mechanisms, interpreting anomalies, and identifying constructs not captured by instrumentation.

20.29. Shared research infrastructure

A cumulative discipline benefits from shared infrastructure:

- canonical event contracts;
- Engineering Change schemas;
- metric definition registries;
- benchmark task specifications;
- synthetic data generators;
- experiment templates;
- analysis libraries;
- privacy-preserving collaboration methods;
- replication registries;

- evidence maps.

Standardization should support comparison without freezing local innovation.

20.30. Privacy-preserving collaboration

Organizations may collaborate without exchanging raw proprietary data.

Possible approaches include:

- federated analysis;
- secure aggregation;
- differential privacy;
- common query packages executed locally;
- standardized aggregate reporting;
- synthetic datasets;
- trusted research enclaves.

Privacy protection must be evaluated against re-identification risk, especially for small teams and rare events.

20.31. Evidence maturity levels

Claims can be classified by maturity.

20.31.1. Level 0: anecdote

A plausible observation without systematic measurement.

20.31.2. Level 1: descriptive evidence

A measured pattern with documented definitions and data quality.

20.31.3. Level 2: replicated association

A relationship observed across populations or periods with relevant adjustment.

20.31.4. Level 3: credible causal evidence

A randomized or defensible quasi-experimental estimate.

20.31.5. Level 4: replicated causal mechanism

Consistent causal evidence with understood boundary conditions.

20.31.6. Level 5: operationally validated policy

A mechanism translated into a governed intervention with durable net benefit.

The maturity level should accompany important recommendations.

20.32. Scientific reporting standard

Every major result should state:

- question;
- claim type;
- population;
- unit of analysis;
- period;
- treatment or exposure;
- outcome definition;
- metric versions;
- data coverage;
- uncertainty;
- design assumptions;
- robustness checks;
- limitations;
- replication status;
- decision consequence.

This reporting standard makes results easier to audit and compare.

20.33. An open evidence graph

Over time, studies can be represented as an evidence graph connecting:

- constructs;
- metrics;
- datasets;
- interventions;
- populations;
- mechanisms;
- findings;
- replications;
- contradictions;
- decisions.

Such a graph would allow decision makers to inspect not only the latest result, but the complete provenance and maturity of a claim.

20.34. Research agenda

Important research areas include:

20.34.1. Engineering Change reconstruction

How accurately can changes be reconstructed across fragmented enterprise systems?

20.34.2. Cognitive load

Which behavioral and self-reported measures best predict harmful overload without becoming surveillance instruments?

20.34.3. AI collaboration

When do AI participants substitute for, complement, or increase human effort?

20.34.4. Review science

What review structures maximize defect discovery, knowledge transfer, and attention efficiency?

20.34.5. Architecture and organizational structure

How do dependency topology, ownership, and team design interact?

20.34.6. Value attribution

How can engineering investment be linked to product, reliability, and strategic outcomes without false precision?

20.34.7. Metric governance

Which institutional practices prevent gaming while preserving useful accountability?

20.34.8. Long-run capability

How do automation and AI affect learning, expertise, and resilience over years rather than weeks?

20.35. Institutional design for research

Large organizations should create mechanisms for cumulative learning:

- a research review board for consequential studies;
- a registry of experiments and natural experiments;
- standard evidence packages;
- reusable benchmark datasets;
- protected publication of negative findings;
- practitioner and domain review;
- periodic synthesis of accumulated evidence.

This need not become academic bureaucracy. Its purpose is to reduce repeated weak studies and unsupported enterprise decisions.

20.36. Anti-patterns

20.36.1. Dashboard empiricism

A visible pattern is treated as explanation.

20.36.2. Metric essentialism

A proxy is treated as the underlying construct.

20.36.3. One-company universalism

A local result is presented as a law of software engineering.

20.36.4. Benchmark overfitting

Systems optimize a public test while losing operational usefulness.

20.36.5. Causal language without causal design

Associations are described as improvements caused by an intervention.

20.36.6. Hidden researcher degrees of freedom

Outcomes, exclusions, and models are selected after results are visible.

20.36.7. Publication of winners only

Negative and contradictory findings disappear.

20.36.8. Permanent validity

A metric or model is assumed to remain valid after the system adapts.

20.37. Key takeaways

- Engineering Intelligence should develop as a cumulative empirical discipline, not a collection of dashboards.
- The Engineering Change is the primary unit of study, embedded in workflow, architecture, organizational, and economic context.
- Constructs must be separated from their observable indicators and validated explicitly.
- Claims must distinguish description, prediction, causation, and prescription.
- Reproducibility, replication, negative results, and transportability are essential.
- Benchmarks must represent operational utility, uncertainty, cost, and realistic populations.

- Scientific findings require continuous revalidation because engineering systems adapt.
- Shared schemas, methods, and privacy-preserving collaboration can create an evidence base larger than any one organization.

20.38. Open questions

- Which constructs should become foundational standards for Engineering Intelligence?
- What minimum evidence package should be required before an enterprise metric influences policy?
- How can organizations share replication results without exposing proprietary systems or employee data?
- Which engineering effects are stable enough to generalize across companies?
- How should contradictory evidence be represented in executive decision products?
- Can an open evidence graph become a practical foundation for cumulative engineering science?

20.39. Scientific practice checklist

1. State the construct, mechanism, population, and unit of analysis.
2. Label the claim as descriptive, predictive, causal, or prescriptive.
3. Version metrics, data, models, and analysis logic.
4. Report coverage, missingness, reliability, validity, and uncertainty.
5. Pre-register consequential confirmatory studies.
6. Preserve exploratory analysis as exploratory.
7. Test calibration, subgroup behavior, and transportability.
8. Replicate before broad policy adoption.
9. Publish negative and contradictory results internally.
10. Revalidate after workflow, model, or incentive changes.
11. Package studies for reproduction without exposing sensitive data.
12. Connect findings to an explicit evidence maturity level and decision record.

20.39.1. Research foundations and source notes

The scientific framework in this chapter draws on measurement theory, causal inference, reproducibility, open science, cumulative evidence, and organizational learning. It extends those foundations into a research discipline for studying software engineering systems in production.

Measurement theory

The distinction among nominal, ordinal, interval, and ratio scales originates with [stevens1946].

Construct validity is grounded in [messick1995], while broader concerns about how measurement choices shape interpretation are discussed in [hand2004].

The chapter therefore requires every scientific claim to state:

- the construct;
- the observed indicators;

- the scale and unit;
- the eligible population;
- the intended interpretation;
- the permitted use;
- known limitations.

A numerical output is not automatically a valid measure of the underlying construct.

Reliability

Reliability concerns the consistency of a measurement process.

The chapter draws on [cronbach1951] to distinguish repeatability from validity.

Relevant reliability questions include:

- Would the same source evidence produce the same metric version?
- Would independent labelers agree?
- Would reconstruction logic produce stable linkage?
- Would the result remain similar under repeated measurement where stability is expected?

A reliable measure can still be invalid if it consistently measures the wrong thing.

Causal inference

The chapter's causal foundation comes from [rubin1974], [holland1986], [pearl2009], [imbens2015], and [hernan2020].

These traditions support explicit separation among:

- observed association;
- prediction;
- intervention effect;
- mediation;
- mechanism;
- transportability.

Engineering Intelligence should not describe a temporal correlation as causal merely because the explanation is plausible.

Experimental design

The design-first approach is informed by [shadish2002], [angrist2009], and [kohavi2020].

Scientific practice therefore includes:

- declared estimands;
- assignment and analysis units;
- primary outcomes and guardrails;
- power analysis;

- missing-data plans;
- treatment-version control;
- stopping rules;
- reproducible analysis.

The research question and decision should determine the design rather than the available dashboard determining the question.

Preregistration

Preregistration and pre-analysis planning help distinguish confirmatory analysis from exploration. See [nosek2018].

The chapter therefore recommends registering:

- hypotheses;
- causal assumptions;
- outcomes;
- exclusions;
- subgroup analyzes;
- analysis methods;
- decision thresholds.

Exploratory analysis remains valuable, but its evidential status should be reported honestly.

Reproducibility

The reproducibility concerns illustrated by [open-science2015] support retaining enough evidence and computation to recreate published results.

The chapter requires preservation of:

- source snapshots or durable references;
- schemas and transformations;
- identity and reconstruction versions;
- metric and model versions;
- experiment assignment and exposure;
- analysis code;
- generated tables and figures;
- deviations and corrections.

A result that can be viewed but not reconstructed is weak scientific evidence.

FAIR evidence stewardship

The FAIR principles emphasize making research objects findable, accessible, interoperable, and reusable. See [wilkinson2016].

In an enterprise context, accessibility remains subject to privacy, security, and contractual constraints.

The chapter therefore interprets FAIR as governed discoverability and reuse with metadata, provenance, identifiers, and access controls.

Replication

Replication tests whether a finding persists across teams, repositories, task classes, time periods, and treatment versions.

A result from one group or one model version should not become universal policy automatically.

The chapter distinguishes:

- exact computational reproduction;
- direct replication;
- conceptual replication;
- operational replication in a new context.

Contradictory replication results are evidence about boundary conditions rather than failures to be hidden.

Cumulative evidence

Individual studies should contribute to a cumulative evidence base.

The chapter therefore records:

- intervention and comparator;
- population;
- context;
- effect estimate and uncertainty;
- treatment version;
- guardrails;
- study quality;
- replication status;
- realized outcome.

This enables later synthesis across studies rather than repeated isolated pilots.

Heterogeneity

Engineering effects are likely to vary by language, architecture, team structure, task complexity, developer experience, and organizational context.

The chapter therefore treats heterogeneity as scientifically important rather than as noise to average away.

Subgroup findings should be predeclared where possible, adequately powered, and replicated before policy use.

External validity

A causal result is local to its population, treatment, comparator, period, and operating conditions.

The internal- and external-validity tradition in [shadish2002] supports the chapter's requirement to document boundary conditions.

Transporting a result requires evidence that the relevant causal mechanisms and populations are sufficiently similar.

Model evaluation

Models used for reconstruction, classification, ranking, prediction, or routing require evaluation appropriate to their function.

Relevant foundations include precision and recall from [manning2008] and ranked relevance from [jarvelin2002].

The chapter also requires:

- calibration;
- subgroup performance;
- uncertainty;
- drift monitoring;
- abstention behavior;
- downstream decision impact.

Offline predictive performance alone is insufficient when model output changes human behavior or policy.

Negative and null results

Scientific practice is distorted when only positive findings are retained.

The chapter therefore treats null, negative, and contradictory outcomes as first-class evidence.

These results can reveal:

- interventions that do not generalize;
- hidden costs;
- weak mechanisms;
- boundary conditions;
- invalid metrics;
- insufficient power;
- implementation failure.

Preserving these findings reduces repeated waste and supports cumulative learning.

Measurement drift

Source systems, workflows, models, and organizational structures change over time.

A metric can continue computing successfully while its meaning changes.

The chapter therefore requires:

- semantic versioning;

- drift monitoring;
- periodic revalidation;
- correction and replay;
- retirement of invalid measures.

This extends ordinary data-drift practice into construct and policy drift.

Research ethics

Engineering research may involve practitioner behavior, sensitive telemetry, and consequential AI systems.

Relevant foundations include [nist2020-privacy], [nist2023-airmf], [nist2024-genai], and [acm2018-ethics].

The chapter therefore requires:

- purpose limitation;
- minimization;
- proportionality;
- transparent governance;
- human accountability;
- restrictions on punitive individual use;
- retention and deletion plans.

Scientific value does not override privacy, fairness, or professional responsibility.

Open science within enterprise constraints

Organizations may be unable to publish raw source code, telemetry, or personnel data.

The chapter therefore promotes layered openness:

- publish definitions and protocols where possible;
- share synthetic or de-identified fixtures;
- preserve internal reproducibility;
- publish aggregate findings and limitations;
- enable independent internal review;
- document unavailable evidence honestly.

This is an Engineering Intelligence-specific interpretation of open-science principles for enterprise settings.

Organizational learning

Argyris and Schön's double-loop learning supports questioning the assumptions behind measurement and policy. See [argyris1978].

Senge's systems-thinking perspective supports learning across technical and organizational boundaries. See [senge1990].

The scientific program should therefore review not only whether an intervention worked, but whether the original causal model and decision process were sound.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- Engineering Intelligence as a cumulative applied science of software engineering systems;
- Engineering Change as a common observational and experimental unit;
- explicit evidence states for observed, reconstructed, inferred, predicted, and causal claims;
- treatment-version drift as a scientific validity threat;
- integrated metric, model, experiment, evidence, and decision registries;
- outcome maturity as part of evidential status;
- enterprise-compatible open science;
- scientific retention of null, negative, and contradictory findings;
- replication across Engineering Change classes before policy generalization;
- post-decision study of realized value and unintended consequence.

These contributions should be evaluated by reproducibility, replication success, forecast calibration, decision quality, correction history, and the organization's ability to learn from accumulated evidence.

Chapter 21. The Learning Engineering Organization



The ultimate purpose of Engineering Intelligence is not measurement itself. It is to create an *organization that can learn deliberately: observe its engineering system, form credible explanations, test interventions, retain what works, abandon what does not, and improve its ability to make future decisions. This chapter brings together the economic, human, architectural, experimental, technical, and governance ideas developed throughout the book. It describes the operating model of a learning engineering organization and the capabilities required to sustain it.*

21.1. Measurement is not the destination

An organization may possess excellent telemetry and still fail to learn.

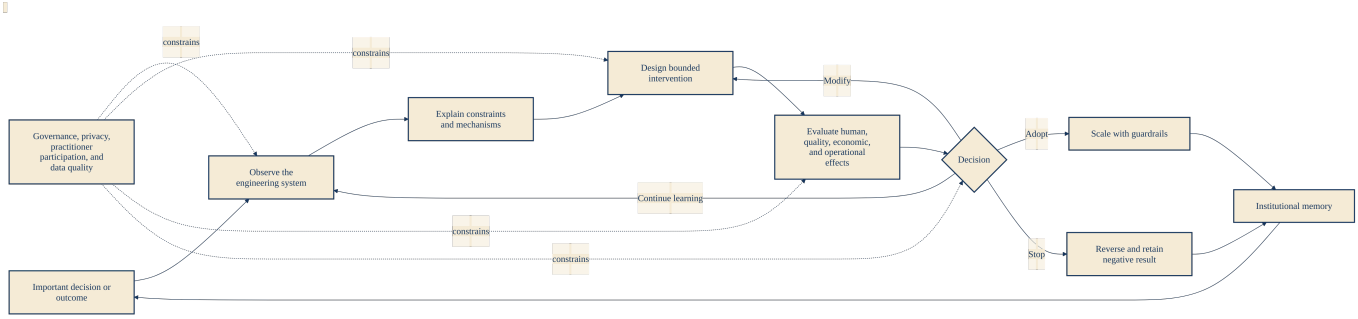
It can collect events without constructing useful evidence. It can compute metrics without changing decisions. It can run experiments without retaining the results. It can deploy AI tools without understanding whether they release capacity or create new validation work. It can build executive dashboards while local teams continue to experience the same queues, interruptions, and architectural constraints.

Engineering Intelligence succeeds only when evidence changes action and the consequences of that action become new evidence.

21.2. The organizational learning loop

A learning engineering organization repeatedly performs seven activities:

1. identify an important outcome or decision;
2. describe the current system and its constraints;
3. form a theory about what limits performance;
4. design a bounded intervention;
5. observe operational, human, quality, and economic effects;
6. decide whether to adopt, modify, or stop;
7. preserve the result as institutional knowledge.



The loop is not a one-time transformation program. It is an operating capability.

21.3. Observation

Observation begins with trustworthy evidence.

The organization must be able to reconstruct Engineering Changes across planning, source control, review, CI, deployment, production, and business systems. It must know the coverage and confidence of that reconstruction. It must distinguish event time from processing time, source absence from true inactivity, and human action from automated action.

Observation also includes evidence that cannot be obtained from telemetry alone:

- practitioner interviews;
- surveys and experience sampling;
- incident narratives;
- architecture reviews;
- workflow walkthroughs;
- financial and product context;
- explicit reports of friction and cognitive load.

The goal is not total visibility. It is sufficient visibility to support a decision without pretending away uncertainty.

21.4. Explanation

A metric identifies a pattern. Explanation identifies a mechanism.

A rise in cycle time may result from:

- larger or riskier changes;
- review queue congestion;
- CI resource contention;
- unclear ownership;
- unavailable test environments;
- increased specialist approval requirements;
- fragmented work and interruption;
- architectural coupling;
- delayed product decisions.

Interventions aimed at the wrong mechanism may improve the dashboard while worsening the engineering system.

Explanation therefore requires combining quantitative evidence with local expertise and an explicit causal model.

21.5. Intervention

A learning organization favors interventions that are:

- bounded;
- reversible;
- observable;
- ethically acceptable;
- economically plausible;
- linked to a mechanism;
- protected by guardrails.

Examples include changing reviewer routing, adding a merge queue, reducing mandatory approvals for low-risk work, improving CI caching, introducing an AI assistant, restructuring ownership, or decomposing a high-coupling interface.

The intervention should be described precisely enough that another team could reproduce it.

21.6. Evaluation

Evaluation should answer more than whether the primary metric moved.

It should examine:

- intended outcome;
- quality and safety guardrails;
- human effort and attention;
- distribution across teams and work types;
- direct and indirect cost;
- adoption and intervention fidelity;
- short-term and delayed effects;
- behavioral adaptation;

- uncertainty and alternative explanations.

An intervention that improves median throughput while concentrating risk in a small set of teams is not an unqualified success.

21.7. Decision

The purpose of evaluation is a decision.

Possible decisions include:

- adopt broadly;
- adopt only for a defined population;
- continue gathering evidence;
- modify the intervention;
- replace it with an alternative;
- stop and reverse;
- preserve the result without immediate action.

A decision record should state the evidence, assumptions, uncertainty, affected population, expected value, guardrails, owner, and review date.

21.8. Institutional memory

Organizations frequently repeat studies because prior learning is difficult to find.

Institutional memory should retain:

- the question asked;
- the system state at the time;
- metric and model versions;
- intervention design;
- results and limitations;
- negative findings;
- practitioner interpretation;
- decision and follow-up;
- conditions under which the conclusion may no longer hold.

This memory should be searchable by construct, system, intervention, outcome, and decision rather than stored only as presentation files.

21.9. The five capabilities

A learning engineering organization requires five mutually reinforcing capabilities.

21.9.1. Evidence capability

The ability to collect, link, govern, and interpret trustworthy engineering evidence.

21.9.2. Experimental capability

The ability to test workflow, tool, architecture, and policy interventions credibly.

21.9.3. Decision capability

The ability to translate evidence into explicit choices under uncertainty.

21.9.4. Adaptation capability

The ability to change tools, processes, architecture, and allocation based on results.

21.9.5. Memory capability

The ability to preserve and retrieve what the organization has learned.

Weakness in any one capability breaks the loop.

21.10. Local autonomy and shared foundations

Learning must occur close to the work, but it cannot depend entirely on isolated local systems.

Central foundations should provide:

- canonical event contracts;
- identity and temporal ownership;
- Engineering Change reconstruction;
- lineage and data-quality controls;
- experiment assignment and exposure;
- governed metric definitions;
- privacy and access policy;
- reusable analysis methods;
- an evidence registry.

Domain teams should provide:

- workflow semantics;
- local hypotheses;
- intervention ownership;
- interpretation;
- validation of reconstructed evidence;
- adoption and follow-through.

This federated model avoids both centralized misinterpretation and duplicated local infrastructure.

21.11. Leadership behavior

Leadership determines whether Engineering Intelligence becomes a learning system or a surveillance system.

Leaders should ask:

- What decision will this metric improve?
- What mechanism do we believe is operating?
- What evidence could change our minds?
- What uncertainty and missingness remain?
- Who bears the cost or risk of this intervention?
- Has the result replicated?
- When will we review or retire this policy?

They should resist asking which team or engineer produced the largest number of visible artifacts.

21.12. Practitioner participation

Practitioners are not merely subjects of measurement. They are essential interpreters and co-designers.

They understand:

- hidden manual work;
- exceptions in workflow;
- why apparently idle time exists;
- where automation creates validation burden;
- which approvals are substantive or ceremonial;
- how architecture constrains change;
- which metrics will be gamed;
- what interventions are operationally feasible.

Participation improves construct validity, intervention design, trust, and adoption.

21.13. Learning from incidents

Incidents are especially rich learning opportunities because they expose interactions among code, architecture, process, ownership, and attention.

A mature incident analysis should connect:

- precursor Engineering Changes;
- review and validation evidence;
- risk predictions;
- dependency and blast-radius structure;
- deployment sequence;
- detection and response;
- prior related incidents;
- organizational and economic consequences;

- corrective interventions and their later effectiveness.

The purpose is not to assign individual blame, but to update the organization’s model of its system.

21.14. Learning from successful changes

Failure receives disproportionate attention.

Successful high-risk changes can reveal:

- effective review patterns;
- useful decomposition strategies;
- strong ownership models;
- valuable test practices;
- resilient deployment mechanisms;
- appropriate use of AI assistance;
- conditions that enabled rapid recovery.

Studying success helps distinguish robust capability from mere absence of observed failure.

21.15. Learning across teams

Copying a practice from one team to another is not the same as transferring learning.

A practice carries assumptions about:

- architecture;
- team size;
- expertise;
- delivery frequency;
- regulatory environment;
- failure tolerance;
- testability;
- dependency structure;
- product maturity.

Cross-team learning should therefore transfer the mechanism and boundary conditions, not only the visible ritual.

21.16. Learning across time horizons

Engineering decisions operate on different horizons.

21.16.1. Immediate

Queue health, failed builds, incident response, and active change risk.

21.16.2. Tactical

Tool configuration, review policy, CI capacity, and team workflow.

21.16.3. Strategic

Architecture, platform investment, vendor adoption, workforce capability, and portfolio allocation.

21.16.4. Institutional

Governance, accumulated knowledge, incentives, and the organization's long-run capacity to adapt.

A learning system must connect these horizons. Immediate optimization should not quietly damage strategic capability.

21.17. AI as participant and subject

AI systems are both participants in engineering work and subjects of evaluation.

As participants, they may generate code, tests, reviews, designs, documentation, queries, and operational actions.

As subjects, they must be measured for:

- task success;
- error severity;
- validation burden;
- ranking quality;
- latency;
- cost;
- calibration;
- change in human behavior;
- long-term skill effects;
- security and governance risk.

A learning organization does not ask only whether engineers use AI. It asks how AI changes the production function of engineering and under which conditions that change creates durable value.

21.18. The changing role of the engineer

As automation expands, engineering work shifts rather than disappears.

More effort may move toward:

- problem formulation;
- architecture;
- constraint definition;
- validation;
- integration;
- risk management;

- model and tool orchestration;
- interpretation of ambiguous outcomes;
- stewardship of complex systems.

Engineering Intelligence should detect these shifts rather than interpreting fewer manual artifacts as reduced contribution.

21.19. Capability preservation

Automation can improve output while weakening latent capability.

Organizations should monitor whether they retain the ability to:

- diagnose failures without a vendor;
- review critical changes deeply;
- understand generated code;
- operate during model or tool outages;
- maintain architecture knowledge;
- train less-experienced engineers;
- challenge automated recommendations.

Resilience requires preserving critical human capability even when routine work is automated.

21.20. Incentives and truth

Measurement systems influence behavior.

When metrics affect status, funding, performance reviews, or headcount, people rationally optimize them. This can corrupt the evidence needed for learning.

A learning organization protects truth by:

- limiting high-stakes use of weak proxies;
- separating exploratory metrics from formal accountability;
- publishing definitions and limitations;
- auditing for gaming and displacement;
- using multiple forms of evidence;
- providing correction and appeal;
- retiring metrics that distort behavior.

Learning depends on maintaining conditions in which inconvenient evidence can survive.

21.21. Economic discipline

Learning must include economic reasoning.

For each intervention, estimate:

- implementation cost;
- operating cost;

- human attention consumed;
- capacity released;
- quality benefit or risk;
- time to value;
- reversibility;
- option value;
- cost of delay;
- expected value under uncertainty.

Not every statistically significant improvement is worth implementing. Not every strategically important investment will show immediate local return.

21.22. Architecture as accumulated decision history

Architecture reflects prior technical and organizational decisions.

Engineering Intelligence can make this history more visible by relating dependency structure and change outcomes to:

- ownership changes;
- platform migrations;
- interface decisions;
- reliability investments;
- repeated coordination cost;
- defect concentration;
- delivery friction.

Architecture then becomes a subject of empirical learning rather than periodic opinion alone.

21.23. Learning portfolio

An organization should manage a portfolio of learning activities.

The portfolio may include:

- low-cost operational experiments;
- tool evaluations;
- architecture pilots;
- policy changes;
- observational studies;
- longitudinal capability research;
- high-risk strategic bets.

Balance is important. A portfolio containing only easy, short-term experiments may optimize local workflow while ignoring structural constraints.

21.24. Learning velocity

A useful organizational outcome is learning velocity: the rate at which consequential uncertainty is reduced and translated into better decisions.

Learning velocity is not the number of experiments run.

It depends on:

- importance of questions;
- quality of evidence;
- time from question to decision;
- replication and transportability;
- implementation of results;
- retention and reuse of knowledge;
- avoidance of repeated failed approaches.

Learning velocity should never become a simplistic target. It is a framing for evaluating the effectiveness of the complete system.

21.25. Signs of maturity

A mature learning engineering organization exhibits recognizable behaviors:

- major metrics have owners, versions, and quality indicators;
- leaders ask for causal evidence rather than dashboard movement alone;
- experiments include exposure, guardrails, and decision rules;
- practitioners can inspect and correct reconstructed evidence;
- negative results are retained;
- AI tools are evaluated on net value and validation burden;
- architectural investments are connected to delivery and risk evidence;
- metrics are retired when their meaning degrades;
- decisions state uncertainty and review dates;
- prior studies are discoverable and reused.

21.26. Signs of failure

The organization is not learning when:

- dashboards multiply but decisions remain unchanged;
- teams cannot explain how metrics are calculated;
- successful pilots are scaled without replication;
- negative studies disappear;
- AI adoption is measured only through usage;
- metrics are used in performance systems beyond their validity;
- data-quality failures remain invisible;
- local practitioners reject the system’s interpretation;

- the same question is repeatedly analyzed from scratch;
- no intervention is ever stopped.

21.27. A practical operating cadence

A learning organization can establish a lightweight but durable cadence.

21.27.1. Weekly system review

Inspect evidence freshness, operational queues, active experiment exposure, guardrails, and incidents.

21.27.2. Monthly learning review

Review completed analyzes, intervention progress, metric quality, and decisions requiring evidence.

21.27.3. Quarterly portfolio review

Assess the balance and value of experiments, architecture studies, AI evaluations, and strategic learning investments.

21.27.4. Semiannual evidence synthesis

Summarize what has been learned, what failed to replicate, which assumptions changed, and which policies should be revisited.

21.27.5. Annual capability review

Evaluate whether the organization has improved its ability to observe, experiment, decide, adapt, and remember.

21.28. The role of an Engineering Intelligence function

A dedicated Engineering Intelligence function should not own all interpretation or decisions.

Its role is to provide:

- evidence infrastructure;
- measurement expertise;
- experimental design;
- economic and statistical analysis;
- governance mechanisms;
- reusable methods;
- cross-domain synthesis;
- coaching for local teams.

Its success is measured by improved organizational decisions and learning capability, not by the number of dashboards or datasets it owns.

21.29. A north-star architecture

The technical platform should support the learning loop end to end:

- event ingestion and immutable evidence;
- temporal identity and ownership;
- Engineering Change graph;
- semantic metric registry;
- complexity and risk models;
- experiment and exposure services;
- decision products;
- evidence and decision registry;
- feedback and correction;
- privacy, access, lineage, and audit.

The architecture should remain modular so that organizations can begin with a narrow evidence path and expand as decisions require.

21.30. A north-star culture

The cultural foundation is equally important.

A learning culture:

- treats evidence as challengeable;
- rewards discovery of mistaken assumptions;
- separates uncertainty from incompetence;
- values negative findings;
- invites practitioner correction;
- expects policies to expire or be reviewed;
- makes tradeoffs explicit;
- avoids individual blame from system telemetry;
- recognizes that every metric is partial.

Without this culture, better instrumentation can merely strengthen existing misconceptions.

21.31. The durable advantage

Tools can be purchased. Models can be copied. Dashboards can be reproduced. Individual practices can diffuse quickly.

A more durable advantage is the ability to learn faster and more honestly than comparable organizations.

That advantage comes from combining:

- high-quality evidence;
- explicit theory;
- disciplined experimentation;
- economic judgment;

- practitioner knowledge;
- architectural understanding;
- trustworthy governance;
- institutional memory.

Engineering Intelligence is the system that connects these capabilities.

21.32. Key takeaways

- The purpose of Engineering Intelligence is organizational learning, not measurement for its own sake.
- Learning requires a closed loop from observation and explanation through intervention, evaluation, decision, and memory.
- Central evidence foundations and local domain ownership must operate together.
- AI, architecture, process, and policy should all be treated as testable interventions.
- Leadership behavior determines whether evidence supports learning or surveillance.
- Negative findings, correction mechanisms, and metric retirement protect the integrity of the system.
- Learning velocity depends on decision consequence, evidence quality, implementation, and retained knowledge—not experiment count.
- The durable organizational advantage is the capacity to reduce important uncertainty and adapt responsibly.

21.33. Open questions

- How should learning velocity be measured without turning it into a corruptible target?
- Which organizational structures best connect centralized evidence expertise with local ownership?
- How can long-run capability loss from automation be detected early?
- What incentives encourage leaders to preserve negative and contradictory findings?
- How should organizations value knowledge that prevents future mistakes but creates no visible immediate outcome?
- Which parts of Engineering Intelligence should become an independent function, and which should remain embedded in engineering teams?

21.34. Learning organization checklist

1. Select consequential uncertainties rather than convenient metrics.
2. Reconstruct the current system with visible coverage and uncertainty.
3. State the mechanism believed to constrain outcomes.
4. Design bounded, reversible, and observable interventions.
5. Evaluate quality, human effort, economics, distribution, and delayed effects.
6. Record explicit adopt, modify, continue, or stop decisions.
7. Preserve methods, results, limitations, and boundary conditions.
8. Enable practitioners to inspect and correct evidence.
9. Review policies and metrics after the system adapts.
10. Maintain critical human capabilities alongside automation.
11. Synthesize learning across teams, systems, and time horizons.

12. Judge the Engineering Intelligence program by improved decisions and durable organizational adaptation.

21.34.1. Research foundations and source notes

The learning-organization framework in this chapter draws on organizational learning, systems thinking, developer experience, socio-technical design, reliable delivery, causal evaluation, and reproducible evidence. It extends those foundations into an operating model where engineering decisions, interventions, outcomes, and corrections become cumulative institutional knowledge.

Single-loop and double-loop learning

Argyris and Schön distinguish single-loop learning, which corrects action within existing assumptions, from double-loop learning, which questions the governing assumptions themselves. See [argyris1978].

This distinction is central to the chapter.

Single-loop learning asks:

- Did the intervention meet its target?
- Was the rollout executed correctly?
- Which operational parameter should change?

Double-loop learning asks:

- Was the original problem framed correctly?
- Was the metric valid for the decision?
- Did the incentive structure create distortion?
- Should the policy or architectural assumption be replaced?

Engineering Intelligence becomes organizational learning only when both loops are supported.

Systems thinking

Senge's learning-organization work emphasizes systems thinking, shared learning, and the ability to understand feedback across interconnected parts of an organization. See [senge1990].

The chapter therefore rejects isolated explanations of engineering performance.

A change in one area may affect:

- queueing elsewhere;
- specialist attention;
- architectural coupling;
- review quality;
- incident exposure;
- practitioner trust;
- economic realization.

Learning requires tracing these feedback relationships rather than rewarding local optimization alone.

Learning from delivery systems

Research on high-performing software delivery provides evidence that technical practices, management systems, culture, and architecture interact. See [forsgren2018].

This supports the chapter's treatment of engineering performance as an organizational capability rather than the sum of individual output.

Relevant learning domains include:

- deployment and recovery;
- work in progress;
- review and validation;
- architecture;
- platform reliability;
- team interaction;
- governance;
- AI-assisted work.

Developer experience as learning evidence

The SPACE framework and DevEx research emphasize multidimensional productivity, feedback loops, cognitive load, and flow. See [forsgren2021-space] and [noda2023-devex].

The chapter therefore treats practitioner experience as evidence about the engineering system.

Repeated reports of slow feedback, ambiguous ownership, noisy automation, or excessive validation burden can reveal structural constraints not visible in throughput metrics.

Practitioner evidence should be combined with operational and outcome data rather than dismissed as subjective noise.

Socio-technical learning

Conway's Law and socio-technical congruence research show that technical and organizational structures are interdependent. See [conway1968] and [cataldo2006].

Distributed-development research further demonstrates the role of coordination boundaries and organizational distance. See [herbsleb2007].

The chapter therefore preserves learning about:

- technical dependencies;
- ownership boundaries;
- communication paths;
- decision rights;
- handoffs;
- organizational change.

A technical intervention may fail because the organizational system surrounding it remains unchanged.

Team cognitive load

Team Topologies emphasizes limiting team cognitive load and designing interaction modes that support flow. See [skelton2019].

The chapter extends this into organizational learning by tracking where teams repeatedly require knowledge outside their stable cognitive boundary.

Recurring coordination, escalation, or specialist dependency may indicate:

- poor service boundaries;
- insufficient platform capability;
- fragmented ownership;
- missing documentation;
- inappropriate team design.

These patterns should trigger structural learning rather than blame.

Learning through experiments

Randomized and quasi-experimental methods provide a disciplined way to learn whether interventions cause outcomes. See [shadish2002], [imbens2015], [hernan2020], and [kohavi2020].

The chapter therefore embeds experiments into normal engineering governance.

A learning organization preserves:

- the original hypothesis;
- assignment and exposure;
- treatment version;
- outcomes and guardrails;
- null and negative findings;
- later replication;
- final decision.

The experiment is not complete when the statistical analysis ends. It is complete when the organization updates policy, records what it learned, and follows realized outcomes.

Learning from incidents

SRE practices treat incidents as opportunities to improve systems and operational capability. See [beyer2016] and [beyer2018].

Systems-oriented incident analysis also cautions against reducing complex failure to one visible human action. See [dekker2014].

The chapter therefore links incidents to:

- Engineering Changes;
- architecture and dependencies;
- decision context;

- review and validation;
- rollout controls;
- observability;
- recovery behavior;
- organizational constraints.

This creates reusable knowledge about failure mechanisms rather than isolated postmortem documents.

Learning from weak signals

High-reliability organizing emphasizes sensitivity to operations, attention to weak signals, and reluctance to simplify. See [weick2007].

The chapter therefore treats the following as learning evidence:

- near misses;
- repeated manual safeguards;
- unusual escalation;
- practitioner concern;
- policy overrides;
- corrections;
- model disagreement;
- repeated workaround behavior.

These signals may reveal fragility before a severe incident occurs.

Reproducible organizational memory

Reproducibility concerns illustrated by [open-science2015] and FAIR principles in [wilkinson2016] support retaining enough evidence to reconstruct important findings.

The chapter therefore connects:

- source evidence;
- metric versions;
- models;
- experiments;
- decisions;
- corrections;
- realized outcomes.

Organizational memory should not depend solely on slide decks, personal recollection, or current dashboards.

Knowledge as versioned evidence

Knowledge changes as evidence, systems, and organizational context change.

The chapter therefore treats knowledge records as versioned objects containing:

- claim;
- supporting evidence;
- causal status;
- applicable context;
- uncertainty;
- owner;
- review date;
- contradictions;
- superseding evidence.

This is an original Engineering Intelligence construct.

A conclusion should be updated or retired when its treatment, population, metric, or context changes materially.

Decision review

A learning organization evaluates both outcomes and reasoning quality.

A decision can be reasonable and still lead to a poor outcome under uncertainty. A good outcome can also result from weak reasoning or luck.

The chapter therefore reviews:

- what was known at decision time;
- assumptions;
- uncertainty;
- alternatives;
- expected mechanisms;
- actual outcomes;
- unexpected consequences;
- which beliefs should change.

This supports better future decisions without hindsight-based blame.

Null and negative knowledge

Null, negative, and contradictory findings are valuable organizational assets.

They can reveal:

- interventions that do not generalize;
- hidden costs;
- invalid metrics;
- treatment-version sensitivity;
- weak causal mechanisms;
- implementation dependence;
- boundary conditions.

The chapter therefore requires retaining unsuccessful experiments and abandoned approaches in the evidence base.

Learning cadence

Learning should occur at multiple cadences:

- operational review for immediate system state;
- experiment review for intervention evidence;
- metric review for validity and gaming;
- architecture review for recurring change cost;
- portfolio review for realized value;
- scientific review for replication and cumulative evidence.

These cadences prevent all learning from being compressed into annual planning or incident response.

Governance of learning

Learning systems can become performative if teams are rewarded for producing positive findings or polished narratives.

Campbell's Law and Goodhart's Law provide the relevant warning. See [campbell1979] and [goodhart1984].

The chapter therefore recommends:

- preserving negative evidence;
- separating study quality from result direction;
- rewarding correction;
- recording uncertainty;
- retiring invalid metrics;
- prohibiting punitive use of exploratory findings.

Economic learning

Cost-benefit analysis provides the basis for comparing expected and realized value. See [boardman2018].

Productivity-realization research in [brynjolfsson1993] and [brynjolfsson2017] supports examining complementary organizational change and delayed benefits.

The chapter therefore tracks:

- expected value;
- released capacity;
- redeployment;
- avoided cost;
- realized operational or customer value;
- forecast error;
- reasons for divergence.

This turns business cases into testable organizational hypotheses.

Original Engineering Intelligence contributions

The following are original synthesis or domain-specific extensions proposed by this book:

- an evidence graph connecting Engineering Changes, metrics, models, experiments, decisions, and outcomes;
- versioned organizational knowledge records with applicability and review dates;
- decision review that separates reasoning quality from outcome luck;
- systematic retention of null, negative, contradictory, and superseded findings;
- learning cadences spanning operations, metrics, experiments, architecture, portfolio, and science;
- correction and dispute treated as positive learning signals;
- treatment-version and context boundaries attached to organizational conclusions;
- business cases treated as hypotheses subject to value-realization review;
- practitioner evidence integrated with telemetry and causal analysis;
- double-loop learning applied to metrics, policies, architecture, and governance.

These constructs should be evaluated through reuse of prior evidence, reduction of repeated failed initiatives, forecast calibration, correction quality, cross-team transfer of valid findings, and the organization's ability to revise deeply held assumptions.

Chapter 22. The Future of Engineering Intelligence



Engineering Intelligence will evolve as software creation becomes increasingly distributed *across people, AI agents, platforms, generated artifacts, and autonomous operational systems. The central challenge will no longer be merely measuring human software-development activity. It will be understanding and governing a dynamic production system in which intent, implementation, validation, and operation may be performed by different combinations of humans and machines. This chapter explores that future, identifies durable principles, and proposes a practical direction for organizations building Engineering Intelligence capabilities today.*

22.1. A changing production system

Software engineering is moving from a predominantly human-authored process toward a mixed production system.

Future Engineering Changes may involve:

- a product manager expressing an outcome in natural language;
- an AI agent decomposing the work;
- specialized agents modifying code, tests, infrastructure, and documentation;
- automated reviewers challenging assumptions;
- policy systems checking security and compliance;

- humans validating architecture, intent, and risk;
- autonomous deployment and recovery systems operating the result.

The boundaries between planning, implementation, review, testing, deployment, and operations will become less distinct.

Engineering Intelligence must therefore measure the complete system of change rather than preserving a tool-centered model inherited from pull requests and tickets.

22.2. Intent becomes a first-class artifact

Traditional instrumentation begins after work has already been translated into tickets, branches, and commits.

As natural-language and multimodal interfaces become more important, the original intent should become a first-class object.

Useful intent evidence may include:

- desired business or user outcome;
- constraints;
- acceptance criteria;
- risk tolerance;
- affected populations;
- architectural boundaries;
- prohibited approaches;
- expected value;
- uncertainty.

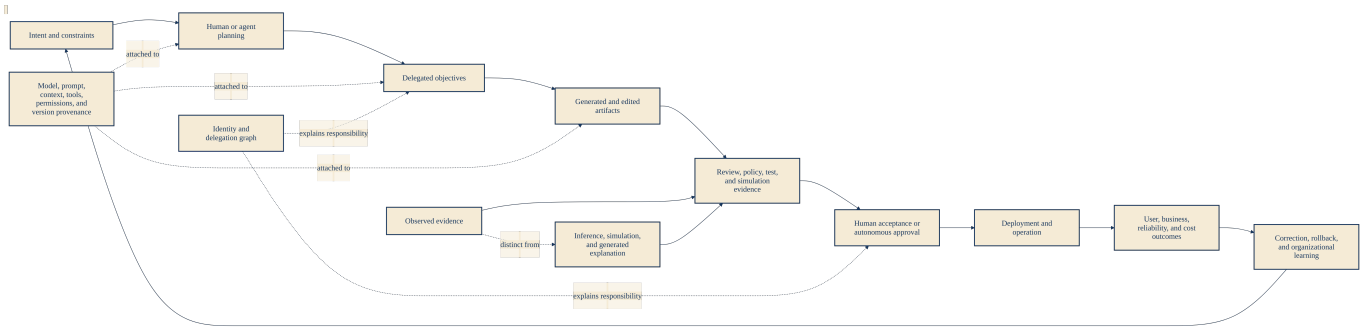
Linking intent to implementation and observed outcome will allow organizations to ask whether the delivered system solved the intended problem, not merely whether the requested artifacts were produced.

22.3. From Engineering Change to change lineage

The Engineering Change remains the primary analytical unit, but its lineage will become richer.

A future lineage graph may connect:

- original intent;
- human and machine interpretations;
- plans and subplans;
- generated and edited artifacts;
- prompts, context, models, tools, and policies;
- review findings and responses;
- test and simulation evidence;
- deployment decisions;
- production behavior;
- business and user outcomes;
- later corrections and reversals.



This lineage should support explanation, audit, learning, and rollback without requiring invasive capture of every private human interaction.

22.4. AI agents as economic actors

AI agents consume resources and create work.

They use:

- inference capacity;
- tool calls;
- repository and environment access;
- human validation;
- test infrastructure;
- deployment capacity;
- organizational risk budget.

Their output must be evaluated as an economic contribution rather than counted as free automation.

The relevant question is not how much code an agent generated. It is whether the agent changed the cost, speed, quality, risk, and capability profile of producing a useful outcome.

22.5. Agent identity and accountability

Autonomous and semi-autonomous agents require durable identities.

An agent identity should make it possible to determine:

- model and version;
- configuration and policy;
- tools and permissions;
- initiating human or system;
- delegated objective;
- evidence available at decision time;
- actions taken;
- review and approval chain;
- resulting outcomes.

Accountability does not mean pretending the agent is legally or morally equivalent to a person. It means preserving enough provenance to understand and govern machine participation.

22.6. Delegation graphs

Simple actor fields will be insufficient when agents delegate to other agents.

A delegation graph should represent:

- who initiated the objective;
- which agent decomposed it;
- what authority was delegated;
- which subagents acted;
- where human approval was required;
- where policy blocked or modified action;
- who accepted the final result.

This graph enables analysis of coordination cost, failure propagation, duplicated work, and unclear responsibility in agentic systems.

22.7. Machine-generated activity inflation

As the marginal cost of generating artifacts falls, activity volume will become even less meaningful.

Organizations may see rapid growth in:

- commits;
- branches;
- pull requests;
- comments;
- tests;
- documentation;
- experiments;
- proposed fixes.

Without quality and outcome context, this abundance can create congestion rather than productivity.

Engineering Intelligence must emphasize accepted value, validation burden, attention consumed, and downstream consequence.

22.8. Scarcity moves to validation

Generation may become plentiful while trustworthy validation remains scarce.

Scarce resources may include:

- domain judgment;
- architecture understanding;

- security expertise;
- production context;
- ethical reasoning;
- accountability for high-impact decisions;
- confidence that requirements reflect real user needs.

The central optimization problem may shift from increasing output to allocating validation attention.

This makes attention economics even more important.

22.9. Review becomes orchestration

Review may evolve from reading a diff to orchestrating multiple evidence sources.

A reviewer may inspect:

- intent and constraints;
- generated plan;
- semantic change summary;
- dependency impact;
- simulations;
- adversarial findings;
- policy checks;
- production analogues;
- uncertainty estimates;
- agent disagreement.

Engineering Intelligence should measure whether this evidence improves decisions or merely overwhelms reviewers.

22.10. Competing and cooperating models

Organizations will increasingly use multiple models and agents.

They may:

- generate alternative designs;
- critique one another;
- vote or rank findings;
- specialize by domain;
- verify tool execution;
- monitor production behavior;
- escalate uncertainty to humans.

Evaluation must therefore move beyond single-model accuracy toward system-level coordination.

Important measures include:

- diversity of useful hypotheses;

- correlated failure;
- disagreement quality;
- arbitration cost;
- validation effort;
- ensemble calibration;
- total decision utility.

22.11. Model monoculture risk

Heavy dependence on one model family or vendor can create correlated failure across the enterprise.

A model monoculture may reproduce the same:

- coding errors;
- security assumptions;
- architectural preferences;
- omissions;
- misleading explanations;
- compliance failures.

Engineering Intelligence should track model concentration, failure correlation, and critical workflows lacking independent validation.

22.12. Continuous evaluation

Periodic vendor evaluations will be insufficient for rapidly changing models.

Future evaluation should be continuous and contextual.

It should detect:

- model-version changes;
- prompt and policy changes;
- toolchain changes;
- domain drift;
- changing human reliance;
- calibration degradation;
- subgroup regressions;
- changes in cost or latency;
- new failure modes.

Evaluation should operate in shadow mode where possible before a new model or configuration affects critical decisions.

22.13. Dynamic treatment systems

AI-assisted workflows will not be static treatments.

The treatment may vary by:

- model;
- prompt;
- context window;
- retrieval source;
- agent topology;
- tool permissions;
- repository state;
- user behavior;
- policy configuration.

Experiment systems must record the actual treatment received, not only whether a nominal feature flag was enabled.

22.14. Personalized engineering assistance

Assistance may adapt to an engineer's role, experience, task, and context.

Personalization can improve usefulness, but it creates risks:

- reinforcing narrow habits;
- unequal access to better assistance;
- inappropriate inference about ability;
- hidden treatment differences;
- reduced comparability;
- privacy intrusion.

Evaluation should focus on task and system outcomes while limiting unsupported personal profiling.

22.15. Skill development and skill atrophy

Long-term effects will become central.

AI assistance may accelerate learning by providing explanations and examples. It may also weaken capability by removing opportunities to practice diagnosis, design, and careful review.

Engineering Intelligence should distinguish:

- immediate task performance;
- learning rate;
- retained understanding;
- independent performance;
- ability to detect machine error;
- resilience during tool absence.

These effects require longitudinal study rather than short pilots.

22.16. Software architecture generated at scale

AI systems may produce local implementations faster than organizations can maintain global coherence.

Potential consequences include:

- duplicated components;
- inconsistent abstractions;
- dependency growth;
- hidden coupling;
- fragmented ownership;
- excessive configuration;
- generated-code maintenance burden.

Architecture metrics must therefore become more dynamic and connected to generation provenance.

22.17. Architecture policy as executable evidence

Architecture principles may increasingly be expressed as machine-readable constraints.

Examples include:

- permitted dependency directions;
- data-boundary rules;
- resilience requirements;
- approved interface patterns;
- ownership constraints;
- regional deployment restrictions.

Engineering Intelligence can evaluate whether these rules prevent harmful outcomes, create unnecessary friction, or require revision.

Policy should be treated as an intervention whose effects can be measured.

22.18. Simulation before deployment

More Engineering Changes will be evaluated through simulation.

Simulation may cover:

- production traffic;
- failure scenarios;
- dependency outages;
- security attacks;
- performance limits;
- cost behavior;
- user interactions;
- operational recovery.

The challenge will be measuring simulation fidelity and knowing when simulated evidence transfers to production.

22.19. Digital twins of engineering systems

Organizations may create operational models of their engineering system.

A digital twin could represent:

- teams and expertise;
- repositories and dependencies;
- queues and capacity;
- environments;
- deployment pathways;
- policy constraints;
- incident propagation;
- costs and outcomes.

Such a model could estimate the likely effect of changing review rules, CI capacity, team ownership, architecture, or AI adoption before implementation.

The twin must remain grounded in observed evidence and expose uncertainty. Otherwise it becomes an elaborate narrative generator.

22.20. Prescriptive systems

Engineering Intelligence may progress from describing and predicting to recommending action.

A prescriptive system might recommend:

- reviewer routing;
- test depth;
- release timing;
- model selection;
- human approval;
- architectural decomposition;
- rollback;
- capacity allocation.

These recommendations should be evaluated as policies, not as isolated predictions.

A policy must account for cost, constraints, feedback effects, and distributional consequences.

22.21. Closed-loop automation

Some recommendations will be executed automatically.

Closed-loop systems may:

- retry or reconfigure CI;

- adjust capacity;
- route reviews;
- pause deployments;
- roll back releases;
- open corrective changes;
- update tests;
- escalate anomalies.

Automation levels should correspond to evidence strength, reversibility, and consequence.

High-impact irreversible actions require stronger controls than low-risk reversible ones.

22.22. Human control is not a binary

The future will not divide neatly into human-controlled and autonomous systems.

Control may take several forms:

- prior policy definition;
- permission boundaries;
- real-time approval;
- exception review;
- retrospective audit;
- kill switches;
- bounded autonomy;
- outcome-based supervision.

Engineering Intelligence should record which form of control existed and whether it was effective.

22.23. Governance at machine speed

Traditional governance processes are often slower than model and workflow changes.

Future governance must become more executable.

It should include:

- machine-readable policy;
- automated evidence collection;
- continuous control testing;
- configuration lineage;
- model and prompt registries;
- dynamic risk classification;
- rapid rollback;
- human escalation for ambiguity.

Governance should constrain dangerous action without requiring every low-risk change to pass through a slow central committee.

22.24. Privacy under richer instrumentation

Richer lineage creates greater privacy risk.

Organizations must avoid the assumption that because data can be collected, it should be collected.

Future privacy design should emphasize:

- purpose limitation;
- data minimization;
- role-based and purpose-based access;
- aggregation where possible;
- short retention for sensitive detail;
- separation of learning from employment decisions;
- transparent collection notices;
- correction and appeal;
- privacy-preserving analysis.

The system should preserve accountability for automated action without constructing unnecessary dossiers on individuals.

22.25. Security of the evidence system

Engineering Intelligence platforms will become high-value targets.

They may expose:

- source code metadata;
- architecture maps;
- vulnerabilities;
- incident history;
- employee relationships;
- model configurations;
- deployment controls;
- business priorities.

Security must include strong isolation, least privilege, tamper evidence, audit, retention controls, and careful treatment of prompts and retrieved context.

Evidence integrity is especially important when the platform influences automated policy.

22.26. Adversarial behavior

Attackers may attempt to manipulate Engineering Intelligence systems by:

- poisoning telemetry;
- generating misleading activity;
- hiding agent provenance;
- exploiting model inputs;

- manipulating risk scores;
- inducing unsafe automation;
- corrupting benchmark datasets;
- targeting evidence lineage.

Metrics and models should be threat-modeled when they influence access, deployment, or prioritization.

22.27. Synthetic evidence risk

As generated text, code, tests, and reports become common, organizations must distinguish original observation from synthetic interpretation.

An AI-generated incident summary is not the incident record. A generated test is not evidence that the requirement is correct. A model's confidence is not empirical uncertainty.

Lineage should preserve the boundary between:

- observed events;
- human assertions;
- model inferences;
- generated explanations;
- simulated outcomes;
- verified outcomes.

22.28. Open standards

The field will benefit from interoperable standards for:

- Engineering Change identity;
- event envelopes;
- agent identity and delegation;
- model and prompt provenance;
- experiment exposure;
- metric definitions;
- evidence quality;
- decision records;
- outcome linkage.

Open standards reduce vendor lock-in and support cross-tool learning.

They should define semantics without requiring organizations to disclose proprietary content.

22.29. Vendor-neutral evidence

Organizations should preserve evidence independently of the tools that generated it.

A vendor-neutral evidence layer allows:

- longitudinal comparison across tool changes;
- independent evaluation;
- migration;
- audit;
- reproducibility;
- portfolio-level analysis.

Without it, each vendor controls the interpretation of its own effectiveness.

22.30. Engineering Intelligence as infrastructure

Over time, Engineering Intelligence may become as fundamental as source control, CI, and observability.

It can provide shared services for:

- change identity;
- evidence lineage;
- risk classification;
- experimentation;
- policy evaluation;
- human and agent coordination;
- economic analysis;
- organizational learning.

This does not mean one monolithic platform. It means a durable layer of contracts and capabilities across the engineering ecosystem.

22.31. What should remain stable

Tools and workflows will change, but several principles should remain stable.

22.31.1. Measure outcomes, not artifact volume

Abundant machine-generated activity makes this more important, not less.

22.31.2. Treat attention as scarce

Human validation, judgment, and accountability remain constrained resources.

22.31.3. Preserve the Engineering Change as a unit of consequence

Tools are transient; intended changes and their outcomes are durable analytical objects.

22.31.4. Separate observation from inference

Recorded events, reconstructed relationships, model predictions, and causal claims must remain distinguishable.

22.31.5. Require lineage

Important decisions should be traceable to evidence, configuration, actors, and outcomes.

22.31.6. Evaluate interventions causally

Adoption and correlation do not establish benefit.

22.31.7. Include economics

Automation must be evaluated on net value, not nominal output.

22.31.8. Govern use, not only collection

The same data can support learning or enable harmful surveillance.

22.31.9. Expect adaptation

Metrics, models, and policies change the system they observe.

22.31.10. Preserve human challenge

People must remain able to question evidence and automated recommendations.

22.32. Near-term priorities

Organizations do not need to predict the distant future before acting.

Useful near-term priorities are:

1. define Engineering Change identity and linkage;
2. preserve immutable source evidence and lineage;
3. separate human, automation, and agent activity;
4. version models, prompts, policies, and metrics;
5. measure validation burden and attention;
6. establish experiment and exposure infrastructure;
7. create model-independent outcome measures;
8. maintain a decision and evidence registry;
9. implement strong prohibited-use policy;
10. begin with consequential bounded decisions.

These investments remain valuable across changes in tools and models.

22.33. A staged future architecture

A future-capable architecture can evolve in stages.

22.33.1. Stage 1: observable workflows

Connect source control, CI, deployment, incidents, identity, and cost.

22.33.2. Stage 2: reconstructed changes

Create reliable Engineering Change lineage and governed metrics.

22.33.3. Stage 3: experimental organization

Support treatment assignment, exposure, guardrails, and decision records.

22.33.4. Stage 4: human-machine production system

Add agent identity, delegation, model provenance, and validation economics.

22.33.5. Stage 5: prescriptive intelligence

Recommend interventions with explicit objectives and uncertainty.

22.33.6. Stage 6: bounded autonomous optimization

Allow reversible closed-loop actions under continuously evaluated policy.

Organizations should not skip foundational stages merely because advanced models are available.

22.34. Failure futures

Several undesirable futures are plausible.

22.34.1. The surveillance enterprise

Telemetry is used to rank individuals through weak proxies, reducing trust and corrupting evidence.

22.34.2. The automation illusion

Generated activity rises dramatically while validation queues, defects, and complexity grow.

22.34.3. The model monoculture

One vendor’s assumptions become embedded across design, implementation, review, and operations.

22.34.4. The unaccountable agent mesh

Autonomous systems delegate work without durable identity, provenance, or clear acceptance responsibility.

22.34.5. The synthetic management system

Executives receive polished machine-generated narratives disconnected from verified operational evidence.

22.34.6. The permanently experimental organization

Endless pilots occur, but no difficult adoption, reversal, or capability decision is made.

Engineering Intelligence should be designed explicitly to avoid these outcomes.

22.35. A desirable future

A desirable future is not one in which humans disappear from engineering.

It is one in which:

- routine implementation becomes cheaper;
- intent is clearer;
- validation is better targeted;
- architecture decisions are evidence-informed;
- AI failures are detectable and reversible;
- engineers spend more time on consequential judgment;
- experimentation replaces unsupported opinion;
- policy adapts to observed effects;
- organizational learning compounds over time.

In this future, Engineering Intelligence is not a control tower watching engineers. It is a shared nervous system helping the organization understand and improve how change is produced.

22.36. Final synthesis

The framework developed in this book rests on a simple proposition:

> Software engineering can be understood and improved when organizations connect intent, change, attention, architecture, evidence, economics, and outcomes without reducing human contribution to visible activity.

Engineering economics establishes why measurement matters.

The Engineering Change provides the unit of analysis.

Attention economics reveals the constrained human resource.

Measurement theory distinguishes constructs from convenient proxies.

Causal inference establishes whether interventions work.

AI evaluation treats machine assistance as part of the production system.

Enterprise instrumentation creates trustworthy evidence.

Complexity and blast radius provide architectural context.

The Engineering Intelligence platform makes the evidence operational.

The metric catalogue and experiment playbooks turn theory into practice.

Governance protects legitimacy.

Portfolio analysis connects engineering evidence to executive decisions.

Implementation guidance creates a path to adoption.

Scientific practice makes learning cumulative.

The learning organization closes the loop.

The future extends that loop across increasingly autonomous human-machine systems.

22.37. Key takeaways

- Software engineering is becoming a mixed human-machine production system.
- Intent, agent identity, delegation, model provenance, and validation burden will become essential evidence.
- Artifact volume will become less meaningful as generation becomes abundant.
- Human attention and trustworthy validation will remain scarce.
- AI systems must be evaluated economically, causally, longitudinally, and at the system level.
- Engineering Intelligence should support bounded autonomy while preserving provenance, reversibility, and human challenge.
- Vendor-neutral evidence and open semantic standards will be strategically important.
- The most durable investments are change identity, lineage, governed metrics, experimentation, outcome linkage, and trustworthy use policy.
- The desirable future is a learning system, not a surveillance system.

22.38. Open questions

- How should agent delegation and acceptance responsibility be represented across vendors and tools?
- Which forms of human validation remain essential as model capability increases?
- How can organizations measure capability atrophy before a critical failure reveals it?
- What evidence threshold should permit autonomous engineering action?
- How should correlated model failure and model monoculture be quantified?
- Can digital twins of engineering organizations produce reliable policy guidance?
- Which Engineering Intelligence standards should be open and interoperable?
- How can evidence-rich systems preserve privacy and professional autonomy?

22.39. Future-readiness checklist

1. Make intent and Engineering Change identity durable across tools.
2. Preserve observed evidence separately from generated interpretation.
3. Record human, automation, model, agent, and delegation provenance.
4. Measure validation burden, attention cost, and accepted outcome value.

5. Maintain vendor-neutral metrics and outcome evidence.
6. Continuously evaluate model, prompt, policy, and workflow changes.
7. Track model concentration and correlated failure risk.
8. Design autonomy around consequence, reversibility, and evidence maturity.
9. Keep human correction, appeal, escalation, and shutdown mechanisms.
10. Protect privacy through purpose limitation and data minimization.
11. Treat architecture policy and AI adoption as testable interventions.
12. Build institutional memory so learning compounds across future tool generations.

22.39.1. Research foundations and source notes

The future-facing framework in this chapter draws on AI risk management, autonomous-system governance, resilience engineering, organizational learning, software-delivery research, and cumulative scientific evidence. Because many claims concern emerging capabilities and organizational forms, this section distinguishes established foundations from forward-looking Engineering Intelligence synthesis.

Established foundations versus forecasts

The chapter's predictions should not be read as settled facts.

Established evidence supports several broad directions:

- software delivery is a socio-technical capability rather than an individual-output problem;
- architecture, workflow, culture, and management practices interact;
- AI assistance can affect bounded task performance and human workflow;
- automation creates verification, trust, governance, and failure-mode concerns;
- organizational benefits often require complementary process and capability change;
- reliable operation requires observability, bounded change, and recovery capability.

Relevant foundations include [forsgren2018], [forsgren2021-space], [noda2023-devex], [peng2023], [vaithilingam2022], [barke2023], [beyer2016], and [beyer2018].

Claims about future agent autonomy, organizational structure, and labor composition remain hypotheses that require ongoing empirical validation.

AI capability and repository-level agents

Repository-level benchmarks such as SWE-bench provide evidence that software agents can perform increasingly complex issue-resolution tasks in existing codebases. See [jimenez2024].

These benchmarks do not establish unrestricted production autonomy.

Future capability assessment must include:

- operational permissions;
- security and privacy boundaries;
- side effects;
- human oversight;

- cost;
- reliability across heterogeneous repositories;
- long-term maintainability;
- rollback and recovery.

The chapter therefore treats benchmark progress as capability evidence, not as a complete deployment decision.

Human–AI collaboration

Empirical studies of coding assistants show that AI changes not only production speed but also comprehension, debugging, expectation, and interaction behavior. See [peng2023], [vaithilingam2022], and [barke2023].

This supports a future in which engineering work is organized around mixed human and machine participation rather than simple substitution.

Relevant future research questions include:

- Which tasks should remain human-led?
- Which tasks can be delegated safely?
- How does verification burden evolve as generation quality improves?
- Which skills strengthen or atrophy?
- How should apprenticeship and mentoring change?
- How should accountability be assigned across human and machine actors?

AI risk management

The NIST AI Risk Management Framework and its generative-AI profile provide foundations for governing, mapping, measuring, and managing AI risk. See [nist2023-airmf] and [nist2024-genai].

Management-system and risk-process standards are represented by 001 and 894.

These foundations support future requirements for:

- model and prompt registries;
- treatment-version control;
- bounded authority;
- continuous evaluation;
- human accountability;
- incident response;
- retirement and rollback;
- vendor and concentration risk.

The standards guide governance but do not determine the correct autonomy level for a specific organization.

Delegation and autonomy

Future engineering systems may involve agents that inspect, modify, test, merge, deploy, and coordinate with other agents.

The chapter therefore models autonomy as a spectrum rather than a binary state.

Authority should be separated into permissions to:

- observe;
- recommend;
- modify;
- execute tests;
- approve;
- merge;
- deploy;
- alter data or infrastructure;
- delegate further authority.

The autonomy-tier and delegation-graph concepts are original Engineering Intelligence extensions compatible with provenance in [moreau2013] and AI risk-governance principles above.

Provenance in machine-rich engineering

W3C PROV provides a foundation for representing entities, activities, agents, generation, use, and derivation. See [moreau2013].

Future Engineering Intelligence systems should preserve:

- human and machine actors;
- delegation chains;
- model, prompt, retrieval, and tool versions;
- evidence used;
- actions performed;
- approvals and overrides;
- downstream outcomes.

Without this provenance, organizations may be unable to explain why an autonomous action occurred or which configuration produced it.

Resilience and graceful extensibility

Future engineering organizations will face model failures, vendor outages, correlated errors, adversarial inputs, and novel operating conditions.

Resilience foundations are provided by [woods2015], while high-reliability organizing is discussed in [weick2007].

The chapter therefore emphasizes:

- bounded failure domains;
- alternate operating paths;
- human fallback;
- degraded modes;

- rapid isolation;
- preserved expert capability;
- tested rollback;
- attention to weak signals.

A system that performs well only under expected conditions is not necessarily resilient.

Model monoculture and correlated failure

Concentrating engineering work on one provider, model family, or toolchain can create correlated risk.

Potential effects include:

- repeated insecure patterns;
- common reasoning blind spots;
- synchronized service interruption;
- shared policy or licensing changes;
- reduced diversity of solutions;
- strategic switching difficulty.

The model-monoculture concept is a forward-looking Engineering Intelligence synthesis. It should be evaluated through provider-concentration analysis, red-team studies, fallback exercises, and cross-model disagreement testing.

Future architecture and organizational structure

Conway's Law suggests continuing interaction between communication structure and system design. See [conway1968].

Socio-technical congruence and distributed-development research further support studying technical and organizational dependencies together. See [cataldo2006] and [herbsleb2007].

As agents participate more deeply, organizations may need to redesign:

- ownership boundaries;
- review responsibility;
- escalation paths;
- platform interfaces;
- decision rights;
- evidence and audit systems.

The exact future structure remains an empirical question rather than a predetermined outcome.

Cognitive load and skill

Team cognitive load is an important design consideration in [skelton2019].

AI may reduce some local cognitive burden while increasing verification, coordination, or oversight burden elsewhere.

The chapter therefore treats future skill and attention effects as multidimensional:

- cognitive offload;
- cognitive reload;
- learning acceleration;
- skill atrophy;
- dependency on machine availability;
- changes in mentoring;
- redistribution of expert attention.

Longitudinal evidence is required before concluding that AI universally augments or degrades engineering skill.

Productivity and complementary change

The productivity effects of new technology may be delayed while organizations redesign processes, roles, and complementary capabilities. See [brynjolfsson1993] and [brynjolfsson2017].

This supports the chapter's claim that future engineering value will depend on more than model capability.

Organizations may need:

- new validation practices;
- new architecture boundaries;
- new governance roles;
- new learning systems;
- new economic and attention measures;
- new incident and recovery capabilities.

Capability adoption without complementary redesign may create cost without proportional value.

Scientific evaluation of future systems

Future claims should be tested using the causal and reproducibility foundations in [shadish2002], [imbens2015], [hernan2020], [kohavi2020], [nosek2018], [open-science2015], and [wilkinson2016].

This requires:

- preregistered hypotheses where appropriate;
- versioned treatments;
- representative populations;
- quality and safety guardrails;
- replication across task classes;
- preservation of negative findings;
- long observation windows for downstream consequence.

Fast-moving technology increases rather than reduces the need for rigorous evaluation.

Governance and professional responsibility

Future systems remain subject to privacy, cybersecurity, and professional obligations.

Relevant foundations include [nist2020-privacy], [nist2024-csf], and [acm2018-ethics].

Organizations should not treat technical feasibility as sufficient justification for:

- covert surveillance;
- unrestricted autonomous action;
- opaque employment decisions;
- indefinite retention of behavioral telemetry;
- delegation without accountability;
- removal of meaningful human correction or appeal.

Organizational learning

Argyris and Schön's double-loop learning and Senge's systems thinking provide the foundation for continuously revising assumptions about future engineering work. See [argyris1978] and [senge1990].

The chapter therefore treats future-readiness as the ability to learn safely rather than the ability to predict one fixed destination.

A future-ready organization should be able to:

- test new capabilities;
- detect harm;
- reverse decisions;
- retain contradictory evidence;
- update governance;
- preserve institutional knowledge;
- redeploy human attention deliberately.

Scenario planning

The chapter's future scenarios are not forecasts with assigned certainty.

They are structured possibilities intended to expose decisions under different assumptions about:

- AI capability growth;
- cost;
- autonomy;
- regulation;
- workforce adaptation;
- concentration risk;
- architecture and platform maturity.

Scenario analysis should be paired with signposts that indicate which future is becoming more plausible and what reversible actions preserve strategic options.

Original Engineering Intelligence contributions

The following are original synthesis or forward-looking extensions proposed by this book:

- machine actors as first-class participants in Engineering Changes;
- autonomy tiers and delegation graphs for engineering agents;
- machine-contribution provenance spanning model, prompt, retrieval, tools, policy, and action;
- model-monoculture and correlated-risk analysis;
- organizational attention as a strategic resource allocated across humans and machines;
- long-horizon skill and capability guardrails;
- evidence-maturity gates for expanding autonomy;
- future scenarios linked to observable signposts and reversible decisions;
- continuous scientific evaluation embedded in engineering operations;
- future-readiness defined as safe learning and adaptation rather than prediction accuracy.

These claims should be treated as a research agenda. Their validity depends on future empirical work, replication, operational evidence, governance outcomes, and the observed consequences of increasingly autonomous engineering systems.

Appendix A: Engineering Change Reference Model

***This appendix defines a practical reference model for representing Engineering Changes across** heterogeneous enterprise systems. It specifies the core entities, relationships, lifecycle states, identifiers, confidence measures, provenance requirements, and implementation rules needed to reconstruct change activity without tying the model to a particular source-control, planning, CI, deployment, or incident platform.*

A.1. Purpose

The Engineering Change reference model provides a stable semantic layer across changing tools and workflows.

It is intended to support:

- lifecycle reconstruction;
- cycle-time and queue analysis;
- review and validation measurement;
- experiment assignment and exposure;
- complexity and blast-radius analysis;
- AI-participant evaluation;
- cost and value attribution;
- audit and governance;
- longitudinal comparison across tool migrations.

The model should preserve uncertainty rather than implying perfect linkage where none exists.

A.2. Design principles

A.2.1. Tool independence

The model describes engineering meaning, not vendor objects. A pull request, merge request, change list, patch set, or direct commit may all contribute to the same semantic role.

A.2.2. Event-derived state

State should be reconstructible from immutable events wherever possible. Mutable snapshots may accelerate queries but should not replace source evidence.

A.2.3. Temporal correctness

Ownership, identity, team membership, service boundaries, and model configuration must be resolved as of the time of the event.

A.2.4. Explicit uncertainty

Inferred relationships require confidence, method, and provenance.

A.2.5. Many-to-many reality

One intent may produce several code changes. One deployment may contain several Engineering Changes. One incident may involve multiple changes. The model must not force false one-to-one relationships.

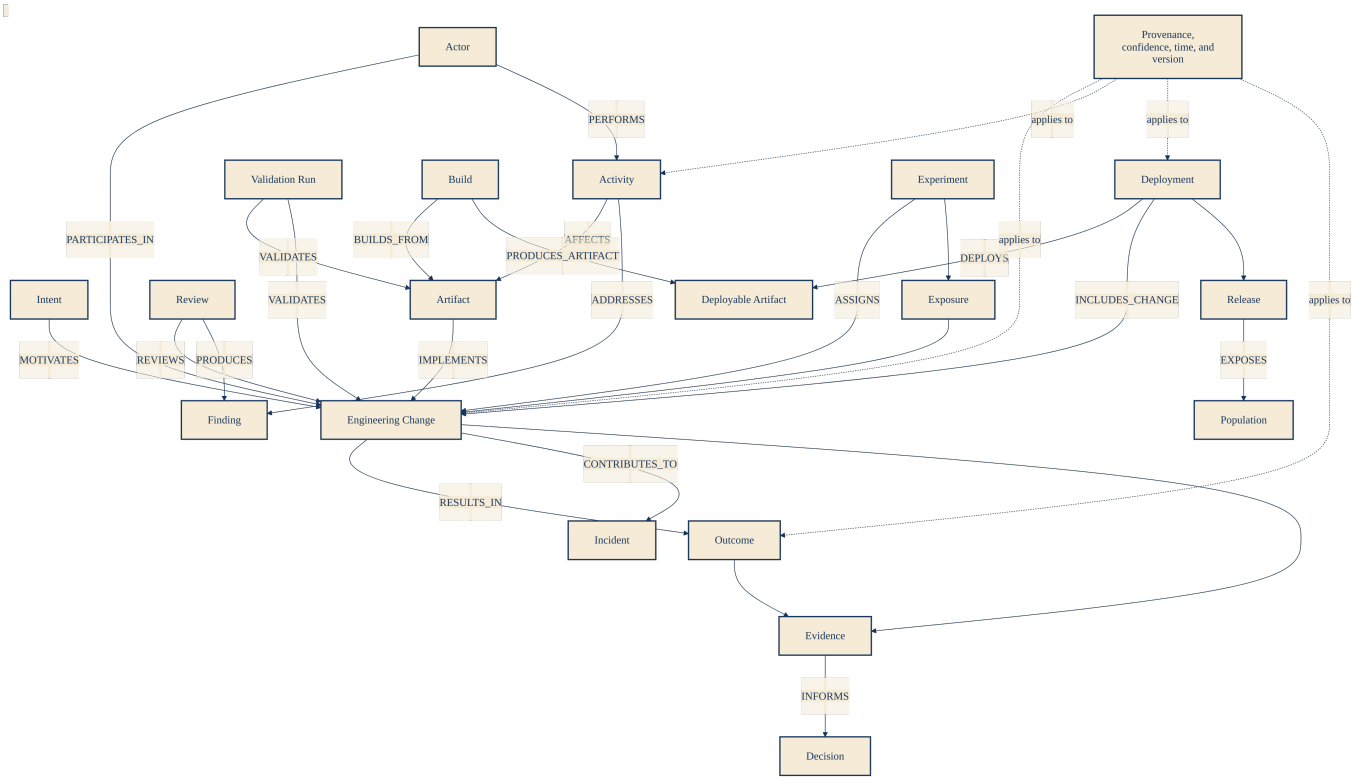
A.2.6. Human and machine participation

People, bots, CI services, AI assistants, and autonomous agents are all actors with distinguishable identity and provenance.

A.2.7. Outcome orientation

Artifacts are intermediate evidence. The model should connect them to accepted, deployed, operational, user, and business outcomes where possible.

A.3. Core entity model



A.3.1. EngineeringChange

The central unit representing an intended modification and its observed consequences.

Recommended fields:

```
engineering_change_id: ec_01J...
title: Reduce checkout authorization latency
change_type: product_feature | defect_fix | reliability | security | platform | compliance |
experiment | maintenance
created_at: 2026-07-15T18:42:11Z
first_activity_at: 2026-07-15T19:10:02Z
accepted_at: 2026-07-18T16:31:44Z
completed_at: 2026-07-21T09:03:15Z
state: operating
risk_class: high
primary_domain_id: domain_payments
primary_owner_team_id: team_checkout
linkage_confidence: 0.94
linkage_method: explicit_id_plus_graph_reconstruction
schema_version: 1.0.0
```

The identifier should be globally unique, durable, opaque, and independent of any single source system.

A.3.2. Intent

Represents the desired outcome, problem, constraint, or hypothesis that motivated the change.

Fields may include:

- intent identifier;
- statement of desired outcome;
- acceptance criteria;
- constraints;
- expected value;
- affected population;
- risk tolerance;
- originating actor;
- source-system references;
- creation and revision timestamps.

An Engineering Change may have multiple intents when work is consolidated, but a primary intent should be identified where meaningful.

A.3.3. Actor

Represents a participant capable of taking or initiating action.

Actor types include:

- human;
- service account;
- deterministic automation;
- AI assistant;
- autonomous agent;
- external vendor system;

- organizational unit.

Recommended actor properties:

- canonical actor identifier;
- actor type;
- source identities;
- organization and team membership over time;
- role at event time;
- model or automation version where applicable;
- authentication and delegation context;
- confidence of identity resolution.

A.3.4. Artifact

Represents a created, modified, reviewed, executed, or deployed object.

Artifact types include:

- source file;
- commit;
- branch;
- pull request;
- test;
- build;
- package;
- container image;
- infrastructure plan;
- schema migration;
- model;
- configuration;
- documentation;
- deployment manifest;
- feature flag;
- incident record.

Artifacts should retain native source identifiers and immutable content identifiers where available.

A.3.5. Activity

Represents an action performed by an actor or system.

Examples include:

- authoring;
- generating;
- editing;

- reviewing;
- commenting;
- approving;
- testing;
- building;
- deploying;
- rolling back;
- diagnosing;
- acknowledging;
- accepting;
- rejecting.

An activity should contain event time, ingestion time, actor, source, affected artifacts, related Engineering Change, and provenance.

A.3.6. Review

Represents a structured validation interaction rather than merely a comment stream.

A review may contain:

- requested reviewers;
- participating reviewers;
- reviewer type;
- findings;
- severity;
- ranking position;
- disposition;
- response latency;
- approval or rejection;
- human validation effort;
- AI model and configuration.

A.3.7. Finding

Represents a specific concern, recommendation, defect, policy violation, or observation produced during review or analysis.

Recommended fields:

- finding identifier;
- producer actor;
- category;
- severity;
- confidence;
- rank;

- affected artifact and location;
- first observed time;
- disposition;
- accepted, rejected, duplicate, or unresolved state;
- remediation activity;
- later outcome linkage.

A.3.8. ValidationRun

Represents execution intended to establish evidence about correctness, quality, safety, or fitness.

Examples include:

- unit tests;
- integration tests;
- security scans;
- static analysis;
- performance tests;
- simulations;
- canary analysis;
- manual verification;
- compliance checks.

A validation run should record configuration, inputs, environment, results, duration, cost, and reliability of the evidence.

A.3.9. Build

Represents creation of a deployable or distributable artifact.

Important fields include:

- build identifier;
- source inputs;
- dependency resolution;
- environment;
- output artifacts;
- result;
- duration;
- queue time;
- resource consumption;
- cache behavior;
- reproducibility metadata.

A.3.10. Deployment

Represents movement of one or more artifacts into an environment.

A deployment may contain several Engineering Changes, and one Engineering Change may reach several environments through separate deployments.

Fields include:

- deployment identifier;
- environment;
- artifact set;
- included Engineering Changes;
- start and completion times;
- strategy;
- actor;
- approval evidence;
- result;
- rollback relationship;
- exposure population.

A.3.11. Release

Represents intentional availability to a user or consuming system.

Deployment and release must remain distinct because feature flags, staged exposure, mobile distribution, or customer entitlements may separate them in time.

A.3.12. Outcome

Represents an observed consequence.

Outcome types include:

- accepted functionality;
- user behavior;
- business result;
- reliability change;
- incident;
- defect;
- rollback;
- support burden;
- cost change;
- performance change;
- learning result;
- capability change.

Outcomes should record observation method, attribution confidence, time horizon, and uncertainty.

A.3.13. Incident

Represents an operational failure, degradation, security event, or significant near miss.

Incident linkage to Engineering Changes should be many-to-many and time-aware. The model should distinguish suspected, contributing, triggering, and corrective relationships.

A.3.14. Experiment

Represents a predeclared comparison intended to estimate an intervention effect.

Fields include:

- experiment identifier;
- hypothesis;
- eligibility;
- assignment unit;
- treatment variants;
- outcome and guardrail definitions;
- analysis plan;
- start and end conditions;
- owner;
- decision record.

A.3.15. Exposure

Represents the treatment actually experienced by an actor, Engineering Change, repository, team, or other unit.

Exposure must include treatment configuration, timestamp, duration, and evidence that the treatment was operationally available.

A.3.16. Decision

Represents an explicit organizational choice informed by evidence.

A decision record should contain:

- question;
- options considered;
- evidence references;
- assumptions;
- uncertainty;
- expected benefit and cost;
- chosen action;
- owner;
- decision date;
- review date;
- reversal criteria.

A.4. Relationship vocabulary

Relationships should be typed, directed, temporal where needed, and versioned.

Recommended relationship types include:

- `MOTIVATES`: Intent to EngineeringChange;
- `IMPLEMENTS`: Artifact to EngineeringChange;
- `PARTICIPATES_IN`: Actor to EngineeringChange;
- `PERFORMS`: Actor to Activity;
- `AFFECTS`: Activity to Artifact;
- `REVIEWS`: Review to Artifact or EngineeringChange;
- `PRODUCES`: Review to Finding;
- `ADDRESSES`: Activity or Artifact to Finding;
- `VALIDATES`: ValidationRun to Artifact or EngineeringChange;
- `BUILDS_FROM`: Build to Artifact;
- `PRODUCES_ARTIFACT`: Build to Artifact;
- `DEPLOYS`: Deployment to Artifact;
- `INCLUDES_CHANGE`: Deployment to EngineeringChange;
- `EXPOSES`: Release or Exposure to population;
- `RESULTS_IN`: EngineeringChange to Outcome;
- `CONTRIBUTES_TO`: EngineeringChange to Incident;
- `CORRECTS`: EngineeringChange to Incident or prior EngineeringChange;
- `ASSIGNED_TO`: Experiment to unit;
- `RECEIVES`: unit to Exposure;
- `INFORMS`: evidence to Decision;
- `SUPERSEDES`: entity version to earlier version;
- `DELEGATES_TO`: Actor to Actor;
- `OWNS`: Team to domain, service, repository, or EngineeringChange.

Free-form relationships should be avoided in canonical storage because they undermine interoperability.

A.5. Lifecycle model

The lifecycle is not a universal workflow. It is a semantic state model that can be mapped from different operational processes.

A.5.1. Proposed

Intent exists, but implementation has not begun.

A.5.2. Active

Implementation or substantive investigation is underway.

A.5.3. In validation

The change is being reviewed, tested, simulated, or otherwise evaluated.

A.5.4. Accepted

The organization has accepted the change for integration or delivery.

A.5.5. Integrated

The change has entered the primary shared artifact or release branch.

A.5.6. Deployed

Artifacts containing the change have reached at least one target environment.

A.5.7. Released

The intended population can experience the change.

A.5.8. Operating

The change is active and outcome observation is underway.

A.5.9. Completed

The declared completion condition has been met and no further active lifecycle work is expected.

A.5.10. Reverted

The effect was intentionally removed.

A.5.11. Abandoned

The work stopped before acceptance or release.

A.5.12. Superseded

A later Engineering Change replaced the original intent or implementation.

An Engineering Change may revisit earlier states. State transitions should be derived from events, and both current state and transition history should be retained.

A.6. State transition rules

Example transition rules:

```
PROPOSED -> ACTIVE
ACTIVE -> IN_VALIDATION
```

```
IN_VALIDATION -> ACTIVE
IN_VALIDATION -> ACCEPTED
ACCEPTED -> INTEGRATED
INTEGRATED -> DEPLOYED
DEPLOYED -> RELEASED
RELEASED -> OPERATING
OPERATING -> COMPLETED
ANY_NONTERMINAL -> ABANDONED
DEPLOYED|RELEASED|OPERATING|COMPLETED -> REVERTED
ANY -> SUPERSEDED
```

Organizations should configure local mappings rather than modify the canonical semantics casually.

A.7. Identity strategy

A.7.1. Explicit identifiers

The preferred approach is to propagate an Engineering Change identifier through:

- planning records;
- branch or change metadata;
- commits;
- pull requests;
- CI runs;
- build provenance;
- deployment manifests;
- feature flags;
- incident and decision records.

Explicit identifiers reduce reconstruction ambiguity but should not be mandatory for every legacy or emergency workflow.

A.7.2. Deterministic reconstruction

Use exact relationships such as:

- commit contained in pull request;
- build references commit SHA;
- deployment references image digest;
- release references deployment;
- incident references deployment identifier.

A.7.3. Probabilistic reconstruction

When exact links are absent, use evidence such as:

- temporal proximity;
- shared actor;
- textual references;

- branch naming;
- repository and service ownership;
- artifact similarity;
- deployment composition;
- dependency graph;
- issue and commit semantics.

Probabilistic links require confidence and explanation.

A.8. Link confidence

A link should include:

```yaml
relationship: INCLUDES_CHANGE
source_entity_id: deployment_8842
target_entity_id: ec_01J...
confidence: 0.91
confidence_band: high
method: artifact_digest_and_commit_graph
rule_version: 3.2.1
evidence:
  - image_digest: sha256:...
  - build_id: build_771
  - commit_sha: 93ab...
created_at: 2026-07-18T16:33:10Z
```

Suggested confidence bands:

- `confirmed`: explicit, cryptographically or transactionally grounded;
- `high`: strong deterministic evidence with negligible ambiguity;
- `medium`: multiple consistent signals but plausible alternatives;
- `low`: weak inference suitable only for exploration;
- `rejected`: reviewed and determined incorrect.

Published metrics should define which bands are eligible.

A.9. Provenance model

Every canonical record should preserve:

- source system;
- source tenant or instance;
- native entity identifier;
- source event identifier;
- source event time;
- ingestion time;
- schema version;
- transformation version;

- identity-resolution version;
- reconstruction rule or model version;
- confidence;
- correction history;
- retention classification.

Derived metrics should remain traceable to these records.

A.10. Temporal ownership

Ownership must be represented as a time interval.

```
owner_team_id: team_checkout
owned_entity_type: service
owned_entity_id: payment_authorizer
valid_from: 2026-01-10T00:00:00Z
valid_to: 2026-09-30T23:59:59Z
source: service_catalog
confidence: confirmed
```

Using current ownership to interpret historical events can misattribute work, incidents, and outcomes.

A.11. Work decomposition and aggregation

Engineering Changes may form a hierarchy or graph.

Useful relationships include:

- parent and child;
- split from;
- merged into;
- depends on;
- blocks;
- enables;
- corrective follow-up;
- experiment variant of;
- rollout wave of.

The model should preserve atomic changes while allowing aggregation into initiatives, programs, releases, and portfolios.

A.12. Change boundaries

A boundary should reflect shared intent and consequence, not merely implementation convenience.

Possible indicators of one Engineering Change include:

- one coherent intended outcome;
- common acceptance decision;

- shared rollout and rollback unit;
- common exposure population;
- unified outcome observation.

A large initiative should generally contain multiple Engineering Changes rather than becoming one indefinitely open record.

A.13. AI and agent provenance

For machine participants, record:

- provider;
- model family and version;
- deployment or endpoint identifier;
- system configuration;
- prompt or instruction version;
- retrieval sources and context policy;
- tool permissions;
- initiating actor;
- delegation chain;
- sampling and inference parameters where relevant;
- response or action identifier;
- safety and policy configuration;
- cost and latency.

Sensitive prompt content may require restricted storage or hashing, but configuration identity should remain auditable.

A.14. Correction model

Canonical evidence must support correction without erasing history.

A correction should record:

- corrected entity or relationship;
- previous value;
- new value;
- reason;
- correcting actor;
- timestamp;
- evidence;
- downstream metrics requiring recomputation.

Corrections should create a new version or event rather than silently mutate historical facts.

A.15. Deletion and retention

Deletion policy should distinguish:

- immutable operational evidence required for audit;
- sensitive detailed telemetry;
- derived aggregates;
- model inputs;
- generated explanations;
- employment-related records;
- experiment data.

Where hard deletion is required, lineage should indicate that source evidence was removed under policy without retaining prohibited content.

A.16. Canonical event envelope

A minimal event envelope may use the following structure:

```
{
  "event_id": "evt_01J...",
  "event_type": "review.finding.created",
  "event_version": "1.1.0",
  "occurred_at": "2026-07-18T12:44:08.217Z",
  "observed_at": "2026-07-18T12:44:09.004Z",
  "source": {
    "system": "code_review",
    "instance": "enterprise-main",
    "native_event_id": "884391"
  },
  "actor": {
    "actor_id": "agent_security_review_v4",
    "actor_type": "ai_agent"
  },
  "engineering_change_id": "ec_01J...",
  "subject": {
    "entity_type": "finding",
    "entity_id": "finding_9201"
  },
  "payload": {},
  "provenance": {
    "connector_version": "2.4.0",
    "schema_mapping_version": "1.8.2"
  },
  "classification": "internal_restricted"
}
```

Payload schemas should be registered and versioned separately.

A.17. Data quality dimensions

The reference model should report:

A.17.1. Completeness

Are expected source events and entities present?

A.17.2. Timeliness

How long after occurrence did evidence become available?

A.17.3. Uniqueness

Are duplicate events or entities controlled?

A.17.4. Consistency

Do related sources agree?

A.17.5. Validity

Do values satisfy semantic and schema rules?

A.17.6. Linkage coverage

What fraction of relevant artifacts and outcomes connect to an Engineering Change?

A.17.7. Linkage confidence

How strong are those relationships?

A.17.8. Temporal correctness

Were identities and ownership resolved for the correct time?

A.17.9. Provenance completeness

Can each derived result be traced to source evidence and transformation versions?

A.18. Minimum viable implementation

A practical first implementation should support:

1. canonical actors, teams, repositories, and services;
2. immutable source events;
3. Engineering Change identifiers;
4. commit and pull-request linkage;
5. CI and build linkage;
6. deployment linkage;
7. basic lifecycle state;

8. linkage confidence and provenance;
9. correction workflow;
10. metric lineage.

Intent, outcomes, incidents, experiments, AI provenance, and business linkage can then be added according to decision need.

A.19. Storage patterns

No single storage technology is required.

A common architecture uses:

- object storage for immutable raw events;
- a stream or log for canonical events;
- relational or lakehouse tables for governed facts;
- a graph projection for relationships and dependency traversal;
- a search index for evidence discovery;
- a metric store for versioned aggregates;
- a registry for schemas, models, experiments, and decisions.

The semantic contract is more important than choosing one universal database.

A.20. Query examples

A.20.1. Changes released without confirmed validation

```sql
SELECT ec.engineering_change_id,
       ec.title,
       r.released_at
FROM engineering_change ec
JOIN release_change r
  ON r.engineering_change_id = ec.engineering_change_id
LEFT JOIN validation_summary v
  ON v.engineering_change_id = ec.engineering_change_id
WHERE r.released_at >= CURRENT_DATE - INTERVAL '30' DAY
  AND COALESCE(v.confirmed_required_checks, 0) = 0;
```

A.20.2. Human validation burden by AI model version

```sql
SELECT ai.model_version,
       COUNT(DISTINCT ai.engineering_change_id) AS changes,
       SUM(rv.human_validation_minutes) AS validation_minutes,
       AVG(rv.human_validation_minutes) AS avg_minutes_per_change
FROM ai_participation ai
JOIN review_effort rv
  ON rv.engineering_change_id = ai.engineering_change_id
GROUP BY ai.model_version;
```

A.20.3. Deployment linkage quality

```sql
SELECT confidence_band,
       COUNT(*) AS links,
       COUNT(DISTINCT deployment_id) AS deployments,
       COUNT(DISTINCT engineering_change_id) AS changes
FROM deployment_change_link
WHERE deployed_at >= CURRENT_DATE - INTERVAL '7' DAY
GROUP BY confidence_band;
```

A.21. Conformance levels

A.21.1. Level 1: identifiable

Engineering Changes have durable identifiers and basic artifact linkage.

A.21.2. Level 2: traceable

CI, builds, deployments, releases, and provenance are connected.

A.21.3. Level 3: measurable

Governed lifecycle, effort, quality, complexity, and cost metrics are supported.

A.21.4. Level 4: experimental

Assignment, exposure, outcomes, and decision records enable causal evaluation.

A.21.5. Level 5: adaptive

Agent provenance, policy actions, continuous evaluation, and closed-loop learning are represented.

Conformance should be evaluated per domain rather than claimed once for the entire enterprise.

A.22. Reference-model anti-patterns

A.22.1. Pull request as the universal change

This excludes direct commits, configuration updates, data changes, infrastructure actions, and multi-repository outcomes.

A.22.2. Mutable current-state tables only

These make historical reconstruction and audit unreliable.

A.22.3. One change, one deployment

Real delivery systems are many-to-many.

A.22.4. Current ownership applied historically

This creates systematic attribution errors.

A.22.5. Inferred links without confidence

Consumers cannot judge whether metrics are trustworthy.

A.22.6. Machine actors collapsed into bot

This prevents evaluation of model, agent, configuration, and policy differences.

A.22.7. Outcome equals deployment success

A technically successful deployment may still fail to produce the intended outcome.

A.22.8. Silent correction

Historical evidence changes without traceability.

A.23. Reference-model checklist

1. Use a durable, vendor-independent Engineering Change identifier.
2. Preserve immutable source evidence and canonical event provenance.
3. Represent human, automation, AI, and agent actors distinctly.
4. Model artifacts, activities, validations, deployments, releases, and outcomes separately.
5. Support many-to-many relationships.
6. Record event time, ingestion time, and temporal ownership.
7. Attach confidence and method to inferred links.
8. Keep observed evidence distinct from model inference.
9. Version schemas, transformations, metrics, and reconstruction logic.
10. Support corrections without erasing history.
11. Publish linkage coverage and quality with every dependent metric.
12. Expand the model according to consequential decisions, not abstract completeness.

Appendix B: Metric Specification Standard

***This appendix defines a complete specification format for Engineering Intelligence metrics. Its** purpose is to prevent ambiguous, unstable, or misleading measures from entering dashboards, experiments, procurement decisions, policy systems, or executive reviews. A governed metric is treated as a versioned analytical product with explicit purpose, semantics, population, formula, data lineage, quality thresholds, uncertainty, interpretation limits, and retirement criteria.*

B.1. Why a metric standard is necessary

A metric name is not a definition.

Terms such as cycle time, productivity, review effort, deployment frequency, quality, acceptance rate, and AI effectiveness can conceal materially different calculations.

Two teams may publish a metric with the same name while differing in:

- unit of analysis;
- start and end events;
- eligible population;
- treatment of missing data;
- inclusion of bots and AI agents;
- aggregation method;
- time zone;
- outlier handling;
- complexity adjustment;
- confidence requirements;
- outcome window.

A metric specification makes these choices explicit and reviewable.

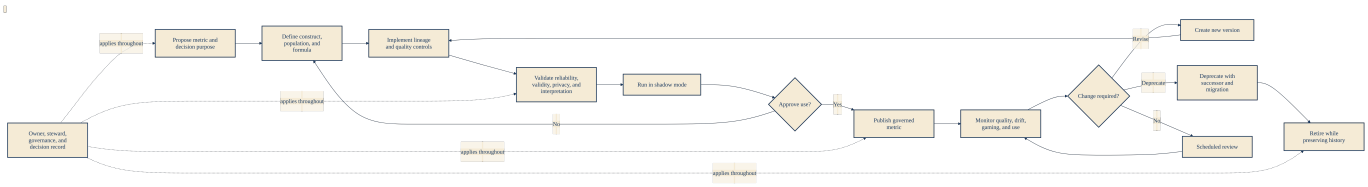
B.2. Governing principle

Every metric should answer four questions:

1. What decision is this metric intended to support?
2. What construct does it represent?
3. How is it computed from evidence?
4. Under what conditions should it not be used?

A metric that cannot answer these questions should remain exploratory.

B.3. Metric specification lifecycle



The lifecycle includes proposal, design, validation, approval, publication, monitoring, revision, deprecation, and retirement.

B.4. Required specification fields

B.4.1. Metric identity

Each metric requires:

- stable metric identifier;
- human-readable name;
- semantic version;
- lifecycle status;
- owning team;
- accountable owner;
- technical steward;
- date created;
- last review date;
- next review date.

Example:

```yaml
metric_id: ei.review.human_effort_minutes
name: Human Review Effort
version: 2.1.0
status: governed
owner_team: engineering_intelligence
accountable_owner: director_developer_productivity
technical_steward: measurement_platform
created_at: 2026-01-12
last_reviewed_at: 2026-07-01
next_review_at: 2026-10-01
```

B.4.2. Purpose

State the decision or workflow the metric supports.

Good purpose statement:

> Estimate the human attention consumed by pull-request validation so that review-routing and AI-review interventions can be evaluated economically.

Weak purpose statement:

Track review productivity.

The purpose should identify the user, decision, and intended action.

B.4.3. Construct

Name and define the underlying concept.

For example:

- construct: human validation effort;
- definition: active human attention spent evaluating and responding to an Engineering Change before acceptance;
- excluded constructs: calendar delay, author implementation effort, automated review runtime.

The construct definition should remain independent of the current instrumentation method.

B.4.4. Claim type

Declare whether the metric is used descriptively, predictively, causally, or prescriptively.

A metric may support several claim types, but each use must be documented separately.

B.4.5. Unit of analysis

Examples include:

- Engineering Change;
- pull request;
- deployment;
- incident;
- review finding;
- actor-day;
- team-week;
- repository-month;
- experiment assignment unit.

The unit of analysis determines valid aggregation and interpretation.

B.4.6. Population

Define eligibility precisely.

A population definition should include:

- organizations or domains;
- repositories or services;
- change types;
- lifecycle states;
- date range;

- risk or complexity criteria;
- actor types;
- exclusion rules;
- minimum evidence quality.

Example:

```yaml
population:
  unit: engineering_change
  include:
    - primary_repository_type: application
    - accepted_at: not_null
    - review_mode: pull_request
  exclude:
    - generated_only_change: true
    - emergency_bypass: true
    - linkage_confidence_below: high
```

B.4.7. Measurement interval

Specify:

- event-time basis;
- reporting window;
- observation delay;
- time zone;
- partial-period behavior;
- late-arriving-event policy.

Metrics based on rare outcomes may require a maturity delay before publication.

B.4.8. Formula

Provide a mathematical or executable definition.

Example:

$$HumanReviewEffort(c) = sum_{(}rin R_c) ActiveReviewMinutes(r)$$

Where R_c is the set of eligible human review sessions linked to Engineering Change c.

The specification must define every term.

B.4.9. Numerator and denominator

For rates and ratios, define both explicitly.

Example:

$$FindingAcceptanceRate = (AcceptedUniqueFindings)/(EligibleUniqueFindings)$$

Questions that must be answered include:

- Are duplicates removed?
- Are unresolved findings included?
- Are findings weighted by severity?
- Is the denominator based on produced or viewed findings?
- What happens when the denominator is zero?

B.4.10. Aggregation

Define valid aggregation functions.

Possible aggregations include:

- count;
- sum;
- mean;
- median;
- percentile;
- trimmed mean;
- weighted mean;
- rate;
- hazard;
- distribution;
- posterior estimate.

Metrics should declare whether they are additive.

For example, medians cannot be averaged safely across teams without access to the underlying distribution.

B.4.11. Dimensions

Declare approved breakdowns such as:

- organization;
- team;
- repository;
- service;
- change type;
- complexity band;
- risk class;
- language;
- AI model version;
- treatment group;
- deployment environment.

Dimensions with privacy or re-identification risk should be restricted.

B.4.12. Source evidence

List all required source entities and events.

Example:

```
sources:
  - canonical.review_session
  - canonical.review_activity
  - canonical.actor_identity
  - canonical.engineering_change_link
  - canonical.pull_request_state
```

Each source should include freshness, completeness, ownership, and lineage requirements.

B.4.13. Transformation lineage

Document the full path from raw source to published metric.

This includes:

- connector version;
- canonical schema version;
- transformation jobs;
- identity-resolution version;
- Engineering Change reconstruction version;
- metric SQL or code version;
- aggregation layer;
- dashboard or API consumer.

B.4.14. Data-quality requirements

Declare minimum publication thresholds.

Possible requirements include:

- source freshness under six hours;
- event completeness above 98 percent;
- Engineering Change linkage coverage above 90 percent;
- high-or-confirmed confidence above 85 percent;
- unresolved identity below 2 percent;
- no active processing incident;
- minimum sample size;
- stable metric version.

A metric should publish an unavailable state when thresholds fail.

B.4.15. Missing-data policy

Missingness must not be silently treated as zero.

The specification should distinguish:

- not applicable;
- not observed;
- source unavailable;
- unresolved linkage;
- delayed outcome;
- intentionally excluded;
- true zero.

Imputation methods, if any, must be stated and sensitivity-tested.

B.4.16. Bot and AI treatment

Specify how activity from the following is handled:

- service accounts;
- deterministic bots;
- AI assistants;
- autonomous agents;
- vendor systems;
- human actions initiated through automation.

Machine-generated activity should not be combined with human effort unless the construct explicitly requires it.

B.4.17. Complexity and work-mix adjustment

If comparisons may be affected by work mix, define adjustment variables.

Examples include:

- code and configuration surface;
- dependency reach;
- novelty;
- risk classification;
- number of repositories;
- service criticality;
- review specialization;
- change type;
- ownership fragmentation.

The unadjusted distribution should remain available alongside adjusted results.

B.4.18. Uncertainty

Every inferential or sampled metric requires an uncertainty method.

Options include:

- confidence intervals;
- credible intervals;
- bootstrap intervals;
- standard errors;
- control limits;
- posterior distributions;
- sensitivity ranges.

Small populations should not be displayed with false precision.

B.4.19. Validity evidence

Document available evidence for:

- construct validity;
- criterion validity;
- convergent validity;
- discriminant validity;
- ecological validity;
- measurement invariance.

The specification should state what remains unvalidated.

B.4.20. Reliability evidence

Possible evidence includes:

- test-retest stability;
- inter-rater agreement;
- identity-resolution consistency;
- reconstruction agreement;
- reproducible computation;
- model stability;
- source consistency.

B.4.21. Interpretation

State what a high or low value may indicate.

Example:

> Higher human review effort may reflect greater complexity, poor change preparation, weak automated validation, specialist scarcity, or unusually careful review. It should not be interpreted as reviewer inefficiency without additional evidence.

Interpretation guidance should include plausible alternative explanations.

B.4.22. Prohibited interpretations

Every governed metric should list unsupported uses.

Examples include:

- individual performance ranking;
- direct comparison across teams without work-mix context;
- causal claims from descriptive movement;
- cost savings claims without capacity realization;
- quality claims before the outcome window matures;
- comparison across metric versions without restatement.

B.4.23. Guardrails

Define metrics that must be considered together.

For example, an AI code-generation metric might require guardrails for:

- severe defects;
- rollback rate;
- human validation effort;
- security findings;
- codebase complexity;
- long-term maintainability.

A primary metric should never be optimized without its declared guardrails.

B.4.24. Privacy classification

Specify:

- data classification;
- minimum aggregation threshold;
- approved audiences;
- permitted purposes;
- retention period;
- whether individual-level display is allowed;
- whether employment use is prohibited.

B.4.25. Access policy

Document roles permitted to view:

- raw evidence;
- row-level metric data;
- team aggregates;
- enterprise aggregates;
- experiment results;

- restricted dimensions.

B.4.26. Behavioral risk

Describe how the metric might be gamed or distort behavior.

Examples include:

- splitting changes to reduce apparent size;
- delaying status transitions;
- avoiding difficult work;
- producing low-value findings;
- suppressing incident linkage;
- moving work outside instrumented systems.

The metric owner should define monitoring for these effects.

B.4.27. Decision thresholds

Thresholds must be justified.

A threshold may originate from:

- policy requirement;
- operational capacity;
- economic break-even;
- experiment decision rule;
- historical control limit;
- risk tolerance;
- external standard.

Arbitrary red-yellow-green boundaries should be avoided.

B.4.28. Versioning rules

Use semantic versioning principles:

- patch: implementation correction without intended semantic change;
- minor: backward-compatible extension or additional dimension;
- major: change to construct, population, formula, unit, or interpretation.

Historical results should be restated or clearly separated after a major version change.

B.4.29. Monitoring

A production metric should be monitored for:

- source freshness;
- completeness;
- distribution shift;

- sudden discontinuity;
- missingness;
- subgroup behavior;
- usage outside approved purpose;
- gaming indicators;
- cost;
- consumer dependencies.

B.4.30. Review and retirement

Retirement criteria may include:

- no active decision use;
- construct no longer valid;
- source semantics changed;
- persistent behavioral distortion;
- replacement by a better measure;
- unacceptable privacy risk;
- maintenance cost exceeds value.

Retired metrics should remain discoverable with a reason and successor where applicable.

B.5. Complete metric template

```yaml
metric:
  id: ei.change.quality_adjusted_cycle_time
  name: Quality-Adjusted Change Cycle Time
  version: 1.0.0
  status: proposed
  owner: engineering_intelligence
  steward: measurement_platform

purpose:
  decision: Evaluate workflow interventions that trade speed against quality risk
  users:
    - engineering_leaders
    - developer_experience
    - platform_owners

construct:
  name: quality_adjusted_delivery_latency
  definition: Time required to produce an accepted outcome, adjusted for observed post-release
failure consequence
  exclusions:
    - individual_productivity

claim_type:
  - descriptive
  - causal_when_used_in_registered_experiment

unit_of_analysis: engineering_change
```

```yaml
population:
  include:
    - released_change: true
  exclude:
    - outcome_window_incomplete: true
    - linkage_confidence_below: high

interval:
  start_event: first_substantive_activity
  end_event: release_exposure
  outcome_window_days: 30
  time_basis: event_time_utc

formula:
  expression: cycle_minutes * quality_penalty_multiplier
  quality_penalty_multiplier:
    no_adverse_outcome: 1.0
    minor_defect: 1.2
    rollback: 2.0
    severe_incident: 5.0

aggregation:
  primary: median
  secondary:
    - p75
    - p90
    - distribution
  additive: false

sources:
  - engineering_change
  - activity_event
  - release_exposure
  - incident_change_link
  - rollback_event

quality_gates:
  linkage_coverage_min: 0.90
  confirmed_or_high_link_min: 0.85
  source_freshness_hours_max: 12
  minimum_population: 50

missing_data:
  incomplete_outcome_window: suppress
  unknown_release_time: exclude_and_report
  unresolved_incident_link: sensitivity_analysis

uncertainty:
  method: stratified_bootstrap
  interval: 95_percent

validity:
  known_limitations:
    - penalty_weights_are_policy_values_not_natural_constants
    - rare_incidents_create_high_variance

prohibited_uses:
  - individual_performance_ranking
  - cross_team_comparison_without_complexity_adjustment
  - financial_savings_without_economic_model
```

```yaml
privacy:
  classification: internal
  minimum_group_size: 10
  individual_display: prohibited

review:
  cadence: quarterly
  next_review: 2026-10-01
```

B.6. Metric dependency graph

Metrics should declare dependencies on:

- canonical entities;
- derived features;
- models;
- identity mappings;
- linkage rules;
- reference data;
- other metrics.

A change to a dependency should identify affected metrics and consumers.

B.7. Metric bundles

Important decisions should use bundles rather than isolated measures.

A bundle may contain:

- primary outcome;
- guardrails;
- mechanism indicators;
- data-quality measures;
- economic measures;
- adoption or exposure measures.

For an AI review evaluation:

- primary outcome: human review effort;
- guardrails: severe defect and rollback outcomes;
- mechanism: actionable finding recall and ranking;
- economic: inference plus validation cost;
- exposure: eligible changes receiving the configured reviewer;
- data quality: change linkage and outcome maturity.

B.8. Metric maturity levels

B.8.1. Level 0: draft

The construct and intended purpose are still being explored.

B.8.2. Level 1: defined

Population, formula, and source evidence are documented.

B.8.3. Level 2: validated

Data quality, reliability, and initial validity evidence are acceptable.

B.8.4. Level 3: governed

Ownership, privacy, prohibited uses, versioning, and review are operational.

B.8.5. Level 4: decision-grade

The metric has demonstrated usefulness for a consequential decision and includes uncertainty and guardrails.

B.8.6. Level 5: policy-grade

The metric may influence automated or formal policy under continuous monitoring, appeal, and rollback controls.

Maturity should be assigned per use case, not universally.

B.9. Validation process

A proposed metric should pass through:

1. construct review;
2. domain review;
3. source and lineage review;
4. computation testing;
5. historical backtest;
6. missingness analysis;
7. reliability analysis;
8. practitioner validation;
9. privacy and governance review;
10. shadow publication;
11. decision-use approval.

B.10. Metric tests

Automated tests should include:

- schema tests;
- unit tests for formulas;
- population inclusion and exclusion tests;
- duplicate handling;
- zero-denominator behavior;
- time-boundary behavior;
- late-event handling;
- version compatibility;
- known-example fixtures;
- distribution-change alerts;
- lineage completeness.

B.11. Publication contract

A published metric response should include more than a value.

Example:

```
{
  "metric_id": "ei.review.human_effort_minutes",
  "version": "2.1.0",
  "value": 37.4,
  "unit": "minutes",
  "aggregation": "median",
  "population_size": 1842,
  "period": {
    "start": "2026-06-01",
    "end": "2026-06-30"
  },
  "uncertainty": {
    "lower": 35.9,
    "upper": 39.2,
    "level": 0.95
  },
  "quality": {
    "status": "publishable",
    "source_completeness": 0.992,
    "change_linkage_coverage": 0.947,
    "freshness_hours": 2.1
  },
  "interpretation_url": "metric://ei.review.human_effort_minutes/2.1.0"
}
```

Consumers should not receive a bare number without version and quality context.

B.12. Change management

When a metric changes:

1. create a proposal;
2. identify semantic and consumer impact;
3. run old and new versions in parallel;
4. quantify differences;
5. review interpretation changes;
6. communicate the effective date;
7. restate history where appropriate;
8. preserve the previous version;
9. update dashboards, experiments, alerts, and decisions.

B.13. Metric incident management

A metric incident occurs when a published measure is materially wrong, delayed, misused, or semantically invalid.

Incident response should include:

- suppressing affected values;
- identifying impacted decisions and consumers;
- preserving erroneous outputs for audit;
- correcting source or transformation logic;
- recomputing affected periods;
- publishing an explanation;
- reviewing whether controls failed.

B.14. Example: deployment frequency

A weak definition is “number of deployments per week.”

A complete specification must answer:

- What is a deployment?
- Are retries counted?
- Are non-production environments included?
- Is a progressive rollout one deployment or many?
- Are rollbacks deployments?
- Are infrastructure and configuration changes included?
- Is the unit repository, service, team, or enterprise?
- Are failed deployments counted?
- How are shared deployments attributed?

The metric may be useful, but only after these choices are explicit.

B.15. Example: AI acceptance rate

Acceptance rate is especially easy to misuse.

A specification must define:

- suggestion unit;
- whether partial edits count;
- observation window;
- duplicate suggestions;
- invisible accepted content;
- later rework or deletion;
- user exposure;
- eligible denominator;
- quality outcome linkage.

Acceptance is evidence of immediate use, not proof of correctness or value.

B.16. Example: review precision

Review precision may be defined as:

$$Precision = (AcceptedActionableFindings)/(EligibleProducedFindings)$$

The specification should additionally define:

- finding deduplication;
- adjudication process;
- actionable criteria;
- severity treatment;
- unresolved findings;
- human and AI producer identity;
- rank-aware variants;
- annotation agreement.

Precision should be paired with recall or coverage because a system can achieve high precision by producing very few findings.

B.17. Anti-patterns

B.17.1. Formula-only definition

The construct, purpose, and interpretation remain ambiguous.

B.17.2. Metric without population

Values change because eligibility changed, but consumers interpret the movement as system performance.

B.17.3. Missing equals zero

Source failure appears as improvement.

B.17.4. Versionless dashboard

Semantic changes silently alter historical trends.

B.17.5. Universal threshold

One boundary is applied across domains with different risk and work mix.

B.17.6. Bare point estimate

Small or noisy samples are presented with false certainty.

B.17.7. Individual ranking by system telemetry

Weak proxies become employment judgments and corrupt behavior.

B.17.8. Metric without retirement

Obsolete measures accumulate and continue influencing decisions.

B.18. Metric specification checklist

1. Assign a durable identifier, owner, steward, and semantic version.
2. State the decision purpose and underlying construct.
3. Define claim type, unit, population, interval, formula, and aggregation.
4. Document source evidence and full transformation lineage.
5. Publish data-quality thresholds and missing-data behavior.
6. Separate human, automation, and AI activity appropriately.
7. Include complexity adjustment where comparison requires it.
8. Report uncertainty, validity, reliability, and limitations.
9. Define interpretation, prohibited uses, guardrails, and privacy controls.
10. Monitor drift, gaming, misuse, quality, and dependency changes.
11. Review metrics on a declared cadence.
12. Deprecate and retire metrics that no longer improve decisions.

Appendix C: Experiment and Causal Study Protocol

This appendix defines a practical protocol for randomized experiments, phased rollouts, natural *experiments, and other causal studies in software engineering organizations. It standardizes how questions are framed, units are assigned, treatment is delivered, exposure is observed, outcomes and guardrails are measured, interference is handled, analyzes are predeclared, and decisions are recorded. The objective is not statistical ceremony. It is to make consequential engineering interventions testable, auditable, reproducible, and safe.*

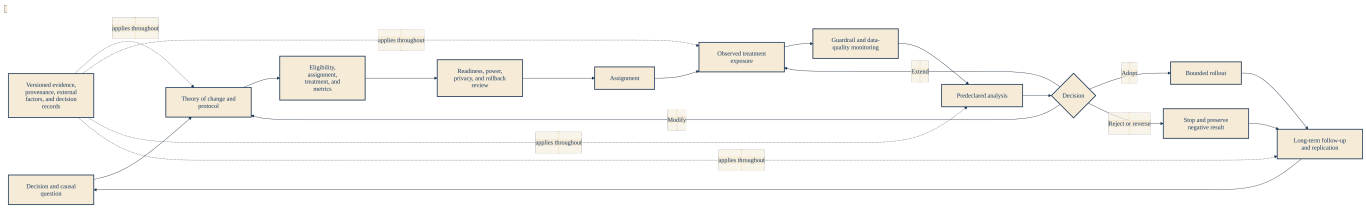
C.1. Scope

The protocol applies to interventions such as:

- AI coding, review, testing, and operational tools;
- review-routing or approval-policy changes;
- CI, build, and test infrastructure changes;
- merge queues and release controls;
- developer-environment improvements;
- architecture and ownership changes;
- reliability practices;
- platform migrations;
- organizational and process policies.

The same protocol can be adapted for randomized trials, cluster-randomized studies, stepped-wedge rollouts, switchback experiments, regression discontinuity, difference-in-differences, interrupted time series, matched observational studies, and synthetic controls.

C.2. The experiment lifecycle



A causal study moves through question definition, design, readiness, assignment, exposure, monitoring, analysis, decision, follow-up, and replication.

C.3. Start with a decision

The study should name the decision it will inform.

Examples:

- Should the AI reviewer become the default for medium-risk pull requests?
- Should low-risk repositories use automated approval instead of mandatory specialist review?
- Should the organization fund a larger CI cache tier?
- Should a merge queue be required for highly coupled services?

A study without a decision owner or decision rule risks becoming informative but operationally irrelevant.

C.4. Study charter

Every study should have a charter containing:

- study identifier;
- title;
- sponsor;
- accountable decision owner;
- study owner;
- domain partners;
- hypothesis;
- intervention;
- eligible population;
- assignment unit;
- primary outcome;
- guardrails;
- expected duration;
- stopping criteria;
- privacy classification;
- approval status.

Example:

```
study_id: exp_ai_review_2026_04
name: AI review for medium-risk changes
sponsor: developer_productivity
owner: engineering_intelligence
decision_owner: vp_engineering_platform
hypothesis: AI review reduces human validation effort without increasing severe escaped defects
assignment_unit: repository
primary_outcome: ei.review.human_effort_minutes@2.1.0
guardrails:
  - ei.quality.severe_escape_rate@1.3.0
  - ei.change.rollback_rate@2.0.0
  - ei.review.author_rework_minutes@1.0.0
```

C.5. Causal question

Frame the question in counterfactual terms.

For an eligible Engineering Change, compare the outcome under treatment with the outcome that would have occurred

without treatment.

Because both cannot be observed for the same change at the same time, the study design must create a credible comparison.

C.6. Hypothesis specification

A useful hypothesis defines:

- population;
- treatment;
- comparator;
- primary outcome;
- expected direction or minimum effect;
- guardrails;
- observation window;
- decision consequence.

Example:

> For eligible medium-risk Engineering Changes, enabling AI review will reduce median active human review effort by at least 15 percent over eight weeks, without increasing severe escaped defects or rollback probability during the following thirty days.

C.7. Theory of change

Document the proposed mechanism.

For an AI reviewer, the theory may be:

1. the tool identifies actionable defects before human review;
2. authors correct some defects earlier;
3. human reviewers spend less time finding routine issues;
4. specialist attention shifts toward architecture and domain risk;
5. quality remains stable or improves.

Possible counter-mechanisms include:

- reviewers spend time validating low-value findings;
- authors create more rework;
- reviewers become less attentive;
- comment volume increases queue congestion;
- the tool performs poorly for particular languages or change types.

Mechanism measures should be selected before launch.

C.8. Eligibility

Eligibility rules must be executable and versioned.

Specify:

- inclusion criteria;
- exclusions;
- risk and complexity boundaries;
- domain or repository scope;
- actor requirements;
- temporal constraints;
- prior treatment restrictions;
- minimum evidence quality.

Eligibility should be evaluated before assignment using only information available at that time.

C.9. Unit of assignment

The assignment unit may be:

- Engineering Change;
- pull request;
- engineer;
- repository;
- team;
- service;
- organization;
- time period;
- deployment wave.

Choose the lowest unit that limits interference and operational complexity without sacrificing statistical efficiency.

Assigning individual changes may fail when treatment changes reviewer behavior across the entire repository. In that case, repository or team assignment may be more defensible.

C.10. Unit of analysis

The analysis unit may differ from the assignment unit.

For example:

- repositories are assigned;
- Engineering Changes produce outcomes;
- standard errors are clustered by repository.

The analysis must respect the assignment structure.

C.11. Treatment specification

A treatment is more than a feature-flag value.

Record:

- tool and version;
- model and version;
- prompt or policy configuration;
- permissions;
- user-interface behavior;
- rollout state;
- training or communication;
- fallback behavior;
- treatment start and end;
- configuration changes during the study.

A material treatment change may require a new study phase or treatment variant.

C.12. Comparator

The comparator may be:

- current workflow;
- placebo or sham intervention;
- alternative tool;
- alternative configuration;
- delayed treatment;
- historical or synthetic control.

The comparator must be described with the same precision as the treatment.

C.13. Randomization

Randomization should be deterministic, reproducible, and auditable.

A common pattern is:

```
assignment = hash(study_id || unit_id || salt) mod number_of_variants
```

The assignment service should record:

- eligible unit;
- assignment timestamp;
- variant;
- algorithm and version;
- salt or seed reference;

- strata;
- override status;
- exclusion reason.

Manual overrides should be rare, logged, and analyzed.

C.14. Stratification and blocking

Stratification can improve balance for important predictors such as:

- repository size;
- baseline cycle time;
- programming language;
- risk class;
- business domain;
- prior tool usage;
- change volume;
- team size.

Strata should be selected before outcome inspection and kept limited enough to avoid sparse cells.

C.15. Cluster randomization

Cluster assignment is appropriate when treatment or behavior spreads within a group.

Potential clusters include teams, repositories, services, or review groups.

Cluster trials must account for:

- intracluster correlation;
- fewer effective independent units;
- cluster-size imbalance;
- cross-cluster dependencies;
- cluster attrition;
- cluster-level implementation differences.

Power calculations based only on the number of Engineering Changes will be overly optimistic.

C.16. Switchback experiments

A switchback alternates treatment by time window.

It is useful when:

- capacity or system behavior changes globally;
- simultaneous treatment and control are operationally difficult;
- effects are expected to occur quickly;
- carryover can be bounded.

Specify window length, washout period, seasonality controls, and treatment schedule before launch.

C.17. Stepped-wedge rollout

In a stepped-wedge design, clusters adopt treatment in randomized or defensible waves until all receive it.

This design can fit enterprise rollouts, but requires care because calendar time is correlated with treatment. Analysis must adjust for time trends and rollout-wave effects.

C.18. Exposure

Assignment does not prove treatment exposure.

Record whether the treatment was:

- available;
- invoked;
- visible to the user;
- completed successfully;
- used before the relevant decision;
- overridden or bypassed;
- configured as assigned.

Example exposure record:

```
{
  "study_id": "exp_ai_review_2026_04",
  "unit_id": "ec_01J...",
  "assigned_variant": "ai_review_enabled",
  "exposure_status": "completed_before_human_review",
  "exposed_at": "2026-07-18T10:14:03Z",
  "treatment_version": "reviewer-4.2/model-x-2026-07",
  "configuration_hash": "sha256:...",
  "failure_reason": null
}
```

C.19. Intention-to-treat and treatment-on-treated

The primary randomized analysis should usually be intention-to-treat: compare units according to assignment.

This preserves the causal benefit of randomization and includes real adoption friction.

Treatment-on-treated or per-protocol analyzes may be useful as secondary analyzes, but require stronger assumptions because exposure is not random.

C.20. Noncompliance

Noncompliance includes:

- treatment assigned but unavailable;
- user bypass;
- treatment contamination in control;
- partial configuration;
- treatment delivered too late;
- model or service failure.

Report noncompliance by variant and reason. Do not silently remove noncompliant units from the primary analysis.

C.21. Interference

Standard causal inference often assumes one unit's treatment does not affect another unit's outcome. Engineering systems frequently violate this assumption.

Examples include:

- reviewers work across treatment and control repositories;
- CI capacity is shared;
- a treated platform changes all dependent teams;
- knowledge learned from treatment spreads;
- one deployment contains both treatment and control changes.

The study should identify plausible interference pathways and choose assignment boundaries accordingly.

C.22. Spillover measurement

Where spillover is likely, record:

- shared reviewers;
- shared services and dependencies;
- cross-team contribution;
- treatment-related communication;
- common queue utilization;
- control units receiving treatment artifacts.

Spillover may be an important outcome rather than only a nuisance.

C.23. Primary outcome

Select one primary outcome whenever possible.

The primary metric must be:

- versioned;
- decision-relevant;
- observable within the study window;
- sufficiently reliable;

- resistant to obvious gaming;
- accompanied by uncertainty.

Multiple equally primary outcomes increase ambiguity and false-positive risk.

C.24. Secondary outcomes

Secondary outcomes may capture:

- mechanisms;
- distributional effects;
- adoption;
- cost;
- experience;
- longer-term quality;
- exploratory signals.

They should be labeled confirmatory or exploratory.

C.25. Guardrails

Guardrails protect against harmful optimization.

A guardrail specification should include:

- metric and version;
- acceptable boundary;
- monitoring cadence;
- maturity delay;
- escalation owner;
- pause or stop rule.

Examples include severe defects, security incidents, rollback, latency, cost, reviewer overload, and subgroup harm.

C.26. Outcome windows

Different outcomes mature at different rates.

Examples:

- review effort: immediate;
- cycle time: at acceptance or release;
- rollback: days;
- escaped defects: weeks;
- skill development: months;
- architecture effects: quarters.

The study should distinguish launch duration from outcome-follow-up duration.

C.27. Baseline period

A baseline helps assess balance, seasonality, power assumptions, and data quality.

Do not use post-assignment baseline variables in ways that introduce bias.

Useful baseline features include prior outcome distributions, change mix, repository activity, team size, and source coverage.

C.28. Sample-size and power planning

Power planning should specify:

- minimum detectable effect;
- expected variance;
- significance or posterior decision criterion;
- desired power;
- assignment ratio;
- cluster correlation;
- attrition or missingness;
- sequential design adjustments;
- expected eligible volume.

Power should be based on the decision consequence, not only on what sample size is convenient.

C.29. Rare outcomes

Severe incidents and critical defects may be too rare for conventional experiment duration.

Possible strategies include:

- composite guardrails with careful interpretation;
- longer follow-up;
- Bayesian hierarchical models;
- severity-weighted outcomes;
- validated leading indicators;
- noninferiority margins;
- pooled replication across cohorts.

Absence of observed severe events does not establish safety.

C.30. Noninferiority

A noninferiority design may be appropriate when treatment is expected to reduce cost or effort while preserving quality.

The noninferiority margin must represent the largest acceptable quality degradation and should be justified before the study.

A poorly chosen wide margin can make an unsafe tool appear acceptable.

C.31. Economic outcomes

Include direct and indirect economics:

- vendor and inference cost;
- infrastructure cost;
- human validation effort;
- implementation and support cost;
- released capacity;
- defect or incident cost;
- delay cost;
- switching and migration cost.

Distinguish realized savings, avoided cost, and modeled capacity value.

C.32. External-factor registry

Record events that may affect interpretation:

- release freezes;
- reorganizations;
- major incidents;
- staffing changes;
- model upgrades;
- platform migrations;
- policy changes;
- holidays;
- outages;
- business-cycle changes.

This registry should be created continuously, not reconstructed only after surprising results.

C.33. Readiness review

Before launch, verify:

1. eligibility is executable;
2. assignment is deterministic;
3. treatment is deployable;
4. exposure is observable;
5. primary and guardrail metrics are publishable;
6. data-quality alerts exist;
7. privacy approval is complete;
8. power assumptions are documented;
9. stop and rollback mechanisms work;

10. decision owner accepts the analysis plan.

No study should launch if exposure or guardrails are unobservable.

C.34. Pre-analysis plan

The plan should specify:

- estimand;
- population;
- assignment and analysis units;
- treatment variants;
- primary and secondary outcomes;
- guardrails;
- covariates;
- exclusion rules;
- missing-data policy;
- model specification;
- clustering and weighting;
- multiplicity handling;
- subgroup analyzes;
- sequential monitoring;
- decision criteria;
- robustness checks.

Version and timestamp the plan before unblinded outcome analysis.

C.35. Estimand

State exactly what effect is being estimated.

Examples include:

- average treatment effect for all eligible changes;
- intention-to-treat effect for assigned repositories;
- effect among changes exposed before human review;
- noninferiority of severe-defect risk;
- distributional effect at the 90th percentile.

The estimand must match the decision.

C.36. Covariate adjustment

Pre-treatment covariates can improve precision.

Useful covariates include baseline outcome, repository, complexity, risk class, language, and prior volume.

Avoid selecting covariates based on post-treatment correlation or adjusting for mediators unless estimating a specific direct effect.

C.37. Missing outcomes

Classify missingness by reason:

- source failure;
- outcome window incomplete;
- unit abandoned;
- treatment failure;
- unresolved linkage;
- domain withdrawal;
- true absence of outcome.

Report missingness by variant. Use sensitivity analyzes where missingness could depend on treatment or outcome.

C.38. Sequential monitoring

Operational studies often monitor before the planned end.

Predeclare whether monitoring is for:

- safety only;
- futility;
- efficacy;
- data quality;
- operational failure.

Repeated significance testing without adjustment inflates false positives. Use appropriate group-sequential, alpha-spending, Bayesian, or always-valid methods.

C.39. Stop rules

Possible stop conditions include:

- severe guardrail breach;
- treatment security incident;
- exposure failure above threshold;
- irrecoverable data-quality failure;
- overwhelming benefit under the declared rule;
- futility;
- external change invalidating the design.

Stopping should trigger a documented decision, not automatic broad rollout.

C.40. Analysis principles

The analysis should:

- follow the pre-analysis plan;
- preserve assignment structure;
- report effect sizes and uncertainty;
- show absolute and relative effects;
- report population and missingness;
- include quality and exposure status;
- distinguish confirmatory from exploratory results;
- assess practical as well as statistical significance.

C.41. Distributional effects

Means and medians can hide important effects.

Inspect:

- percentiles;
- variance;
- tail latency;
- zero-inflation;
- severe-event concentration;
- subgroup distributions;
- probability of improvement.

An intervention may help routine changes while harming the most complex work.

C.42. Heterogeneous treatment effects

Predeclare important subgroups such as:

- complexity band;
- risk class;
- language;
- repository maturity;
- team experience;
- change type;
- reviewer specialization.

Subgroup findings should include interaction uncertainty and should not be based only on separate significance tests.

Unexpected subgroup results are hypotheses for replication unless strongly supported.

C.43. Multiple comparisons

When many outcomes, variants, or subgroups are tested, specify how false discoveries are controlled or how posterior decisions are made.

Do not hide the total number of analyzes performed.

C.44. Robustness checks

Useful checks include:

- unadjusted and adjusted estimates;
- alternative missing-data assumptions;
- alternative outcome definitions;
- exclusion of rollout anomalies;
- cluster-robust and random-effects models;
- randomization inference;
- placebo outcomes;
- pre-trend checks;
- sensitivity to linkage-confidence thresholds.

Robustness checks should test assumptions, not search for a preferred answer.

C.45. Quasi-experimental studies

When randomization is unavailable, document the identification strategy explicitly.

C.45.1. Difference-in-differences

Requires credible parallel trends absent treatment. Show pre-treatment trends and account for staggered adoption carefully.

C.45.2. Interrupted time series

Model baseline trend, seasonality, autocorrelation, intervention timing, and simultaneous events.

C.45.3. Regression discontinuity

Document assignment threshold, manipulation risk, bandwidth, continuity, and local interpretation.

C.45.4. Instrumental variables

Justify relevance, exclusion, and monotonicity assumptions. The effect often applies only to compliers affected by the instrument.

C.45.5. Synthetic control

Document donor pool, pre-treatment fit, predictor selection, and placebo comparisons.

C.45.6. Matching and weighting

Show overlap, balance, weight stability, and sensitivity to unmeasured confounding.

Causal language should reflect the credibility of the assumptions.

C.46. Decision framework

A decision should combine:

- primary effect;
- guardrails;
- uncertainty;
- economics;
- treatment fidelity;
- subgroup effects;
- operational feasibility;
- external validity;
- reversibility.

Possible decisions are adopt, bounded adopt, extend study, modify, reject, or reverse.

C.47. Decision record

Example:

```yaml
decision_id: dec_2026_081
study_id: exp_ai_review_2026_04
decision: bounded_adoption
population: medium_risk_java_and_kotlin_repositories
rationale:
  primary_effect: -18_percent_human_review_effort
  uncertainty: 95_percent_interval_-24_to_-11
  quality_guardrails: noninferior_with_current_follow_up
  economics: positive_expected_value
limitations:
  - insufficient_security_repository_sample
  - long_term_skill_effect_unknown
conditions:
  - continue_30_day_quality_follow_up
  - exclude_high_risk_services
  - repeat_for_python_repositories
review_date: 2026-11-01
```

C.48. Reporting package

The final package should include:

- executive decision summary;
- charter and theory of change;
- protocol and analysis plan;
- assignment and exposure report;
- data-quality report;
- population flow diagram;
- primary and guardrail results;
- economic analysis;
- subgroup and robustness results;
- limitations;
- decision record;
- reproducibility assets.

C.49. Reproducibility

Preserve:

- protocol version;
- metric versions;
- assignment code;
- treatment configuration;
- analysis code;
- data-extraction logic;
- environment and package versions;
- synthetic or de-identified fixtures;
- result tables;
- decision record.

Where raw data cannot be shared, provide executable specifications and aggregate checksums.

C.50. Follow-up

After a decision, continue measuring:

- longer-term quality;
- adoption and treatment drift;
- cost;
- model or tool changes;
- behavioral adaptation;
- subgroup performance;
- skill and capability effects.

A positive pilot does not permanently validate a changing intervention.

C.51. Replication

Replicate across relevant variations:

- teams;
- repositories;
- languages;
- complexity;
- risk;
- time periods;
- model versions;
- organizational contexts.

Replication should test the mechanism and boundary conditions, not merely repeat the headline metric.

C.52. Study maturity levels

C.52.1. Level 0: observational signal

A pattern suggests a question but supports no intervention claim.

C.52.2. Level 1: designed observational study

Population, confounders, and assumptions are documented.

C.52.3. Level 2: credible quasi-experiment

A defensible identification strategy estimates a local causal effect.

C.52.4. Level 3: randomized evaluation

Assignment supports a causal estimate for the study population.

C.52.5. Level 4: replicated causal evidence

Effects and boundary conditions are reproduced across relevant settings.

C.52.6. Level 5: governed operational policy

Evidence has been translated into monitored policy with rollback and revalidation.

C.53. Anti-patterns

C.53.1. Adoption as treatment

Users who choose a tool are compared with those who do not, and selection bias is ignored.

C.53.2. Assignment without exposure

The organization cannot tell whether the configured treatment actually operated.

C.53.3. Moving outcome definitions

Metrics change after results become visible.

C.53.4. Success metric without guardrails

Speed improves while quality or human burden is ignored.

C.53.5. Unit mismatch

Changes are analyzed as independent even though teams were assigned.

C.53.6. Post-treatment filtering

Units are removed based on behavior caused by treatment.

C.53.7. Dashboard significance testing

Repeated informal checks generate a favorable result by chance.

C.53.8. Rare-event safety claim

No severe event occurred in a small sample, so safety is declared.

C.53.9. Experiment without decision

Results are presented, but no owner adopts, modifies, or stops the intervention.

C.54. Protocol checklist

1. Name the consequential decision and accountable owner.
2. State the causal question, hypothesis, and theory of change.
3. Define eligibility, assignment unit, analysis unit, treatment, and comparator.
4. Make assignment deterministic and exposure observable.
5. Predeclare primary outcome, guardrails, windows, and economic measures.
6. Account for clustering, interference, spillover, noncompliance, and missingness.
7. Conduct power and readiness reviews before launch.
8. Version the pre-analysis plan before unblinded outcome analysis.
9. Monitor safety and quality under declared sequential rules.

10. Report effect sizes, uncertainty, population, exposure, and limitations.
11. Record an explicit adopt, modify, extend, reject, or reverse decision.
12. Preserve reproducibility assets and continue long-term follow-up.
13. Replicate before universal policy adoption.

Appendix D: Governance, Privacy, and Acceptable Use Standard

This appendix defines an operating standard for governing Engineering Intelligence systems. It *translates ethical principles into concrete controls for data collection, access, metric use, experimentation, AI evaluation, automated policy, privacy, retention, incident response, appeal, and audit. The objective is to enable credible organizational learning while preventing surveillance, unsupported employment decisions, discriminatory treatment, and unsafe automation.*

D.1. Purpose

Engineering Intelligence systems combine evidence from source control, planning tools, CI, deployment systems, incidents, identities, organizational structures, AI tools, and business outcomes.

That combination creates substantial value and substantial risk.

The same evidence can be used to:

- improve review routing;
- evaluate an AI tool;
- reduce queue delay;
- identify architecture constraints;
- optimize infrastructure cost;
- support portfolio decisions;

or to:

- rank individuals through weak proxies;
- infer sensitive characteristics;
- monitor private behavior;
- automate punitive decisions;
- conceal uncertainty behind authoritative dashboards;
- create irreversible policy from invalid metrics.

Governance must therefore control not only what data is collected, but how evidence is interpreted and used.

D.2. Core principles

The principles below are not a checklist; they are the constraints that keep an Engineering Intelligence program legitimate over time. Every subsequent rule in this appendix — classification, registration, retention, appeal — is an operational expression of one or more of these principles, and any exception process should trace its reasoning back to them explicitly.

D.2.1. Purpose limitation

Data and metrics may be used only for declared, reviewed purposes.

D.2.2. Data minimization

Collect the least detailed evidence needed to support the approved decision.

D.2.3. Proportionality

The intrusiveness and consequence of measurement should be proportionate to the value and risk of the decision.

D.2.4. Transparency

Affected practitioners should understand what is collected, how metrics are defined, and which uses are prohibited.

D.2.5. Contestability

People and teams must be able to challenge incorrect data, interpretations, models, and decisions.

D.2.6. Human accountability

Automated systems may recommend or execute bounded actions, but accountable organizational owners remain responsible for policy and outcomes.

D.2.7. Evidence quality

No high-consequence use should proceed when source quality, metric validity, linkage, or uncertainty is inadequate.

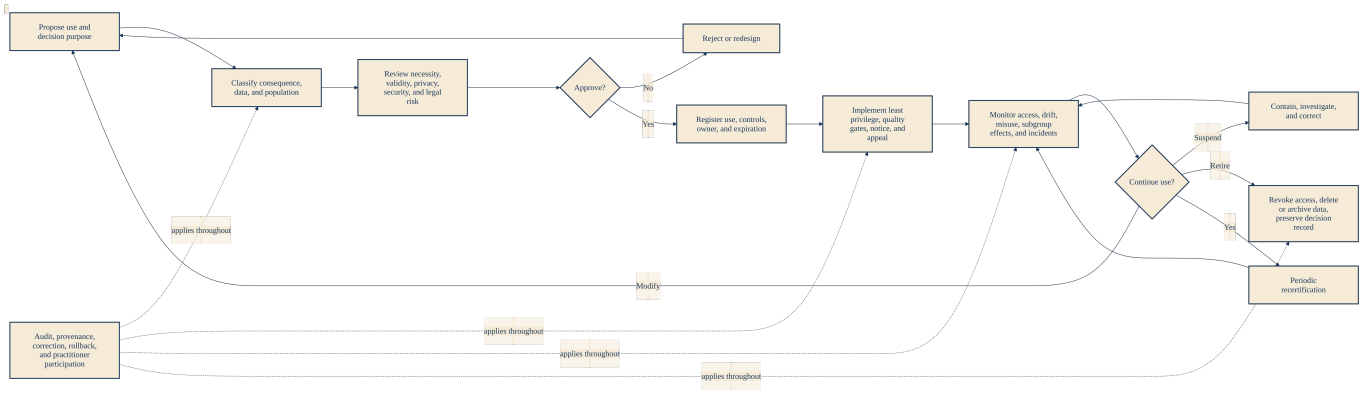
D.2.8. Temporal relevance

Evidence should be interpreted in the context in which it was produced. Old metrics and model outputs should not be treated as permanently valid.

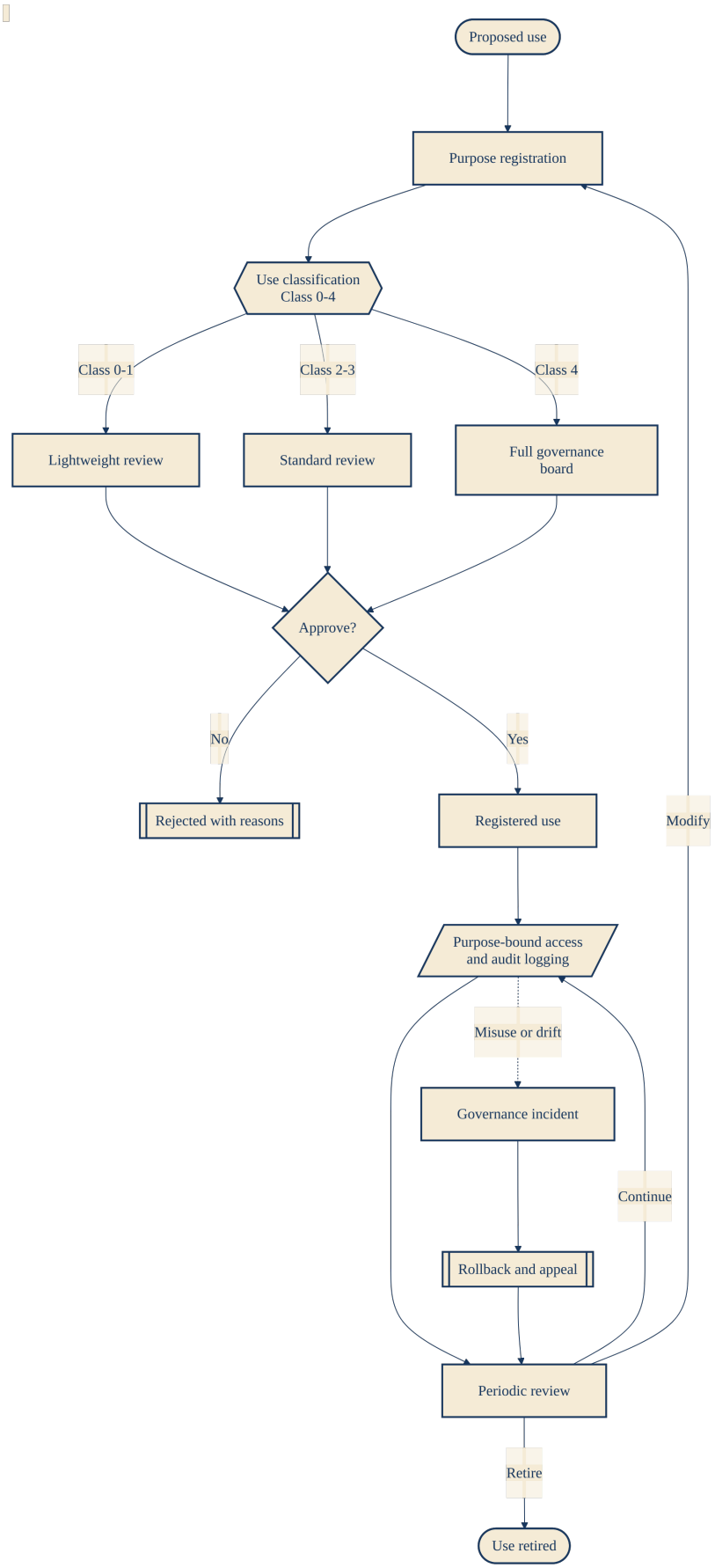
D.2.9. Separation of learning and punishment

Exploratory and operational learning systems should not be silently repurposed into employment or disciplinary systems.

D.3. Governance operating model



Every governed use moves through proposal, classification, review, approval, implementation, monitoring, reassessment, and retirement. The diagram below expands the same loop with the two mechanisms that give it teeth: the classification step routes use proposals to a review body proportionate to their consequence, and the monitoring step feeds discovered drift or misuse back into a rollback-and-appeal path rather than allowing quiet continuation. Governance without both branches degrades quickly into paperwork.


Proposed use
Purpose registration
Use classification Class 0-4
Class 0-1
Class 2-3
Class 4
Lightweight review
Standard review
Full governance board
Approve?
No
Yes
Rejected with reasons
Registered use
Modify
Purpose-bound access and audit logging
Misuse or drift
Governance incident
Continue
Rollback and appeal
Periodic review
Retire
Use retired

D.4. Roles and responsibilities

D.4.1. Executive sponsor

Provides mandate, resources, and accountability for the overall program.

D.4.2. Engineering Intelligence owner

Owns the platform, evidence quality, operating model, and adherence to this standard.

D.4.3. Data owner

Owns source semantics, permitted use, retention, and source-specific access decisions.

D.4.4. Metric owner

Owns construct definition, interpretation, versioning, quality gates, and retirement.

D.4.5. Study owner

Owns experimental design, exposure, analysis, guardrails, and reporting.

D.4.6. Domain representative

Validates local workflow meaning and operational feasibility.

D.4.7. Privacy reviewer

Assesses purpose limitation, minimization, retention, re-identification, and sensitive-data risk.

D.4.8. Security reviewer

Assesses access, integrity, isolation, model and tool risk, and adversarial threats.

D.4.9. Legal or compliance reviewer

Assesses contractual, regulatory, employment, labor, and jurisdictional obligations.

D.4.10. Practitioner representative

Provides the perspective of people whose work or teams are represented in the system.

D.4.11. Governance board

Approves high-risk uses, resolves disputed interpretations, and reviews incidents and policy changes.

D.5. Use classification

Every proposed use should be classified by consequence and sensitivity. The class determines which review body inspects the proposal, what evidence must accompany it, how deeply access is scoped, and how frequently the use is re-examined. A single misclassification — most commonly a Class 3 or 4 use being registered as Class 1 to speed approval — undermines every downstream control, so classification decisions themselves are audited and appealable.

D.5.1. Class 0: platform operations

Examples:

- connector health;
- event freshness;
- schema failures;
- metric pipeline incidents.

These uses generally involve low human consequence but may expose sensitive technical metadata.

D.5.2. Class 1: descriptive team learning

Examples:

- queue analysis;
- review-workload distribution;
- CI bottleneck discovery;
- workflow retrospectives.

Results are intended for local improvement and should use appropriate aggregation.

D.5.3. Class 2: organizational decision support

Examples:

- tool procurement;
- platform investment;
- architecture prioritization;
- staffing-capacity analysis;
- portfolio allocation.

These uses require governed metrics, uncertainty, and decision records.

D.5.4. Class 3: policy influence

Examples:

- mandatory review rules;
- risk-based deployment gates;
- automated reviewer routing;
- access or approval policy;
- organization-wide tool defaults.

These uses require causal or strong operational evidence, guardrails, monitoring, and rollback.

D.5.5. Class 4: individual or employment consequence

Examples:

- performance evaluation;
- promotion;
- compensation;
- disciplinary action;
- termination;
- individual productivity ranking.

Engineering telemetry and derived metrics should be presumed unsuitable for these uses unless a separate formal process demonstrates exceptional validity, fairness, necessity, legal authority, transparency, and due process. In most organizations, these uses should be prohibited.

D.6. Permitted uses

Subject to appropriate review and controls, permitted uses include:

- improving engineering-system reliability;
- identifying process bottlenecks;
- evaluating tools and interventions;
- estimating human attention and infrastructure cost;
- improving review, CI, deployment, and incident workflows;
- understanding architecture and dependency risk;
- supporting capacity and portfolio planning;
- measuring source and metric quality;
- conducting approved research;
- supporting practitioner-led retrospectives;
- auditing automated engineering actions.

Permission is tied to the declared purpose, not to broad access to the data.

D.7. Prohibited uses

The following uses should be prohibited by default:

- ranking individual engineers by commit, pull-request, review, or ticket volume;
- inferring individual productivity from cycle time or visible activity;
- covert monitoring of work patterns outside an approved purpose;
- using exploratory metrics in compensation or disciplinary decisions;
- collecting keystrokes, screen activity, or message content without exceptional necessity and explicit approval;
- inferring health, disability, family status, political activity, protected characteristics, or private beliefs;
- penalizing people for correcting erroneous data or challenging a metric;

- using AI-generated assessments as sole evidence for an adverse employment decision;
- publishing identifiable comparisons without legitimate need;
- reusing data for a materially different purpose without review;
- allowing vendors to train external models on organizational evidence without explicit contractual permission;
- executing irreversible high-impact actions solely from an unvalidated model or metric.

D.8. Restricted uses

Some uses are neither automatically permitted nor universally prohibited.

Examples include:

- individual coaching;
- expertise discovery;
- workload or burnout-risk indicators;
- staffing decisions;
- organizational redesign;
- fraud or insider-threat detection;
- compliance investigation;
- model-based risk gating.

These require heightened review, narrow access, evidence of necessity, minimized data, explicit due process, and ongoing monitoring.

D.9. Purpose registration

Every nontrivial use should be registered.

Example:

```
use_id: use_review_routing_2026_03
name: Risk-based specialist review routing
owner: secure_development_platform
use_class: 3
purpose: Route high-risk Engineering Changes to qualified security reviewers while reducing
unnecessary low-risk review load
population: production_application_repositories
data:
  - engineering_change_complexity
  - dependency_blast_radius
  - security_sensitive_paths
  - historical_severe_findings
metrics:
  - ei.review.specialist_wait_time@1.2.0
  - ei.quality.severe_security_escape_rate@1.0.0
decisions:
  - reviewer_assignment
prohibited_extensions:
  - engineer_performance_ranking
  - promotion_decisions
review_date: 2026-10-01
```

The registration should include purpose, owner, affected population, data, metrics, decisions, access, retention, risks, prohibited extensions, and expiration date.

D.10. Approval criteria

A use may be approved only when:

- the decision purpose is clear;
- the proposed evidence is relevant;
- less intrusive alternatives were considered;
- source and metric quality are adequate;
- affected populations are defined;
- privacy and security controls are proportionate;
- prohibited interpretations are documented;
- an accountable owner exists;
- correction and appeal are available;
- monitoring and retirement criteria are defined.

High-risk uses additionally require evidence of validity, subgroup behavior, causal justification where relevant, and rollback capability.

D.11. Data inventory

The platform should maintain an inventory of:

- source systems;
- event types;
- entity types;
- personal and organizational identifiers;
- sensitive technical information;
- generated model outputs;
- derived metrics;
- access roles;
- retention periods;
- approved uses;
- downstream consumers.

Unknown or undocumented data should not enter decision-grade products.

D.12. Data classification

Recommended classifications include:

D.12.1. Public

Approved for external release.

D.12.2. Internal

General organizational use with low sensitivity.

D.12.3. Confidential

Business, organizational, or technical information requiring restricted access.

D.12.4. Restricted

Sensitive code, identity, security, incident, model, or individual-level evidence.

D.12.5. Highly restricted

Evidence whose exposure could create substantial security, privacy, employment, legal, or regulatory harm.

Classification should propagate to derived datasets unless de-identification or aggregation demonstrably reduces risk.

D.13. Sensitive data

Sensitive categories may include:

- individual identity and activity history;
- private repository metadata;
- security findings;
- incident details;
- access credentials or secrets;
- customer or regulated data;
- source-code content;
- model prompts and retrieved context;
- organizational restructuring information;
- employment-related records;
- protected or inferred personal characteristics.

Sensitive data requires explicit purpose, restricted access, retention limits, and enhanced audit.

D.14. Data minimization review

For each field, ask:

- Is it necessary for the approved purpose?
- Can it be replaced by a less detailed feature?
- Can it be aggregated earlier?

- Can identity be pseudonymized?
- Can content be replaced by metadata?
- Can the retention period be shortened?
- Can computation occur locally without centralizing raw evidence?

Fields without a defensible purpose should be removed.

D.15. Identity handling

Canonical identity is useful for lineage and temporal ownership, but it creates risk.

Controls should include:

- separate storage for identity mappings;
- pseudonymous analytical identifiers;
- role-based re-identification;
- time-bounded access;
- access logging;
- minimum aggregation thresholds;
- prohibition on unnecessary individual display;
- deletion or correction workflows.

Identity resolution confidence should be visible when evidence is used.

D.16. Aggregation policy

Aggregation can reduce risk but does not guarantee anonymity.

Policy should define:

- minimum group size;
- treatment of small teams;
- suppression of sparse dimensions;
- prevention of repeated-query reconstruction;
- time-window minimums;
- handling of unique roles;
- cross-filter restrictions.

Small-group suppression should consider whether values can be inferred from totals.

D.17. Re-identification risk

A dataset may be nominally anonymous but easily re-identifiable through repository, timestamps, language, team, or rare events.

Risk review should evaluate:

- uniqueness;

- linkability with other systems;
- rare-event exposure;
- repeated releases;
- access to auxiliary data;
- insider knowledge.

De-identification claims should be tested rather than assumed.

D.18. Access-control model

Access should be granted by role, purpose, data class, and time.

Recommended access layers include:

D.18.1. Raw source evidence

Limited to connector operators, data stewards, investigators, and approved researchers.

D.18.2. Canonical row-level evidence

Limited to platform, metric, and domain owners with a registered purpose.

D.18.3. Restricted analytical datasets

Available to approved analysts under purpose-bound access.

D.18.4. Team aggregates

Available to relevant practitioners and leaders, subject to minimum group size.

D.18.5. Enterprise aggregates

Available to broader decision makers when interpretation and uncertainty accompany the metric.

Access should expire automatically and require periodic recertification.

D.19. Purpose-based access

A person’s organizational role alone should not grant access to every use.

Access requests should identify:

- registered use;
- required datasets;
- necessary dimensions;
- expected duration;
- output audience;
- export requirements;

- accountable sponsor.

Using data outside the approved purpose should be treated as a governance incident.

D.20. Least privilege

Users and services should receive only the data and operations required.

Examples include:

- aggregated metrics instead of raw events;
- repository-limited access instead of enterprise-wide access;
- read-only views instead of export privileges;
- pseudonymous IDs instead of names;
- short-lived credentials;
- separate access for security-sensitive fields.

D.21. Separation of duties

High-consequence systems should separate:

- metric definition from employment decision making;
- model development from approval;
- data administration from audit;
- experiment analysis from unilateral rollout authority;
- vendor evaluation from procurement approval;
- policy recommendation from production deployment.

Separation reduces conflicts of interest and hidden manipulation.

D.22. Data retention

Retention should be based on purpose and risk rather than indefinite convenience.

A retention schedule should distinguish:

- immutable technical audit evidence;
- individual-level activity detail;
- raw message or content data;
- model prompts and outputs;
- experiment assignments;
- derived metrics;
- decision records;
- security and compliance evidence.

Detailed sensitive data should generally have shorter retention than governed aggregates and decision records.

D.23. Deletion

Deletion workflows should support:

- legal or policy deletion;
- source-system deletion propagation;
- user correction requests;
- removal of expired data;
- vendor termination;
- experiment closure;
- decommissioned metrics.

Deletion should include replicas, caches, indexes, model-training sets, and exports where applicable.

Where complete deletion is impossible due to immutable audit requirements, policy should document the legal basis, access restrictions, and minimization approach.

D.24. Export controls

Exports create copies outside central controls.

Policy should define:

- who may export;
- approved formats;
- encryption requirements;
- destination restrictions;
- expiration;
- watermarking or tagging;
- onward-sharing prohibition;
- deletion confirmation;
- audit logging.

Highly restricted evidence should remain within controlled analytical environments where possible.

D.25. Vendor governance

Vendors processing Engineering Intelligence evidence should be assessed for:

- data-use rights;
- model-training restrictions;
- data residency;
- subprocessors;
- retention and deletion;
- incident notification;
- security controls;
- audit rights;

- model and service changes;
- output ownership;
- portability and exit.

Vendor effectiveness should be evaluated using organization-controlled evidence rather than vendor-reported usage alone.

D.26. AI model governance

Introducing an AI-assisted use into Engineering Intelligence changes the substrate of the decision, not just its interface. The model's training data, prompt, retrieval context, and evaluation history become part of the analytical claim; a silent upgrade rewrites that substrate without informing anyone who relied on the previous behavior. AI model governance therefore treats each configuration as a versioned artefact under the same registration and rollback disciplines applied to metrics and experiments.

AI-assisted Engineering Intelligence uses require registration of:

- provider;
- model and version;
- purpose;
- input fields;
- prompt and policy version;
- retrieved context;
- output use;
- human review requirements;
- known limitations;
- evaluation results;
- permitted populations;
- rollback and replacement plan.

A model upgrade is a treatment change and may require revalidation.

D.27. Generated explanations

AI-generated summaries and recommendations should be labeled as generated inference.

They should not be presented as direct observation.

Important outputs should link to:

- source evidence;
- model and configuration;
- uncertainty;
- generated timestamp;
- human review status;
- corrections.

Generated narratives should never silently replace underlying incident, review, or decision records.

D.28. Automated decisions

Automated or semi-automated actions should be classified by consequence and reversibility. The four tiers below are not arranged by sophistication of the automation but by how much of the decision can be reversed without harm. A well-tuned model operating at the wrong tier is more dangerous than a crude one operating at the right tier, because reviewers assume the safeguards match the actions.

D.28.1. Advisory

The system recommends; a human decides.

D.28.2. Confirm-before-action

The system prepares an action requiring explicit approval.

D.28.3. Bounded autonomous

The system acts within preapproved low-risk limits and records the action.

D.28.4. High-impact autonomous

The system executes consequential action without immediate approval.

High-impact autonomous use should require exceptional evidence, strict bounds, continuous monitoring, independent review, and rapid shutdown.

D.29. Automation control requirements

Automated policy should include:

- explicit objective;
- eligible scope;
- evidence threshold;
- model and metric versions;
- confidence requirement;
- action limits;
- human override;
- appeal;
- audit trail;
- guardrails;
- stop and rollback rules;
- expiration or review date.

No automated policy should depend on a metric in unavailable or degraded quality state.

D.30. Individual-level views

Individual-level evidence may be operationally necessary for debugging, review assignment, incident response, or personal self-reflection.

Controls should include:

- narrow declared purpose;
- access limited to the person, appropriate operator, or authorized investigator;
- prohibition on comparative ranking;
- visible data provenance;
- correction workflow;
- retention limits;
- audit of access.

An individual should generally be able to see the evidence attributed to them unless doing so would compromise a legitimate investigation or security control.

D.31. Practitioner notice

Practitioners should receive clear notice describing:

- source systems used;
- categories of data collected;
- approved purposes;
- prohibited uses;
- aggregation practices;
- retention;
- who may access detail;
- correction and appeal;
- contact for governance concerns.

Notice should be understandable without requiring interpretation of a long legal policy.

D.32. Consent

Employment settings complicate meaningful consent because participation may not be freely optional.

Organizations should not use nominal consent to justify disproportionate monitoring.

Where consent is appropriate, it should be specific, informed, revocable where feasible, and separated from mandatory operational collection.

For required collection, organizations should rely on necessity, proportionality, transparency, governance, and applicable legal authority rather than fictional voluntariness.

D.33. Correction rights

Affected people and teams should be able to request correction of:

- identity attribution;
- team ownership;
- Engineering Change linkage;
- source-event interpretation;
- metric population inclusion;
- model classification;
- generated explanation;
- decision-record factual errors.

The system should preserve correction provenance and recompute affected outputs where material.

D.34. Appeal process

High-consequence decisions should provide an appeal route.

An appeal should support:

- access to relevant evidence;
- explanation of metrics and models;
- opportunity to provide context;
- review by a person independent of the original decision;
- correction of data errors;
- documented resolution;
- protection from retaliation.

Appeal volume and outcomes should be monitored as governance signals.

D.35. Fairness and subgroup review

Uses affecting populations should be evaluated across relevant groups and contexts.

Relevant dimensions may include:

- geography;
- organization;
- role;
- tenure band;
- language and technology stack;
- repository maturity;
- accessibility needs;
- work type;
- shift or time-zone constraints.

Protected characteristics should not be inferred casually. Fairness analysis involving sensitive attributes requires explicit

legal and privacy review.

D.36. Distributional effects

An intervention may improve the average while shifting burden to a smaller group.

Governance review should consider:

- who receives benefit;
- who bears validation work;
- who experiences false positives;
- who loses autonomy;
- who is excluded from treatment;
- who carries residual risk;
- whether scarce experts become more overloaded.

Distributional outcomes should accompany aggregate results.

D.37. Metric-use review

Before a metric is used in a new context, verify:

- the purpose is compatible with its specification;
- the population is valid;
- the interpretation remains supported;
- quality gates are met;
- the use does not violate prohibited interpretations;
- aggregation and privacy controls are adequate;
- the metric version is current;
- behavioral effects are acceptable.

A governed metric is not universally reusable.

D.38. Experiment governance

Experiments should receive review proportional to risk.

Review should cover:

- intervention consequence;
- assignment fairness;
- informed notice where appropriate;
- exposure logging;
- guardrails;
- stop rules;
- privacy;

- subgroup risk;
- rollback;
- post-study data retention.

Randomization does not excuse unsafe treatment.

D.39. Research publication

External publication should review:

- confidentiality;
- intellectual property;
- re-identification;
- employee privacy;
- customer and regulated data;
- vendor restrictions;
- security exposure;
- reproducibility tradeoffs.

Organizations should share methods, schemas, synthetic data, and aggregate findings where raw evidence cannot be disclosed.

D.40. Monitoring approved uses

Approved uses should be monitored for:

- data-quality degradation;
- population drift;
- metric-version changes;
- access anomalies;
- export activity;
- use outside declared purpose;
- behavioral gaming;
- subgroup harm;
- model drift;
- decision reversal or appeal;
- lack of continued value.

Monitoring should be tied to an owner and review cadence.

D.41. Audit logging

Audit records should include:

- identity of user or service;
- purpose and registered use;

- datasets and fields accessed;
- query or action;
- timestamp;
- export destination;
- metric or model version;
- automated decision or override;
- approval and review status.

Audit logs themselves are sensitive and require protection.

D.42. Periodic access review

Access should be recertified periodically.

Review should remove:

- inactive users;
- expired projects;
- overly broad roles;
- access inherited from previous teams;
- unused export permission;
- service accounts without owners;
- vendor access after contract change.

D.43. Governance incident

A governance incident includes:

- unauthorized access;
- purpose violation;
- prohibited individual ranking;
- erroneous high-impact metric;
- undisclosed vendor reuse;
- re-identification;
- automated action outside bounds;
- missing audit evidence;
- failure to delete expired data;
- retaliation for correction or appeal;
- material subgroup harm.

D.44. Incident response

Response should include:

1. contain access or automated action;

2. preserve evidence;
3. identify affected data, people, decisions, and outputs;
4. notify appropriate owners and reviewers;
5. correct or suppress affected metrics and decisions;
6. satisfy legal or contractual notification duties;
7. review root causes and control failures;
8. remediate systems and policy;
9. communicate findings to affected populations where appropriate;
10. track follow-up to closure.

D.45. Metric and model rollback

The platform should support rapid rollback of:

- metric versions;
- model versions;
- prompts and policies;
- access grants;
- automated actions;
- dashboard publications;
- derived datasets;
- experiment treatments.

Rollback procedures should be tested before high-risk use.

D.46. Misuse detection

Potential signals include:

- repeated queries targeting one person;
- unusually small group filters;
- exports of individual-level data;
- access outside registered use;
- attempts to join restricted identity data;
- dashboards embedded in performance-review workflows;
- automated actions exceeding policy limits;
- metric use after deprecation.

Detection should be balanced against legitimate privacy for analysts and investigators.

D.47. Policy exception process

Exceptions should require:

- written justification;

- defined scope;
- accountable sponsor;
- legal, privacy, and security review;
- compensating controls;
- expiration date;
- monitoring;
- documented renewal or termination.

Permanent informal exceptions undermine the standard.

D.48. Governance review cadence

D.48.1. Weekly operational review

Access anomalies, data incidents, automated-action failures, and active guardrail breaches.

D.48.2. Monthly use review

New uses, changed populations, metric changes, access requests, and exceptions.

D.48.3. Quarterly governance review

Retention, privacy, subgroup effects, model performance, appeals, and policy retirement.

D.48.4. Annual program review

Overall effectiveness, prohibited-use compliance, trust, incidents, organizational value, and changes in law or technology.

D.49. Governance maturity levels

D.49.1. Level 0: implicit

Rules exist mainly as informal expectations.

D.49.2. Level 1: documented

Permitted purposes, access, and prohibited uses are written.

D.49.3. Level 2: operational

Use registration, review, access controls, retention, and correction are functioning.

D.49.4. Level 3: auditable

Lineage, purpose-based access, audit, incidents, and periodic recertification are measurable.

D.49.5. Level 4: adaptive

Governance responds continuously to metric, model, workflow, and population change.

D.49.6. Level 5: trustworthy autonomy

Bounded automated policy operates with strong evidence, monitoring, appeal, and rollback.

Maturity should be demonstrated through controls and outcomes, not policy documents alone.

D.50. Governance anti-patterns

D.50.1. Collect first, justify later

Data becomes permanent before purpose and risk are considered.

D.50.2. Access by seniority

Leadership role is treated as a universal need to inspect individual detail.

D.50.3. Aggregate means anonymous

Small-group and auxiliary-data re-identification are ignored.

D.50.4. Metric governance without use governance

A valid metric is reused for an invalid purpose.

D.50.5. Consent theater

Employees are asked to agree to monitoring they cannot realistically refuse.

D.50.6. AI explanation as due process

A generated rationale substitutes for access to evidence and independent review.

D.50.7. Permanent pilot

Experimental access and data collection continue after the study ends.

D.50.8. Policy without enforcement

Prohibited uses are documented but not detectable or sanctionable.

D.50.9. Governance as delay only

Review adds bureaucracy without improving evidence quality, safety, or accountability.

D.51. Standard implementation checklist

1. Establish accountable governance roles and a review board.
2. Classify uses by consequence and sensitivity.
3. Register purpose, population, data, metrics, owner, and expiration.
4. Publish permitted, restricted, and prohibited uses.
5. Maintain data inventory, classification, lineage, and retention schedules.
6. Apply purpose-based least-privilege access and separation of duties.
7. Minimize identity detail and enforce aggregation thresholds.
8. Provide practitioner notice, correction, and appeal.
9. Govern vendors, models, prompts, and automated policy versions.
10. Monitor access, exports, drift, misuse, subgroup effects, and appeals.
11. Maintain incident response and tested rollback procedures.
12. Recertify access and approved uses on a declared cadence.
13. Retire data, metrics, models, and uses that no longer justify their risk.
14. Judge governance by preserved trust and safe decision quality, not review volume.

Appendix E: Economic Evaluation and Business Case Standard

***This appendix defines a rigorous method for evaluating the economic value of engineering** tools, platforms, architecture investments, workflow changes, and AI-assisted development systems. It distinguishes accounting savings from released capacity, modeled value from realized value, and local efficiency from enterprise outcomes. The standard is designed to prevent inflated return-on-investment claims while still enabling organizations to make timely decisions under uncertainty.*

E.1. Purpose

Engineering investments are often justified using incomplete arithmetic.

Common claims include:

- engineers save several hours each week;
- the tool pays for itself if productivity improves by one percent;
- automation replaces a fixed amount of labor;
- faster delivery automatically produces revenue;
- avoided incidents equal historical incident cost;
- generated code volume represents productive output.

These claims may be directionally useful, but they are not decision-grade until assumptions, mechanisms, opportunity costs, uncertainty, and realization conditions are explicit.

This standard supports decisions such as:

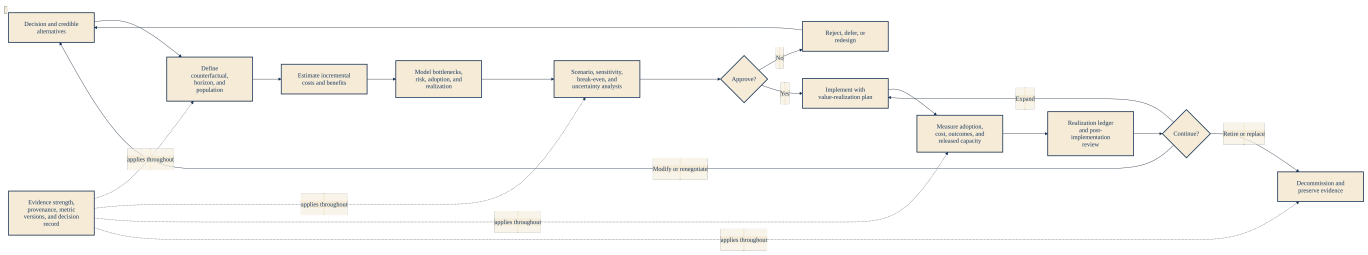
- purchase or renew a vendor tool;
- build an internal platform;
- expand an AI assistant;
- increase CI capacity;
- fund an architecture migration;
- change review policy;
- retire a legacy system;
- allocate investment across engineering portfolios.

E.2. Core economic model

The economic value of an intervention is the difference between outcomes under the intervention and outcomes under the credible alternative.

$$NetValue = IncrementalBenefits - IncrementalCosts$$

The comparison is not against zero. It is against the best realistic counterfactual, such as continuing the current tool, building internally, choosing another vendor, delaying investment, or stopping the activity.



E.3. Economic evaluation principles

E.3.1. Compare alternatives

Every business case should evaluate at least:

- proposed intervention;
- current state;
- credible alternative;
- defer or do-nothing option.

E.3.2. Count net effects

Savings in one area may create cost elsewhere. AI generation may reduce authoring effort while increasing validation, infrastructure, security, and maintenance costs.

E.3.3. Distinguish capacity from cash

Time released does not become financial savings unless the organization changes spending, avoids hiring, delivers additional valuable work, or reduces delay.

E.3.4. Value outcomes, not activity

More code, reviews, deployments, or suggestions are not automatically benefits.

E.3.5. Include uncertainty

Business cases should use ranges, scenarios, and sensitivity analysis rather than a single precise forecast.

E.3.6. Track realization

An approved business case is a hypothesis. Actual costs, adoption, capacity use, outcomes, and benefit realization must be measured after implementation.

E.3.7. Include distribution

Benefits and costs may fall on different teams, roles, business units, or time horizons.

E.4. Business-case identity

Each evaluation should contain:

- business-case identifier;
- intervention and version;
- decision owner;
- economic owner;
- study or evidence references;
- eligible population;
- comparison alternatives;
- evaluation horizon;
- currency and price basis;
- discount rate where relevant;
- approval and review dates.

Example:

```yaml
business_case_id: bc_ai_review_2026_02
intervention: enterprise_ai_code_review
version: 1.0
decision_owner: vp_engineering_platform
economic_owner: finance_business_partner
population: 8000_engineers_and_eligible_repositories
alternatives:
  - current_review_tool
  - internal_build
  - no_enterprise_standard
horizon_years: 3
currency: USD
price_basis_year: 2026
review_date: 2027-01-15
```

E.5. Define the counterfactual

The business case should describe what happens without the investment.

A credible counterfactual includes:

- current operating cost;
- expected price changes;
- current adoption and effectiveness;
- planned hiring or attrition;
- likely architecture and volume growth;
- known reliability or security risk;
- replacement needs;
- costs already committed;
- other interventions likely to occur.

Using a static current state as the counterfactual can overstate value when the organization would have improved anyway.

E.6. Cost taxonomy

E.6.1. Acquisition cost

Includes licenses, subscriptions, usage charges, committed spend, minimums, support tiers, and taxes.

E.6.2. Implementation cost

Includes integration, security review, procurement, migration, configuration, testing, enablement, and rollout labor.

E.6.3. Operating cost

Includes infrastructure, inference, storage, networking, observability, support, administration, model evaluation, and governance.

E.6.4. Human attention cost

Includes training, validation, review, prompt or policy maintenance, troubleshooting, workflow interruption, and exception handling.

E.6.5. Switching cost

Includes contract exit, data migration, retraining, workflow changes, decommissioning, temporary duplication, and productivity loss during transition.

E.6.6. Risk cost

Includes expected defects, incidents, security exposure, regulatory failure, vendor dependency, data leakage, and operational concentration.

E.6.7. Opportunity cost

Includes valuable work displaced by implementation, migration, or ongoing administration.

E.6.8. Residual and terminal cost

Includes decommissioning, archival, contract termination, retained infrastructure, and knowledge-transfer obligations at the end of the horizon.

E.7. Benefit taxonomy

E.7.1. Direct cash savings

Examples include:

- lower vendor spend;
- reduced infrastructure bills;
- contract consolidation;

- lower support or outsourcing cost;
- eliminated duplicated systems.

These are the strongest benefits when they produce an observable budget change.

E.7.2. Avoided cost

Examples include:

- hiring avoided because capacity meets growth;
- future infrastructure expansion avoided;
- renewal price increase avoided;
- migration or compliance cost avoided.

Avoided cost requires evidence that the expenditure would otherwise have occurred.

E.7.3. Released capacity

Time or attention becomes available for other work.

Released capacity is valuable, but it is not identical to cash savings.

E.7.4. Throughput value

The organization delivers additional valuable outcomes with the same resources.

This requires evidence that released capacity reaches constrained, valuable work and that downstream systems can absorb it.

E.7.5. Cost-of-delay reduction

Earlier delivery may produce value through earlier revenue, risk reduction, customer retention, regulatory compliance, or learning.

E.7.6. Quality and reliability value

Includes fewer defects, incidents, rollbacks, support cases, security failures, and emergency interruptions.

E.7.7. Risk reduction

Represents lower probability or consequence of adverse outcomes.

E.7.8. Option value

An investment may create future choices, such as easier vendor switching, faster experimentation, reusable platform capability, or access to new markets.

E.7.9. Capability value

Includes improved expertise, operational resilience, architecture understanding, and ability to act without external

dependency.

Capability value should be described carefully and not converted to cash without a defensible model.

E.8. Labor cost model

A fully loaded labor rate may include:

- salary;
- bonus;
- benefits;
- payroll taxes;
- facilities and equipment;
- management and shared services;
- contractor premiums.

However, multiplying minutes saved by a loaded hourly rate usually estimates capacity value, not cash savings.

$$ReleasedCapacityValue = HoursReleased * LoadedHourlyRate * RealizationFactor$$

The realization factor reflects how much released time becomes valuable productive capacity.

E.9. Capacity realization

Released time creates value only when several conditions hold:

1. the time is truly released rather than shifted;
2. the time appears in usable blocks;
3. engineers recognize and can redirect it;
4. there is valuable queued work;
5. downstream review, testing, product, and release systems can absorb it;
6. incentives do not cause the time to disappear into additional low-value activity.

A realization factor should be explicit and sensitivity-tested.

Example scenarios:

- conservative: 20 percent realized;
- expected: 45 percent realized;
- optimistic: 70 percent realized.

These are assumptions, not universal defaults.

E.10. Fragmentation discount

Five minutes saved across many tasks may be less useful than one uninterrupted hour.

A fragmentation discount can reflect whether released capacity is actionable.

$$UsableCapacity = GrossHoursReleased * ContinuityFactor$$

The continuity factor may depend on task batching, workflow design, interruption, and local control over scheduling.

E.11. Bottleneck adjustment

Improving a non-bottleneck may not increase system throughput.

For example, faster code generation may produce no delivery benefit when specialist review or test environments remain constrained.

Economic models should identify:

- current bottleneck;
- expected bottleneck after intervention;
- capacity of adjacent stages;
- queue and utilization effects;
- possible congestion created by higher upstream output.

E.12. Queueing effects

Small reductions in service time can produce disproportionate waiting-time benefits near high utilization.

Conversely, a modest increase in work arrival rate can destabilize a queue.

Business cases should consider:

- arrival rate;
- service rate;
- variability;
- utilization;
- queue discipline;
- shared specialists;
- peak periods.

This is especially important for review, CI, test environments, security approval, and release systems.

E.13. Cost of delay

Cost of delay estimates the value lost while an outcome is unavailable.

Possible components include:

- deferred revenue;
- delayed customer learning;
- continued operational cost;
- extended security or compliance exposure;
- customer dissatisfaction;
- competitive disadvantage;
- delayed retirement of legacy systems.

The model must state whether earlier technical completion actually leads to earlier release and benefit.

E.14. Quality economics

Quality value should include frequency and consequence.

$$ExpectedLoss = sum_i Probability_i * Consequence_i$$

Consequences may include:

- engineering response effort;
- customer support;
- service credits;
- revenue loss;
- data remediation;
- regulatory penalties;
- reputation damage;
- future prevention work;
- interruption of planned work.

Historical incident averages may be misleading when severity has a heavy-tailed distribution.

E.15. Risk-adjusted value

For uncertain outcomes, use expected values and scenarios.

Example:

```yaml
risk_reduction:
  event: severe_change_related_incident
  baseline_annual_probability: 0.12
  treated_annual_probability: 0.08
  consequence_range_usd:
    low: 500000
    expected: 2500000
    high: 15000000
```

Rare catastrophic outcomes should also be considered through risk tolerance, not expected value alone.

E.16. AI tool economics

AI systems require a complete net-value model.

Costs may include:

- per-seat license;
- inference and token usage;
- premium models;
- retrieval and indexing;

- human validation;
- false-positive handling;
- generated-code maintenance;
- security review;
- model evaluation;
- vendor integration;
- policy and governance;
- incident risk;
- fallback capability.

Benefits may include:

- reduced routine authoring effort;
- earlier defect detection;
- improved test coverage;
- faster information retrieval;
- reduced review burden;
- reduced queue delay;
- increased accessibility to unfamiliar systems;
- faster experimentation.

Usage or acceptance rate alone is insufficient for valuation.

E.17. Validation burden

AI-produced output creates validation work.

$$NetAttentionChange = AuthoringAttentionReduced - ValidationAttentionAdded - ReworkAttentionAdded$$

Validation burden should include:

- reviewing suggestions;
- checking correctness;
- resolving false positives;
- understanding generated code;
- investigating subtle failures;
- correcting style or architecture drift;
- security and compliance checks.

A tool can have high adoption and negative net attention value.

E.18. Vendor pricing models

Compare pricing under realistic scale and usage.

Common models include:

- per seat;

- per active user;
- per request or review;
- token or compute usage;
- repository tier;
- enterprise flat fee;
- committed capacity;
- outcome-based pricing.

The evaluation should model:

- growth;
- unused seats;
- usage concentration;
- price escalation;
- minimum commitments;
- overages;
- premium-feature dependency;
- negotiation and renewal risk.

E.19. Build versus buy

A build-versus-buy analysis should compare:

- time to usable capability;
- product and engineering labor;
- infrastructure;
- support burden;
- model and feature evolution;
- security and compliance;
- integration depth;
- customization;
- vendor concentration;
- switching cost;
- opportunity cost;
- expected lifetime.

Internal build cost should include ongoing product ownership, not only initial engineering effort.

E.20. Architecture investment economics

Architecture investments often produce indirect, delayed, and distributed value.

Potential benefits include:

- lower coordination cost;
- reduced change blast radius;

- fewer incidents;
- easier testing;
- faster onboarding;
- independent deployment;
- reduced vendor dependency;
- easier regulatory separation;
- greater option value.

The evaluation should link architectural mechanisms to observable delivery, quality, and operating outcomes.

E.21. Platform economics

A platform creates shared capability and may exhibit scale economies.

Model:

- fixed platform cost;
- marginal cost per tenant or workload;
- adoption curve;
- avoided duplicated investment;
- local integration cost;
- support and reliability obligations;
- migration cost;
- value concentration among heavy users;
- cost of forced standardization.

A platform is not valuable merely because many teams use it.

E.22. Allocation of shared costs

Shared costs should be transparent even when not charged back.

Possible allocation methods include:

- equal allocation;
- usage-based allocation;
- active-user allocation;
- transaction allocation;
- capacity allocation;
- business-value allocation;
- central strategic funding.

The chosen method can change behavior and should not be mistaken for the underlying economic cost.

E.23. Sunk costs

Past spending that cannot be recovered should not determine the forward decision.

However, switching costs, remaining contract obligations, migration effort, and residual value are relevant.

The analysis should distinguish sunk cost from avoidable future cost.

E.24. Evaluation horizon

Choose a horizon appropriate to the intervention.

Examples:

- workflow configuration: 6 to 18 months;
- vendor platform: 2 to 4 years;
- architecture migration: 3 to 7 years;
- foundational platform: longer, with staged decision gates.

Long horizons increase uncertainty and require scenario analysis.

E.25. Discounting

Future costs and benefits may be discounted.

$$NPV = sum_(t = 0)^T (Benefit_t - Cost_t)/(1 + r)^t$$

Where r is the discount rate and t is the time period.

The business case should disclose the rate and whether values are nominal or real.

E.26. Standard financial outputs

Useful outputs include:

- net present value;
- internal rate of return where meaningful;
- payback period;
- annualized cost;
- cost per eligible Engineering Change;
- cost per active user;
- cost per accepted outcome;
- expected value;
- break-even adoption;
- break-even effect size.

No one output should replace the underlying assumptions.

E.27. Break-even analysis

Break-even analysis is often more useful than a single forecast.

Examples:

- How many review minutes must be saved per change to cover the vendor cost?
- What adoption level is required?
- How much incident probability reduction is required?
- How many hires must be avoided?
- What inference price makes the tool uneconomic?

$$BreakEvenMinutesPerChange = AnnualIncrementalCost / (EligibleChanges * LoadedCostPerMinute * RealizationFactor)$$

E.28. Scenario analysis

At minimum, provide conservative, expected, and optimistic scenarios.

Scenarios should vary multiple coherent assumptions rather than changing only one number.

Example dimensions include:

- adoption;
- effect size;
- realization;
- price;
- eligible volume;
- quality effect;
- migration duration;
- staffing growth;
- vendor discount.

E.29. Sensitivity analysis

Identify which assumptions most affect the decision.

Common high-sensitivity variables include:

- human time saved;
- realization factor;
- vendor price;
- inference usage;
- outcome attribution;
- incident consequence;
- adoption;
- time to rollout;
- useful life.

A business case dominated by one weak assumption should be treated cautiously.

E.30. Monte Carlo analysis

For large or uncertain investments, model assumptions as distributions rather than fixed values.

Outputs may include:

- probability of positive NPV;
- expected NPV;
- downside percentile;
- probability of payback within horizon;
- major value drivers.

Distribution choices and correlations must be documented.

E.31. Evidence strength

Each economic input should be labeled by evidence strength.

Suggested levels:

- observed accounting fact;
- measured operational fact;
- randomized causal estimate;
- credible quasi-experimental estimate;
- calibrated model estimate;
- expert assumption;
- vendor claim;
- placeholder.

A business case should not present all inputs as equally certain.

E.32. Attribution

Benefits should be attributed only to the incremental effect of the intervention.

Avoid double-counting when several programs influence the same outcome.

Possible methods include:

- randomized experiment;
- phased rollout;
- difference-in-differences;
- contribution model;
- explicit shared attribution;
- conservative residual attribution.

E.33. Double-counting controls

Common double-counting includes:

- counting time savings and the same increased throughput separately;
- counting avoided hiring and labor savings for the same capacity;
- counting fewer incidents and recovered engineering time separately without adjustment;
- counting revenue acceleration and total revenue;
- assigning the full outcome to multiple tools.

The business case should include a benefit-dependency map.

E.34. Capacity, savings, and value labels

Every benefit should use one of these labels:

E.34.1. Realized cash impact

An observed budget, spend, or revenue change.

E.34.2. Contracted or committed impact

A signed or formally approved future financial change.

E.34.3. Avoided planned cost

A documented expenditure no longer required.

E.34.4. Released capacity

Measured time or attention available for other work.

E.34.5. Modeled outcome value

Estimated value based on causal or predictive assumptions.

E.34.6. Strategic or option value

A qualitative or scenario-based future capability.

These labels prevent modeled capacity from being reported as booked savings.

E.35. Value-realization plan

Before approval, define how benefits will become real.

For released capacity, specify:

- which bottleneck is relieved;

- which work will absorb capacity;
- who owns reallocation;
- how utilization will be observed;
- what downstream constraints must change;
- when realization will be reviewed.

For cash savings, specify:

- contract action;
- budget owner;
- timing;
- accounting treatment;
- dependency on migration or decommissioning.

E.36. Realization ledger

Track forecast and actual value over time.

Example:

```yaml
benefit_id: benefit_review_capacity
business_case_id: bc_ai_review_2026_02
benefit_type: released_capacity
forecast_annual_value_usd: 4200000
forecast_hours: 28,000
realization_factor: 0.45
actual_to_date:
  measured_hours_released: 9100
  verified_reallocated_hours: 3400
  booked_cash_savings_usd: 0
  avoided_hiring_usd: 650000
confidence: medium
owner: developer_productivity
review_date: 2027-01-15
```

E.37. Post-implementation review

Review actual performance against the case.

Measure:

- implementation cost;
- operating cost;
- adoption and eligible coverage;
- treatment fidelity;
- measured effect;
- validation burden;
- quality outcomes;
- capacity realization;

- cash or avoided cost;
- unanticipated costs;
- vendor and switching risk;
- continued strategic relevance.

The review should support expand, modify, renegotiate, replace, or retire decisions.

E.38. Renewal economics

Renewal should not rely on the original purchase case.

Reevaluate:

- current alternatives;
- actual usage;
- causal or outcome evidence;
- realized value;
- price changes;
- switching cost;
- market evolution;
- model or product changes;
- concentration risk;
- contract flexibility.

A successful initial rollout does not imply an indefinitely favorable renewal.

E.39. Portfolio allocation

Compare investments using more than nominal ROI.

Portfolio decisions should consider:

- expected value;
- downside risk;
- strategic alignment;
- time to value;
- reversibility;
- evidence strength;
- dependency on scarce expertise;
- option value;
- correlation with other investments;
- organizational capacity to execute.

A smaller high-confidence investment may be preferable to a larger speculative return.

E.40. Decision thresholds

Possible economic decision rules include:

- positive expected NPV with downside limit;
- payback within a specified horizon;
- probability of positive value above threshold;
- cost per outcome below an alternative;
- noninferior quality at lower cost;
- strategic capability under a maximum affordable loss.

Thresholds should reflect organizational risk tolerance and reversibility.

E.41. Decision record

Example:

```yaml
decision_id: dec_bc_2026_119
business_case_id: bc_ai_review_2026_02
decision: approve_bounded_rollout
financial_summary:
  expected_three_year_npv_usd: 6800000
  conservative_npv_usd: 900000
  optimistic_npv_usd: 14200000
  probability_positive_npv: 0.82
key_assumptions:
  - 45_percent_capacity_realization
  - 70_percent_eligible_change_coverage
  - quality_noninferiority
conditions:
  - annual_vendor_cost_below_2500000
  - severe_defect_guardrail_maintained
  - six_month_realization_review
  - portability_plan_completed
owner: vp_engineering_platform
review_date: 2027-01-15
```

E.42. Business-case maturity levels

E.42.1. Level 0: narrative

Benefits and costs are described qualitatively.

E.42.2. Level 1: estimated

Major cost and benefit categories are quantified with visible assumptions.

E.42.3. Level 2: evidence-linked

Inputs are connected to operational data, studies, contracts, and metric versions.

E.42.4. Level 3: decision-grade

Alternatives, uncertainty, scenarios, break-even, risks, and realization are complete.

E.42.5. Level 4: realized

Actual costs and benefits are tracked after implementation.

E.42.6. Level 5: adaptive portfolio evidence

Realization history improves future forecasts and portfolio allocation.

E.43. Anti-patterns

E.43.1. Salary multiplication as savings

Minutes saved are multiplied by salary and reported as cash savings without a realization mechanism.

E.43.2. Vendor usage as value

High invocation or acceptance is treated as proof of economic benefit.

E.43.3. Gross benefit without validation cost

Automation savings omit the human work required to trust the output.

E.43.4. Static counterfactual

The current state is assumed never to change.

E.43.5. Best-case-only forecast

Adoption, effect size, and price all use optimistic assumptions.

E.43.6. Double-counted benefit

Time savings, throughput, avoided hiring, and revenue are all claimed from the same effect.

E.43.7. Ignored bottleneck

Local efficiency is assumed to improve end-to-end outcomes.

E.43.8. Ignored migration and exit

Implementation and switching costs are omitted.

E.43.9. Modeled value reported as realized savings

Forecast capacity is presented as booked financial impact.

E.43.10. No post-implementation review

The original business case is never tested against reality.

E.44. Economic evaluation checklist

1. Define the intervention, decision, horizon, owner, and credible alternatives.
2. Model the changing counterfactual rather than a frozen current state.
3. Include acquisition, implementation, operating, human, switching, risk, and opportunity costs.
4. Separate cash savings, avoided cost, released capacity, outcome value, and option value.
5. Measure validation burden and downstream work created by automation.
6. Identify system bottlenecks and queueing effects.
7. State realization and fragmentation assumptions for released capacity.
8. Link quality and risk value to probability and consequence.
9. Use scenarios, sensitivity, break-even, and uncertainty analysis.
10. Label every major input by evidence strength.
11. Prevent double-counting with a benefit-dependency map.
12. Define a concrete value-realization plan before approval.
13. Track forecast versus actual value in a realization ledger.
14. Reevaluate economics at renewal, expansion, and material treatment change.
15. Report modeled value honestly and reserve “savings” for realized or committed financial impact.

Appendix F: Technical Reference Architecture

***This appendix defines a deployable technical reference architecture for an Engineering** Intelligence platform. It describes the responsibilities, contracts, data flows, storage patterns, reliability controls, security boundaries, and staged implementation needed to connect engineering-system evidence to trustworthy metrics, experiments, decisions, and organizational learning. The architecture is intentionally vendor-neutral. Its durable value lies in semantic contracts, provenance, temporal correctness, and governed use rather than any particular database or cloud service.*

F.1. Architectural objective

The platform should answer consequential questions such as:

- What happened across the lifecycle of an Engineering Change?
- Where was human attention consumed?
- Which queues and architectural constraints limited flow?
- Did an AI-assisted workflow improve net outcomes?
- Which intervention caused an observed change?
- What economic value was realized?
- Can the result be reproduced from source evidence?
- Is the data permitted for this decision and audience?

The platform must answer these questions without reducing engineering to artifact counts or creating an unrestricted surveillance system.

F.2. Architectural principles

F.2.1. Immutable evidence first

Preserve source events before applying interpretation. Derived state can be recomputed; lost source evidence usually cannot.

F.2.2. Canonical semantics

Normalize vendor-specific objects into stable entities and events while retaining native identifiers and payload lineage.

F.2.3. Event time over ingestion time

Analysis should reflect when an event occurred. Ingestion time remains essential for freshness and operational diagnosis.

F.2.4. Temporal identity and ownership

Resolve people, agents, teams, repositories, services, and ownership as they existed at event time.

F.2.5. Confidence-aware linkage

Inferred Engineering Change relationships must expose method and confidence.

F.2.6. Version everything consequential

Schemas, transformations, identity rules, reconstruction logic, metrics, models, prompts, experiments, and policies require versions.

F.2.7. Separate evidence, inference, and decision

Observed facts, reconstructed relationships, predictions, causal estimates, generated narratives, and organizational decisions should remain distinguishable.

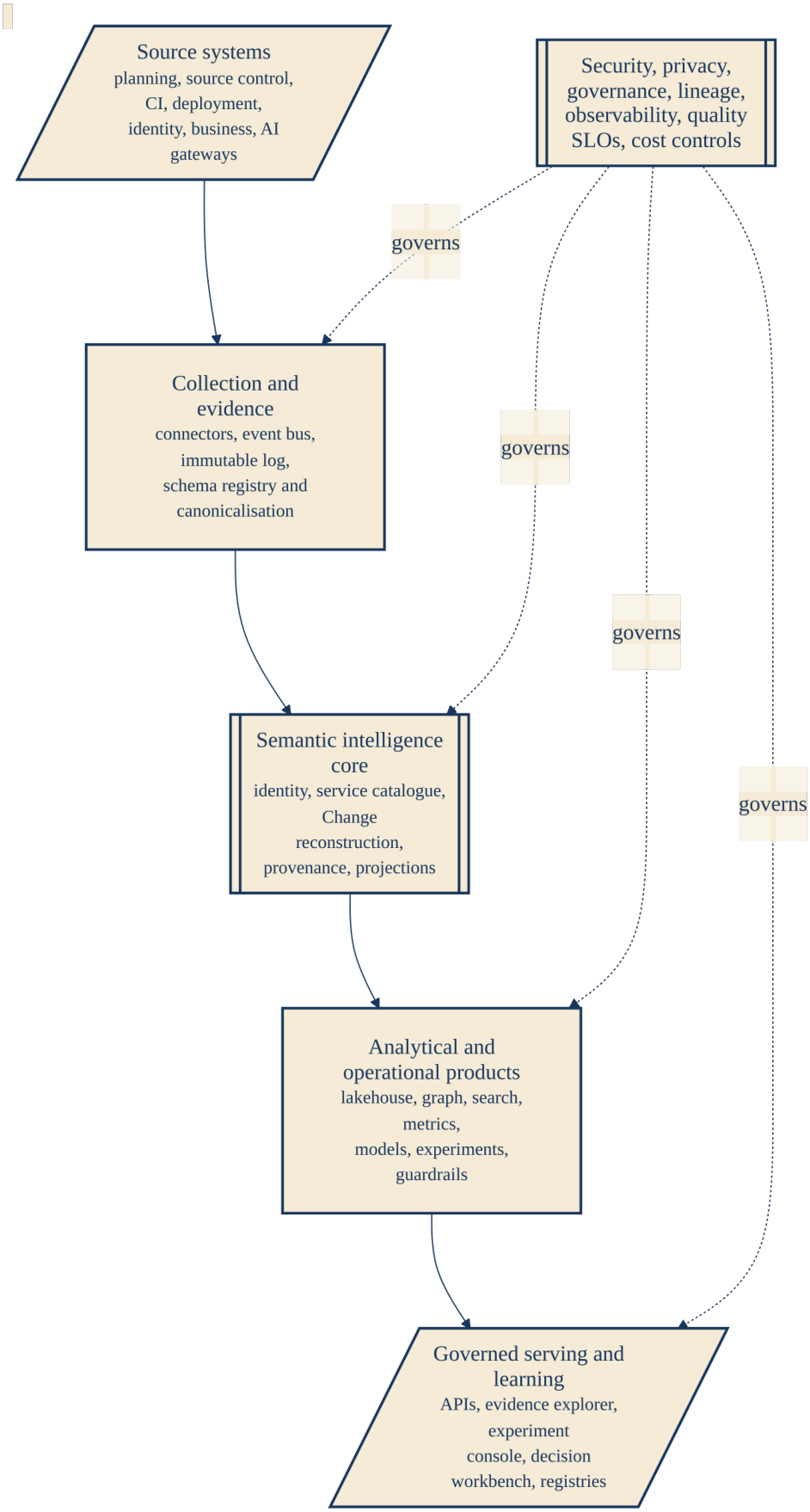
F.2.8. Purpose-bound access

Authorization should consider approved use and data sensitivity, not role alone.

F.2.9. Progressive capability

The platform should create value from an initial narrow evidence path and grow without requiring a complete enterprise ontology on day one.

F.3. Logical architecture

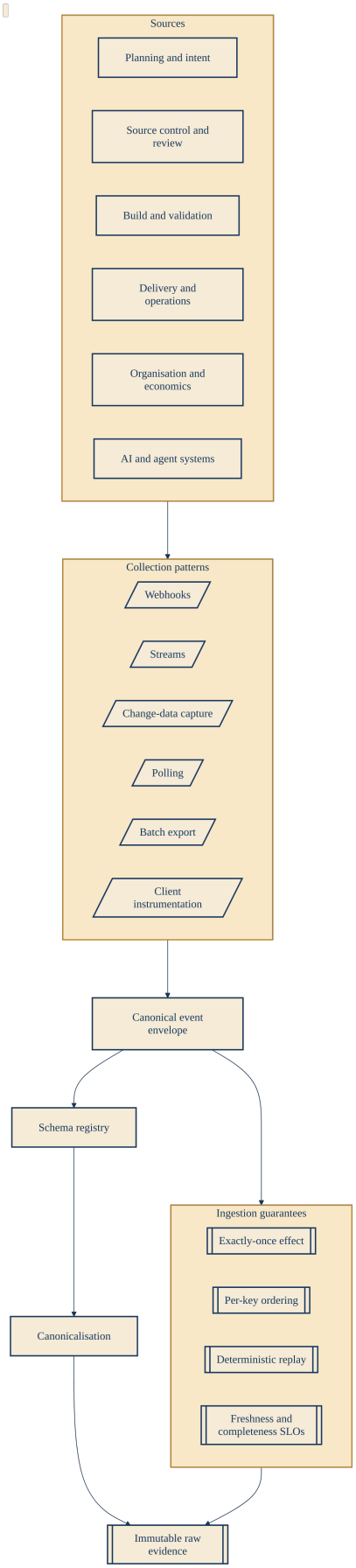

Source systems
planning, source control, CI, deployment, identity, business, AI gateways
Security, privacy, governance, lineage, observability, quality SLOs, cost controls
governs
Collection and evidence
connectors, event bus, immutable log, schema registry and canonicalisation
governs
governs
Semantic intelligence core
identity, service catalogue, Change reconstruction, provenance, projections
governs
Analytical and operational products
lakehouse, graph, search, metrics, models, experiments, guardrails
Governed serving and learning
APIs, evidence explorer, experiment console, decision workbench, registries

The logical architecture contains twelve major capability areas:

1. source systems;
2. collection and ingestion;
3. immutable raw evidence;
4. canonicalization and schema management;
5. identity, ownership, and reference data;
6. Engineering Change reconstruction;
7. analytical and graph projections;
8. metric and model computation;
9. experimentation and treatment exposure;
10. governed serving and decision products;
11. evidence, metric, model, and decision registries;
12. platform observability, security, privacy, and governance.

F.4. Source systems

The collection layer connects heterogeneous source systems to a single canonical event stream that everything downstream will read from. The following diagram shows how the source domains fan into a small set of collection patterns, land in immutable raw evidence under explicit ingestion guarantees, and are canonicalized against a shared schema before any semantic work begins.


Sources
Planning and intent
Source control and review
Build and validation
Delivery and operations
Organisation and economics
AI and agent systems
Collection patterns
Webhooks
Streams
Change-data capture
Polling
Batch export
Client instrumentation
Canonical event envelope
Schema registry
Canonicalisation
Ingestion guarantees
Exactly-once effect
Per-key ordering
Deterministic replay
Freshness and completeness SLOs
Immutable raw evidence

Typical source domains include:

F.4.1. Planning and intent

- product planning;
- issue tracking;
- request management;
- architecture decisions;
- design documents;
- experiment proposals.

F.4.2. Source control and review

- repositories;
- commits;
- branches;
- pull or merge requests;
- review comments and approvals;
- code ownership;
- AI review findings.

F.4.3. Build and validation

- CI pipelines;
- test systems;
- static analysis;
- security scanners;
- performance testing;
- simulation;
- artifact repositories.

F.4.4. Delivery and operations

- deployment systems;
- release and feature-flag platforms;
- service catalogues;
- runtime observability;
- incident systems;
- rollback and recovery systems.

F.4.5. Organization and economics

- directory and identity systems;
- team and ownership records;

- finance and procurement;
- cloud and vendor cost;
- portfolio and business-outcome systems.

F.4.6. AI and agent systems

- assistants;
- agent runtimes;
- model gateways;
- prompt registries;
- tool invocation logs;
- evaluation systems;
- policy engines.

Not every source belongs in the first implementation. Sources should be prioritized by the decisions they enable.

F.5. Collection patterns

F.5.1. Webhooks

Use webhooks for low-latency event delivery when the source supports reliable signatures, retries, and stable identifiers.

F.5.2. Event streams

Use native streams or logs when high-volume ordered evidence is available.

F.5.3. Change-data capture

CDC may be appropriate for internally owned relational systems, but database changes still require semantic interpretation.

F.5.4. Polling

Polling is acceptable when APIs expose modification timestamps, pagination, and stable cursors. The collector must tolerate repeated records and eventual consistency.

F.5.5. Batch export

Large historical backfills, finance data, organizational snapshots, and vendor exports may arrive as files.

F.5.6. Client-side instrumentation

Use sparingly. Client telemetry can measure otherwise invisible interactions, but increases privacy, compatibility, and data-quality risk.

F.6. Connector contract

Every connector should provide:

- source-system identity;
- source instance or tenant;
- authentication mode;
- supported entity and event types;
- cursor or checkpoint behavior;
- idempotency strategy;
- ordering guarantees;
- deletion behavior;
- rate-limit handling;
- backfill procedure;
- expected freshness;
- schema mapping version;
- data classification;
- operational owner.

A connector should not embed business metrics. Its responsibility is faithful collection and source-specific normalization.

F.7. Canonical event envelope

A canonical envelope should include:

```
{
  "event_id": "evt_01J...",
  "event_type": "review.finding.dispositioned",
  "event_version": "1.2.0",
  "occurred_at": "2026-07-18T12:44:08.217Z",
  "observed_at": "2026-07-18T12:44:09.004Z",
  "ingested_at": "2026-07-18T12:44:10.221Z",
  "source": {
    "system": "code_review",
    "instance": "enterprise-main",
    "native_event_id": "884391"
  },
  "actor_ref": "actor_01J...",
  "subject": {
    "entity_type": "finding",
    "entity_id": "finding_9201"
  },
  "engineering_change_ref": "ec_01J...",
  "payload": {},
  "provenance": {
    "connector_version": "2.4.0",
    "mapping_version": "1.8.2",
    "schema_registry_id": "review.finding.dispositioned@1.2.0"
  },
  "classification": "restricted"
}
```

Unknown Engineering Change references may be null at ingestion and added later through a versioned relationship event.

F.8. Ingestion guarantees

The platform should assume at-least-once delivery unless stronger guarantees are demonstrably available.

Required controls include:

- stable event identifiers;
- idempotent writes;
- source cursor checkpoints;
- duplicate detection;
- dead-letter handling;
- replay;
- backpressure;
- late-event support;
- source-specific ordering keys;
- poison-record quarantine.

Exactly-once marketing language should not replace explicit idempotency and replay design.

F.9. Immutable raw evidence layer

Raw evidence should preserve the received source representation with minimal transformation.

Recommended partition dimensions include:

- source system;
- source instance;
- event or entity type;
- ingestion date;
- region or classification where required.

Raw evidence should include checksums, encryption, retention class, and ingestion metadata.

Access should be highly restricted because raw payloads may contain source code, messages, identities, security findings, or customer information.

F.10. Schema registry

The schema registry should manage:

- canonical event schemas;
- entity schemas;
- compatibility rules;
- ownership;
- classification;

- field descriptions;
- required provenance;
- deprecation;
- consumers;
- sample fixtures.

Schema changes should be tested against historical and current events before promotion.

F.11. Canonicalization

Canonicalization translates source objects into stable engineering semantics.

Examples include:

- GitHub pull request and GitLab merge request to `CodeReviewChange`;
- Jenkins job and GitHub Actions run to `ValidationRun` or `Build`;
- Kubernetes rollout and proprietary release event to `Deployment`;
- vendor AI comment to `Finding` with machine provenance.

Canonicalization should preserve source-specific extensions rather than forcing all detail into the common schema.

F.12. Data contracts

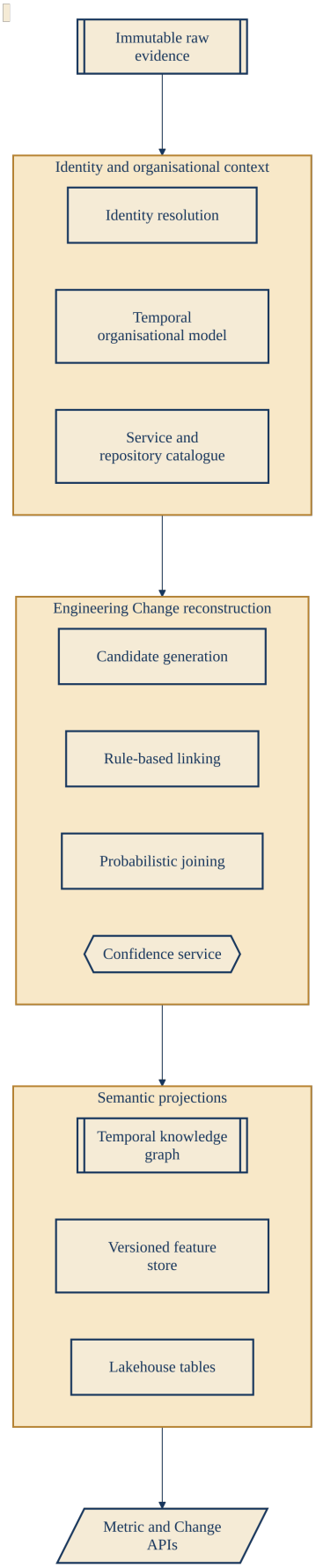
A producing domain should publish a data contract specifying:

- entity and event semantics;
- identifiers;
- event-time behavior;
- completeness expectations;
- known exclusions;
- deletion semantics;
- compatibility policy;
- data owner;
- quality SLOs;
- change-notification process.

Contracts turn hidden assumptions into operational dependencies.

F.13. Identity resolution service

The semantic core is the layer at which raw evidence stops being tool-shaped and starts being Engineering-Change-shaped. It resolves who did the work, in which organizational context, and how the resulting activity connects into a coherent Change. Its projections are what downstream metrics, models, and experiments actually read.


Immutable raw evidence
Identity and organisational context
Identity resolution
Temporal organisational model
Service and repository catalogue
Engineering Change reconstruction
Candidate generation
Rule-based linking
Probabilistic joining
Confidence service
Semantic projections
Temporal knowledge graph
Versioned feature store
Lakehouse tables
Metric and Change APIs

The identity service maps source identities to canonical actors.

It should support:

- people with multiple accounts;
- service accounts;
- deterministic automation;
- AI assistants and autonomous agents;
- vendors and external contributors;
- identity changes over time;
- uncertain matches;
- manual correction;
- restricted re-identification.

Identity mappings should be separated from general analytical storage and accessed through controlled interfaces.

F.14. Temporal organizational model

The organizational model represents:

- team membership;
- reporting relationships where approved;
- repository and service ownership;
- domain ownership;
- on-call or specialist roles;
- organization changes;
- effective intervals;
- source and confidence.

Every ownership join should specify an as-of time.

F.15. Service and repository catalogue

The catalogue should link:

- repositories;
- services;
- libraries;
- data products;
- infrastructure components;
- owners;
- criticality;
- runtime environments;
- dependencies;
- compliance and data classes.

This catalogue provides essential context for blast radius, portfolio analysis, incident linkage, and risk-based policy.

F.16. Engineering Change reconstruction

The reconstruction service groups intents, artifacts, activities, validations, deployments, releases, and outcomes into Engineering Changes.

It should use a hierarchy of evidence:

1. explicit propagated Engineering Change identifier;
2. deterministic artifact lineage;
3. source-native relationships;
4. versioned rule-based inference;
5. probabilistic or model-based inference;
6. manual correction.

Each relationship should retain confidence, method, evidence references, and reconstruction version.

F.17. Reconstruction pipeline

A typical pipeline performs:

1. candidate generation;
2. deterministic linking;
3. temporal and ownership filtering;
4. artifact-graph traversal;
5. confidence scoring;
6. conflict resolution;
7. lifecycle derivation;
8. quality assessment;
9. publication of relationship events;
10. correction and recomputation.

Reconstruction must be replayable because improved rules may alter historical linkage.

F.18. Candidate generation

Candidate links may be generated from:

- explicit IDs;
- commit ancestry;
- build provenance;
- artifact digests;
- branch or issue references;
- deployment manifests;
- timestamps;
- shared actors;
- ownership;

- textual similarity;
- dependency paths.

Candidate generation should favor recall; later scoring and validation control precision.

F.19. Confidence service

Confidence should not be a single unexplained score.

The service should publish:

- score;
- band;
- evidence features;
- rule or model version;
- calibration date;
- known ambiguity;
- manual-review state.

Metrics should declare eligible confidence bands.

F.20. Lifecycle state projection

A state projection derives the semantic lifecycle from events.

It should retain:

- current state;
- transition history;
- transition event;
- derivation rule;
- time in state;
- uncertainty;
- reopened or reversed transitions.

Mutable state tables may serve queries, but transition events remain authoritative.

F.21. Analytical storage

A practical architecture may use several complementary stores.

F.21.1. Lakehouse or warehouse

Best suited for governed facts, dimensions, historical analysis, model training, and reproducible metric computation.

F.21.2. Graph projection

Best suited for dependency traversal, Engineering Change lineage, ownership networks, blast radius, and multi-hop investigation.

F.21.3. Search index

Best suited for finding evidence, decisions, findings, incidents, and metric documentation.

F.21.4. Time-series store

Best suited for freshness, operational queues, platform SLOs, and high-frequency runtime signals.

F.21.5. Key-value or operational database

Best suited for low-latency assignment, exposure, registry, and serving state.

The canonical semantic model should not depend exclusively on one physical representation.

F.22. Fact and dimension design

Useful governed facts include:

- activity fact;
- review session fact;
- finding fact;
- validation run fact;
- build fact;
- deployment fact;
- release exposure fact;
- incident-change fact;
- experiment assignment and exposure fact;
- cost fact;
- outcome fact.

Dimensions include actor, team, repository, service, domain, artifact, model, tool, experiment, and calendar.

Slowly changing dimensions should preserve historical context.

F.23. Metric computation layer

The metric layer executes versioned specifications from the metric registry.

It should support:

- batch and incremental computation;
- reproducible historical restatement;
- population filters;

- confidence thresholds;
- missing-data semantics;
- quality gates;
- uncertainty calculation;
- approved dimensions;
- privacy suppression;
- semantic versioning.

Metric code should be reviewed and tested like production software.

F.24. Metric registry

The registry stores:

- metric identity and version;
- construct and purpose;
- executable definition;
- source dependencies;
- quality requirements;
- interpretation;
- prohibited uses;
- approved dimensions;
- owner and steward;
- validation evidence;
- lifecycle status;
- dependent products.

A dashboard should resolve metrics through the registry rather than reimplementing formulas locally.

F.25. Feature and model layer

Predictive models may estimate:

- change risk;
- likely review expertise;
- expected queue delay;
- blast radius;
- defect probability;
- intervention eligibility;
- outcome maturity.

The model registry should contain:

- model identity and version;
- training population;
- feature versions;

- evaluation results;
- calibration;
- subgroup behavior;
- intended and prohibited uses;
- deployment state;
- owner;
- rollback target.

Predictions should retain model version and feature-time provenance.

F.26. Experiment assignment service

The assignment service should provide:

- deterministic assignment;
- unit eligibility;
- stratification;
- stable variants;
- override controls;
- assignment audit;
- assignment API;
- historical reconstruction.

Assignment should occur before treatment-dependent information can affect eligibility.

F.27. Exposure service

The exposure service records the treatment actually received.

It should distinguish:

- assigned;
- available;
- invoked;
- visible;
- completed;
- used before decision;
- bypassed;
- failed;
- contaminated control.

Treatment version and configuration hash are required.

F.28. Guardrail monitoring

Experiment and policy guardrails should run through a low-latency monitoring path where harm requires rapid response.

The path should include:

- event ingestion;
- eligibility and treatment context;
- current guardrail computation;
- quality status;
- threshold logic;
- alerting;
- pause or rollback integration;
- human escalation.

Operational stop rules should not depend solely on a slow analytical warehouse refresh.

F.29. Decision registry

The decision registry connects evidence to action.

It should store:

- decision question;
- alternatives;
- studies and metrics;
- versions;
- economic analysis;
- assumptions;
- uncertainty;
- chosen action;
- owner;
- conditions;
- review date;
- reversal criteria;
- later outcome.

This prevents evidence from disappearing into presentation files.

F.30. Evidence registry

The evidence registry indexes:

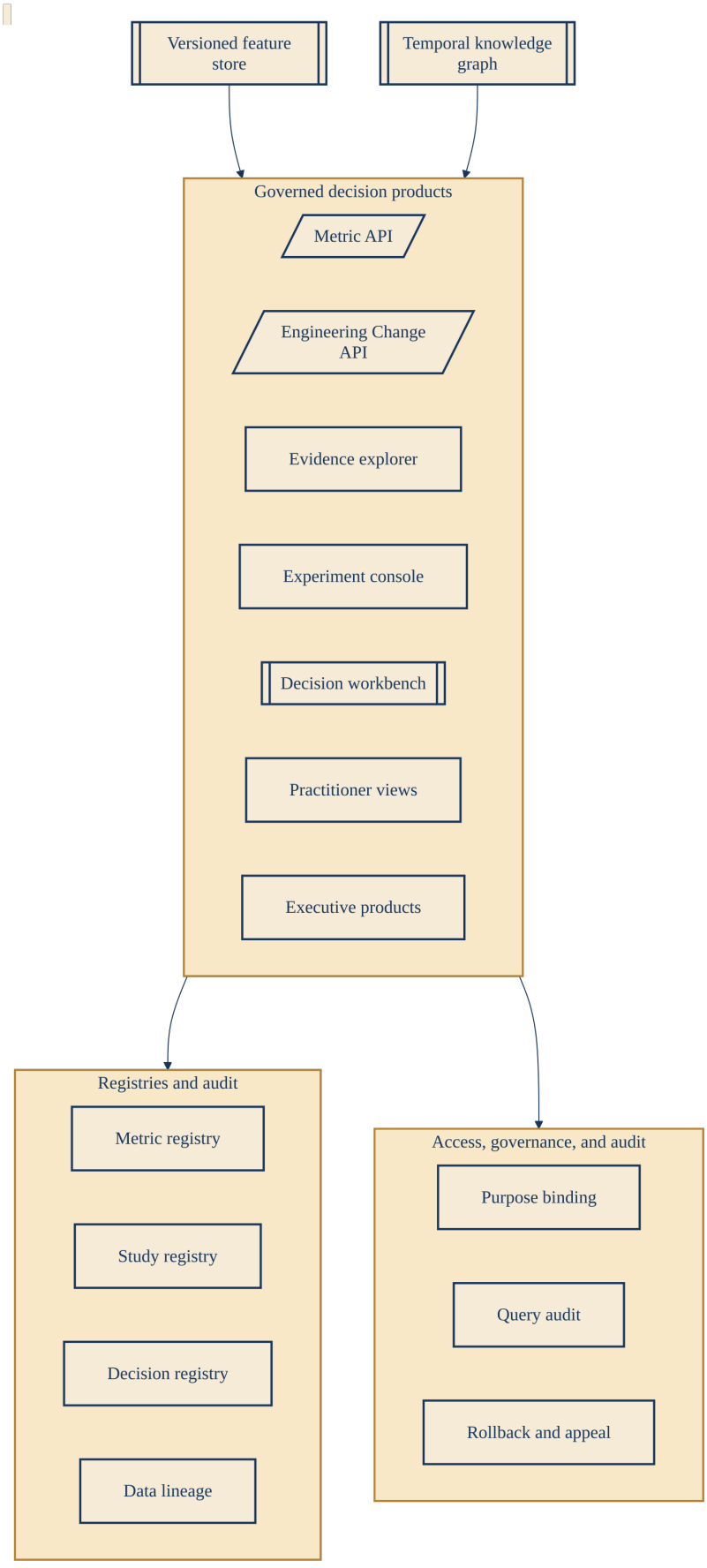
- studies;
- benchmark datasets;
- annotations;
- metric validations;
- incident analyzes;
- architecture studies;
- business cases;

- negative results;
- replication attempts;
- reproducibility packages.

Search should support construct, intervention, population, outcome, system, and decision.

F.31. Serving layer

The serving layer is what users, analysts, and downstream systems actually touch. It exposes governed products backed by the semantic core, records what those products were used for, and enforces the access and auditing rules that the platform depends on for legitimacy. Nothing should reach a user except through a serving product whose purpose, uncertainty, and lineage are traceable.


Versioned feature store
Temporal knowledge graph
Governed decision products
Metric API
Engineering Change API
Evidence explorer
Experiment console
Decision workbench
Practitioner views
Executive products
Registries and audit
Metric registry
Study registry
Decision registry
Data lineage
Access, governance, and audit
Purpose binding
Query audit
Rollback and appeal

Serving patterns include:

F.31.1. Metric API

Returns value, version, population, uncertainty, quality, period, and interpretation reference.

F.31.2. Engineering Change API

Returns lineage, lifecycle, actors, artifacts, validations, deployments, outcomes, and confidence.

F.31.3. Evidence explorer

Supports investigation and correction of linked evidence.

F.31.4. Experiment console

Supports assignment, exposure, guardrails, study status, and reproducibility.

F.31.5. Decision workbench

Combines evidence, economics, alternatives, uncertainty, and decision records.

F.31.6. Practitioner views

Provide locally meaningful team and self-service evidence with correction controls.

F.31.7. Executive products

Present decision-oriented summaries rather than unrestricted metric catalogues.

F.32. API design

APIs should expose semantic versions and quality state.

Example metric response:

```
{
  "metric_id": "ei.review.human_effort_minutes",
  "version": "2.1.0",
  "value": 37.4,
  "unit": "minutes",
  "aggregation": "median",
  "population_size": 1842,
  "quality_status": "publishable",
  "uncertainty": {
    "lower": 35.9,
    "upper": 39.2,
    "level": 0.95
  },
  "lineage_ref": "lineage://metric-run/01J...",
  "interpretation_ref": "metric://ei.review.human_effort_minutes/2.1.0"
```

```
}
```

A degraded metric should return an explicit unavailable or degraded state rather than a stale unqualified number.

F.33. Data lineage

Lineage should connect:

- published value;
- metric code version;
- transformation jobs;
- canonical facts;
- Engineering Change links;
- identity and ownership versions;
- source events;
- connector and source instance.

Lineage must support impact analysis when a source, schema, rule, or metric changes.

F.34. Reproducibility

A reproducible metric or study run should preserve:

- input snapshot or immutable references;
- code and query versions;
- schema versions;
- environment;
- random seeds;
- configuration;
- output checksums;
- execution logs;
- quality report.

Reproducibility should be tested periodically, not merely documented.

F.35. Privacy architecture

Privacy controls should exist in architecture, not only policy.

Required patterns include:

- isolated identity mapping;
- pseudonymous analytical IDs;
- row- and column-level security;
- purpose-bound access tokens or claims;
- minimum group suppression;

- export controls;
- retention enforcement;
- deletion propagation;
- access auditing;
- restricted secure analysis environments.

Generated summaries should not bypass underlying access restrictions.

F.36. Security architecture

The platform is a high-value target because it contains architecture, code, incident, identity, model, and business evidence.

Controls should include:

- zero-trust service authentication;
- workload identity;
- least privilege;
- encryption in transit and at rest;
- secrets isolation;
- network segmentation;
- tamper-evident audit;
- dependency and artifact integrity;
- secure connector execution;
- data-loss prevention;
- threat detection;
- tested incident response.

F.37. Multi-tenancy

Large enterprises may require tenant boundaries for business units, acquisitions, regulated domains, or regions.

The model should define:

- tenant identity;
- shared versus isolated services;
- cross-tenant aggregation policy;
- encryption boundaries;
- identity federation;
- regional storage;
- administrative separation;
- export and replication controls.

Tenant boundaries should not be simulated only through dashboard filters.

F.38. Regional and regulatory architecture

Support for regional constraints may require:

- local ingestion;
- regional raw storage;
- field-level residency;
- regional identity resolution;
- approved aggregate export;
- separate encryption keys;
- deletion and retention differences;
- jurisdiction-specific access review.

Global metrics should document when populations are incomplete because evidence cannot cross boundaries.

F.39. Reliability objectives

The platform should define SLOs by product.

Example targets:

- ingestion freshness;
- source completeness;
- assignment availability;
- exposure-write durability;
- guardrail alert latency;
- metric publication freshness;
- API availability;
- lineage-query success;
- correction propagation time.

Not every analytical product requires real-time processing. Reliability should match decision need.

F.40. Failure modes

Important failure modes include:

- source API outage;
- expired credentials;
- schema drift;
- replay duplication;
- delayed events;
- identity merge error;
- incorrect ownership interval;
- reconstruction regression;
- metric semantic change;

- experiment contamination;
- model drift;
- privacy suppression failure;
- stale dashboard cache;
- unauthorized export.

Each critical failure mode should have detection, containment, recovery, and impact analysis.

F.41. Backfill and replay

Backfill is required for historical baselines, source migrations, and corrected logic.

The platform should support:

- bounded source extraction;
- immutable import manifests;
- rate-aware execution;
- idempotent canonicalization;
- versioned replay;
- side-by-side comparison;
- quality reconciliation;
- consumer impact control.

Historical restatement should be explicit when metric values change.

F.42. Correction workflow

Corrections may originate from practitioners, data owners, source reconciliation, or improved models.

A correction service should:

1. record the disputed entity or relationship;
2. preserve previous state;
3. capture evidence and reason;
4. apply approved correction;
5. identify affected metrics, studies, and decisions;
6. recompute dependent products;
7. notify material consumers;
8. retain correction lineage.

F.43. Observability of the intelligence platform

The platform should monitor itself.

Operational signals include:

- connector lag;

- event volume anomalies;
- duplicate rate;
- dead-letter depth;
- schema rejection;
- identity-resolution coverage;
- linkage coverage and confidence;
- metric quality status;
- model and feature drift;
- experiment assignment and exposure mismatch;
- access anomalies;
- query and serving latency;
- cost by pipeline and tenant.

Platform observability should be separate from the engineering metrics it produces.

F.44. Quality SLOs

Example quality SLOs include:

- 99 percent of source events ingested within six hours;
- fewer than 0.1 percent unresolved duplicate events;
- at least 95 percent repository ownership coverage;
- at least 90 percent deployment-to-change linkage coverage;
- fewer than 2 percent unresolved human identities in governed populations;
- 100 percent of published metrics accompanied by version and quality state.

Targets must be tailored to source capability and decision consequence.

F.45. Cost architecture

Track platform cost by:

- connector;
- source domain;
- storage layer;
- transformation;
- graph computation;
- model inference;
- metric workload;
- tenant;
- serving product;
- retention class.

High-cardinality raw evidence and graph traversal can become expensive. Retention tiers, incremental computation, and workload isolation should be designed early.

F.46. Technology selection criteria

Technology choices should consider:

- event volume and ordering;
- replay requirements;
- schema evolution;
- temporal joins;
- graph traversal depth;
- analytical concurrency;
- regional constraints;
- privacy controls;
- operational expertise;
- cost predictability;
- portability;
- ecosystem integration.

No technology should be chosen solely because it is fashionable or because one source vendor recommends it.

F.47. Deployment topology

A common deployment topology includes:

- isolated connector runtime;
- regional ingestion endpoints;
- durable event bus;
- immutable object store;
- stream and batch canonicalization;
- lakehouse or warehouse;
- graph and search projections;
- operational registries;
- policy-enforced APIs;
- separate production and research workspaces.

Security-sensitive connectors may require dedicated network zones.

F.48. Environment strategy

At minimum, maintain:

- development with synthetic fixtures;
- integration with representative source contracts;
- staging with shadow production flows;
- production;
- restricted research environment;

- disaster-recovery path.

Production identity and sensitive source payloads should not be copied casually into lower environments.

F.49. Testing strategy

F.49.1. Connector tests

- authentication;
- pagination;
- retries;
- replay;
- deletion;
- schema fixtures;
- rate limits;
- duplicate behavior.

F.49.2. Contract tests

- required fields;
- event semantics;
- compatibility;
- source-to-canonical mapping.

F.49.3. Reconstruction tests

- known Engineering Change fixtures;
- deterministic lineage;
- ambiguous cases;
- confidence calibration;
- temporal ownership;
- correction.

F.49.4. Metric tests

- formula;
- population;
- missingness;
- versioning;
- quality gates;
- privacy suppression;
- historical restatement.

F.49.5. Experiment tests

- deterministic assignment;
- balance;
- exposure timing;
- contamination;
- guardrail alerts;
- rollback.

F.49.6. Security and privacy tests

- authorization;
- purpose claims;
- row and field restrictions;
- export controls;
- minimum-group suppression;
- deletion propagation;
- audit completeness.

F.50. Release management

Platform releases should identify changes to:

- connectors;
- canonical schemas;
- reconstruction;
- identity;
- metrics;
- models;
- policy;
- APIs;
- serving products.

High-impact semantic changes should run in shadow mode and include consumer migration plans.

F.51. Disaster recovery

Recovery plans should specify:

- recovery-point objective;
- recovery-time objective;
- event-bus recovery;
- raw evidence restoration;
- registry restoration;
- assignment continuity;

- replay procedure;
- audit preservation;
- regional failover;
- access-control recovery.

The immutable raw layer and registries are foundational recovery assets.

F.52. Staged implementation

F.52.1. Stage 1: evidence foundation

Implement source-control, review, CI, deployment, identity, raw evidence, canonical events, and basic quality monitoring.

F.52.2. Stage 2: Engineering Change graph

Add explicit identifiers, artifact lineage, temporal ownership, reconstruction, lifecycle state, and confidence.

F.52.3. Stage 3: governed metric products

Add metric registry, lineage, quality gates, privacy controls, and decision-oriented serving.

F.52.4. Stage 4: experimentation

Add assignment, exposure, guardrails, study registry, and reproducibility.

F.52.5. Stage 5: economics and portfolio

Connect cost, business outcomes, realization ledgers, and decision records.

F.52.6. Stage 6: AI and agent intelligence

Add model, prompt, agent, delegation, validation burden, and continuous evaluation.

F.52.7. Stage 7: bounded adaptive policy

Add prescriptive recommendations and reversible automated actions under strong governance.

F.53. Minimum viable platform

A credible minimum platform should include:

1. three to five decision-relevant sources;
2. immutable raw storage;
3. canonical event envelope;
4. actor, team, repository, and service identity;
5. Engineering Change reconstruction for one workflow;

6. source and linkage quality dashboards;
7. a small governed metric registry;
8. metric lineage;
9. purpose-based access;
10. correction workflow;
11. one decision product with an accountable owner.

The minimum should be defined by a complete evidence-to-decision path, not by the number of connectors.

F.54. Build sequence

A practical build sequence is:

1. choose a consequential decision;
2. define the required Engineering Change evidence;
3. establish source contracts;
4. ingest and preserve raw events;
5. canonicalize entities and activities;
6. resolve identity and ownership;
7. reconstruct changes and publish confidence;
8. implement governed metrics;
9. expose quality and lineage;
10. deliver the decision product;
11. measure whether the product changed a decision;
12. expand only after the loop works.

F.55. Team topology

A platform team may include:

- product owner;
- data and streaming engineers;
- platform engineers;
- analytics engineers;
- measurement scientist or statistician;
- security and privacy engineering;
- domain representatives;
- UX or decision-product design;
- reliability ownership.

Federated domain stewards remain necessary because central teams cannot infer every workflow correctly.

F.56. Ownership boundaries

Recommended ownership boundaries are:

- connector teams own faithful extraction;
- source domains own source semantics;
- platform owns canonical contracts and raw durability;
- identity domain owns canonical identity and temporal mappings;
- Engineering Intelligence owns reconstruction and metric standards;
- metric owners own interpretation and quality;
- study owners own causal protocol;
- governance owns approved use and access policy;
- decision owners own action and review.

F.57. Architecture decision records

Major platform choices should record:

- decision context;
- alternatives;
- forces and constraints;
- chosen approach;
- expected consequences;
- reversibility;
- evidence;
- review trigger.

Architecture decisions should be revisited when scale, privacy, regulation, or tool capability changes.

F.58. Technical anti-patterns

F.58.1. Dashboard-first architecture

Charts are built before canonical evidence, metric definitions, and lineage.

F.58.2. One giant enterprise table

All sources are flattened into a brittle schema that loses relationships and provenance.

F.58.3. Current-state-only ingestion

Historical transitions, correction, and causal ordering become unrecoverable.

F.58.4. Tool object as universal unit

Pull requests or tickets become the permanent analytical model.

F.58.5. Identity everywhere

Names and email addresses are copied into every table rather than isolated and pseudonymized.

F.58.6. Graph database as semantics

A graph technology is adopted without stable relationship definitions or confidence rules.

F.58.7. Metric SQL in dashboards

Definitions diverge and cannot be versioned or audited.

F.58.8. Experiment flag without exposure

Assignment exists, but actual treatment is unknown.

F.58.9. Generated narrative without evidence links

AI summaries appear authoritative but cannot be traced.

F.58.10. Real-time everything

Cost and complexity rise even when decisions require only daily or weekly freshness.

F.58.11. Raw data access as self-service

Convenience bypasses purpose, interpretation, privacy, and quality controls.

F.59. Technical reference checklist

1. Begin from a consequential decision and complete evidence path.
2. Preserve immutable source evidence with provenance and replay.
3. Normalize into versioned canonical events and entities.
4. Resolve identity and ownership temporally.
5. Reconstruct Engineering Changes with explicit confidence.
6. Maintain complementary analytical, graph, search, and operational projections.
7. Compute metrics from registered specifications with quality gates and lineage.
8. Record experiment assignment separately from actual exposure.
9. Store evidence, metrics, models, studies, and decisions in registries.
10. Enforce privacy, purpose-bound access, retention, correction, and audit in the architecture.
11. Define reliability and quality SLOs by decision consequence.
12. Support backfill, replay, historical restatement, and disaster recovery.
13. Introduce AI and autonomous policy only after evidence and governance foundations are operational.
14. Judge platform success by improved, reproducible decisions rather than connector or dashboard count.

Appendix G: Operating Templates and Records

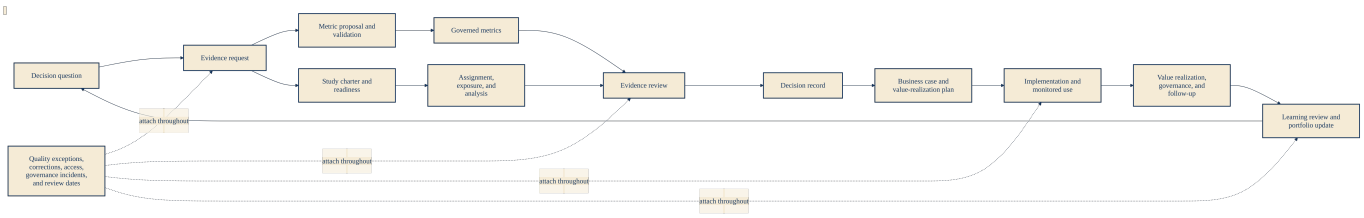
This appendix provides reusable templates for running Engineering Intelligence as an *operating discipline. The templates translate the concepts in this book into records that can be reviewed, versioned, audited, and reused across teams. They are intentionally concise enough for routine use while preserving the evidence, uncertainty, ownership, governance, and follow-up required for consequential decisions.*

G.1. How to use these templates

The templates may be implemented as YAML, JSON, database records, forms, or governed documents.

They should share several conventions:

- durable identifier;
- owner and accountable decision maker;
- semantic or document version;
- creation and review dates;
- evidence and metric references;
- explicit uncertainty;
- privacy and access classification;
- lifecycle status;
- correction history;
- expiration or next-review date.



G.2. Decision question template

Use this before deciding what to measure.

```yaml
decision_question:
  id: dq_01J...
  title: Select enterprise pull-request review approach
  owner: vp_engineering_platform
  decision_due: 2026-10-15
  context: Current review tooling has low practitioner trust and uncertain economic value
  population: enterprise_engineering_repositories
  options:
    - retain_current_tool
    - adopt_vendor_a
    - adopt_vendor_b
```

```
    - build_internal_capability
    - use_no_enterprise_standard
  required_evidence:
    - actionable_finding_precision_and_recall
    - ranking_quality
    - human_validation_effort
    - escaped_defect_guardrails
    - enterprise_cost
    - security_and_operational_fit
  uncertainty_tolerance: medium
  reversibility: moderate
  stakeholders:
    - engineering
    - security
    - procurement
    - finance
    - developer_experience
```

G.3. Evidence request template

Use this to turn a decision question into a bounded analytical request.

```
evidence_request:
  id: er_01J...
  decision_question_id: dq_01J...
  requested_by: developer_productivity
  question: Which review option produces the highest net decision value for eligible changes?
  construct_scope:
    - review_effectiveness
    - human_attention
    - quality_risk
    - economic_value
  populations:
    - medium_risk_application_changes
    - high_risk_application_changes
  required_time_horizon:
    operational: 8_weeks
    quality_follow_up: 30_days
  acceptable_claim_types:
    - descriptive
    - causal
  exclusions:
    - individual_performance_use
  delivery_date: 2026-09-30
  owner: engineering_intelligence
```

G.4. Metric proposal template

```
metric_proposal:
  id: mp_01J...
  proposed_metric_id: ei.review.validation_burden_minutes
  name: Review Validation Burden
  purpose: Compare the human effort required to validate findings from different review systems
  construct: active_human_attention_spent_assessing_and_resolving_review_findings
  decision_users:
```

```
    - developer_productivity
    - tool_procurement
  unit_of_analysis: engineering_change
  proposed_formula: sum(active_validation_session_minutes)
  population: eligible_reviewed_engineering_changes
  data_sources:
    - review_activity
    - finding_disposition
    - actor_identity
    - engineering_change_link
  known_risks:
    - inactive_browser_time
    - unobserved_offline_reasoning
    - multitasking
  prohibited_uses:
    - reviewer_performance_ranking
  owner: measurement_science
  validation_plan: compare_instrumentation_with_sampled_experience_reports
  status: proposed
```

G.5. Metric validation record

```
metric_validation:
  id: mv_01J...
  metric_id: ei.review.validation_burden_minutes
  metric_version: 1.0.0
  construct_validity:
    evidence:
      - practitioner_interviews
      - workflow_walkthroughs
      - correlation_with_experience_sampling
    status: supported_with_limitations
  reliability:
    test_retest: acceptable
    source_consistency: acceptable
  data_quality:
    completeness: 0.97
    identity_resolution: 0.99
    change_linkage: 0.94
  invariance:
    reviewed_dimensions:
      - language
      - repository_type
      - team
    limitations:
      - mobile_and_offline_review_underobserved
  approved_uses:
    - aggregate_tool_evaluation
    - team_workflow_analysis
  prohibited_uses:
    - individual_comparison
  reviewer: metric_governance_board
  reviewed_at: 2026-08-10
  next_review: 2026-11-10
```

G.6. Data-quality exception record

```
data_quality_exception:
  id: dqe_01J...
  affected_product: ei.review.validation_burden_minutes@1.0.0
  detected_at: 2026-08-18T09:20:00Z
  issue: Review-session events missing from one source-control instance
  affected_population: credit_domain_repositories
  affected_period:
    start: 2026-08-16
    end: null
  severity: high
  publication_action: suppress_affected_breakdowns
  decisions_at_risk:
    - dq_01J...
  owner: measurement_platform
  remediation:
    - restore_connector_credentials
    - backfill_events
    - recompute_metric
  resolved_at: null
```

G.7. Engineering Change correction request

```
change_correction_request:
  id: ccr_01J...
  engineering_change_id: ec_01J...
  submitted_by: practitioner_or_domain_steward
  correction_type: deployment_link
  disputed_relationship:
    source: deployment_8821
    target: ec_01J...
    relationship: INCLUDES_CHANGE
  reason: Deployment contained a reverted build that did not include this change
  evidence:
    - artifact_digest
    - deployment_manifest
  requested_action: remove_relationship_and_recompute_dependents
  status: under_review
  reviewer: deployment_data_steward
  submitted_at: 2026-08-22T16:05:00Z
```

G.8. Study charter template

```
study_charter:
  id: exp_01J...
  title: AI review effect on human validation burden
  sponsor: developer_productivity
  study_owner: engineering_intelligence
  decision_owner: vp_engineering_platform
  hypothesis: AI review reduces median human validation effort by at least 15 percent without
degrading severe-defect outcomes
  theory_of_change:
    - routine_findings_are_detected_earlier
```

```
    - authors_correct_before_human_review
    - human_review_focus_shifts_to_domain_and_architecture_risk
  assignment_unit: repository
  analysis_unit: engineering_change
  treatment: ai_reviewer_configuration_4_2
  comparator: current_review_workflow
  primary_outcome: ei.review.validation_burden_minutes@1.0.0
  guardrails:
    - ei.quality.severe_escape_rate@1.0.0
    - ei.change.rollback_rate@2.0.0
  duration: 8_weeks
  follow_up: 30_days
  privacy_classification: restricted
  status: proposed
```

G.9. Experiment readiness review

```
experiment_readiness:
  study_id: exp_01J...
  eligibility_executable: pass
  deterministic_assignment: pass
  treatment_deployable: pass
  exposure_observable: pass
  primary_metric_publishable: pass
  guardrails_publishable: pass
  power_review: pass
  privacy_review: pass
  security_review: pass
  rollback_tested: pass
  known_external_events:
    - annual_release_freeze_during_week_7
  approval: approved_with_condition
  conditions:
    - exclude_release_freeze_from_primary_window
  reviewers:
    - measurement_science
    - security
    - privacy
    - domain_owner
```

G.10. Pre-analysis plan template

```
pre_analysis_plan:
  study_id: exp_01J...
  version: 1.0.0
  estimand: repository_assignment_intention_to_treat_effect
  eligible_population: medium_risk_engineering_changes
  primary_model: covariate_adjusted_cluster_level_regression
  clustering: repository
  covariates:
    - baseline_validation_burden
    - complexity_band
    - repository_change_volume
  missing_data:
    incomplete_outcome_window: exclude_until_mature
```

```
    source_failure: suppress_and_report
  subgroup_analyzes:
    confirmatory:
      - complexity_band
    exploratory:
      - programming_language
  sequential_monitoring: safety_and_data_quality_only
  decision_rule:
    benefit: primary_effect_at_least_15_percent
    quality: severe_escape_noninferior
  registered_at: 2026-08-20T12:00:00Z
  registered_by: study_owner
```

G.11. Study result record

```
study_result:
  study_id: exp_01J...
  analysis_version: 1.0.0
  population:
    assigned_clusters: 120
    engineering_changes: 8240
  exposure:
    treatment_completion_rate: 0.91
    control_contamination_rate: 0.02
  primary_result:
    estimate_relative: -0.18
    interval_95: [-0.24, -0.11]
  guardrails:
    severe_escape_rate: noninferior
    rollback_rate: no_material_difference
  economics:
    released_capacity_hours_annualized: 28,000
    realized_value_status: modeled_not_realized
  limitations:
    - high_risk_repositories_excluded
    - long_term_skill_effect_not_measured
  conclusion: supports_bounded_adoption
  reproducibility_package: evidence://study/exp_01J...
  completed_at: 2026-10-01
```

G.12. Decision record template

```
decision_record:
  id: dec_01J...
  question_id: dq_01J...
  decision: bounded_adoption
  decided_at: 2026-10-15
  owner: vp_engineering_platform
  scope: medium_risk_application_repositories
  alternatives_considered:
    - current_tool
    - vendor_a
    - vendor_b
    - internal_build
  evidence:
```

```
    studies:
      - exp_01J...
    metrics:
      - ei.review.validation_burden_minutes@1.0.0
      - ei.quality.severe_escape_rate@1.0.0
    business_case: bc_01J...
  rationale:
    - strongest_actionable_finding_quality
    - reduced_human_validation_burden
    - acceptable_security_and_operating_model
    - positive_expected_economic_value
  uncertainty:
    - high_risk_effect_unknown
    - long_term_capability_effect_unknown
  conditions:
    - exclude_high_risk_repositories
    - continue_quality_monitoring
    - repeat_evaluation_after_major_model_change
  reversal_criteria:
    - severe_escape_guardrail_breach
    - annual_cost_above_contract_limit
  review_date: 2027-04-15
```

G.13. Business-case summary template

```
business_case_summary:
  id: bc_01J...
  intervention: enterprise_ai_review
  horizon_years: 3
  alternatives:
    - current_tool
    - internal_build
    - no_standard
  incremental_costs:
    vendor: 2200000
    implementation: 450000
    operations_and_governance: 600000
    validation_and_support: 500000
  benefits:
    released_capacity_value: 4200000
    avoided_tool_cost: 1800000
    quality_risk_reduction: 900000
  benefit_labels:
    released_capacity_value: modeled_outcome_value
    avoided_tool_cost: committed_financial_impact
    quality_risk_reduction: modeled_outcome_value
  expected_npv: 6800000
  conservative_npv: 900000
  probability_positive_npv: 0.82
  key_sensitivities:
    - capacity_realization
    - eligible_coverage
    - renewal_price
  realization_owner: developer_productivity
  review_date: 2027-01-15
```

G.14. Value-realization record

```yaml
value_realization:
  id: vr_01J...
  business_case_id: bc_01J...
  period: 2026_Q4
  adoption:
    eligible_coverage: 0.68
    active_teams: 420
  actual_cost:
    vendor: 550000
    operations: 130000
  measured_effect:
    human_validation_minutes_change: -0.16
  released_capacity:
    measured_hours: 6400
    verified_reallocated_hours: 2300
  financial_effect:
    booked_savings: 0
    avoided_cost: 500000
  quality:
    severe_guardrails: within_bounds
  forecast_variance:
    cause: slower_repository_onboarding
  action: continue_with_adoption_remediation
  owner: developer_productivity
```

G.15. Governance use registration

```yaml
governed_use:
  id: use_01J...
  name: Risk-based specialist review routing
  owner: secure_development_platform
  use_class: policy_influence
  purpose: Direct scarce security-review attention to changes with elevated expected risk
  affected_population: production_application_repositories
  data:
    - complexity_features
    - dependency_blast_radius
    - security_sensitive_paths
  models:
    - change_risk_model@3.2.0
  decisions:
    - review_routing
  approved_uses:
    - specialist_queue_assignment
  prohibited_uses:
    - engineer_performance_ranking
    - promotion_decisions
  access:
    raw: restricted
    aggregate: engineering_leaders
  appeal: governance_portal
  expiration: 2027-02-01
```

G.16. Access request template

```yaml
access_request:
  id: ar_01J...
  requester: analyst_identity
  registered_use: use_01J...
  requested_resources:
    - restricted_change_risk_dataset
  required_dimensions:
    - repository
    - complexity_band
  excluded_fields:
    - human_name
    - email
  access_mode: secure_analysis_workspace
  export_required: false
  duration: 60_days
  sponsor: secure_development_platform
  status: pending
```

G.17. Governance exception template

```yaml
governance_exception:
  id: ge_01J...
  requested_by: security_investigations
  policy_exception: temporary_individual_level_access
  purpose: Investigate suspected credential compromise affecting deployment approvals
  scope: named_incident_and_related_events
  necessity: aggregate_evidence_is_insufficient
  compensating_controls:
    - two_person_approval
    - no_export
    - restricted_workspace
    - complete_query_audit
  expires_at: 2026-09-05T00:00:00Z
  reviewers:
    - security
    - privacy
    - legal
  status: approved
```

G.18. Governance incident record

```yaml
governance_incident:
  id: gi_01J...
  detected_at: 2026-09-12T14:03:00Z
  type: prohibited_metric_use
  description: Team-level workflow metric was exported and used to create individual rankings
  severity: high
  affected_population: 34_engineers
  containment:
    - revoke_export
    - remove_rankings
    - suspend_dashboard_access
```

```
  correction:
    - notify_affected_population
    - document_invalid_interpretation
  root_causes:
    - export_control_gap
    - insufficient_prohibited_use_notice
  owner: engineering_intelligence_governance
  status: under_investigation
```

G.19. Architecture decision record

```
architecture_decision:
  id: adr_ei_014
  title: Preserve immutable raw events in object storage
  status: accepted
  context: Canonical models and reconstruction rules will evolve, requiring replay and audit
  decision: Store encrypted source events with checksums and retention classification before
transformation
  alternatives:
    - canonical_tables_only
    - source_system_requery
  consequences:
    positive:
      - replayable_transformations
      - source_audit
      - historical_reconstruction
    negative:
      - storage_cost
      - stronger_access_and_retention_controls_required
  evidence:
    - connector_failure_analysis
    - metric_restatement_requirements
  owner: engineering_intelligence_architecture
  review_trigger: major_change_in_privacy_or_storage_cost
```

G.20. Source onboarding record

```
source_onboarding:
  id: source_ci_enterprise
  source_system: enterprise_ci
  owner: build_platform
  connection_pattern: event_stream_plus_backfill_api
  supported_entities:
    - validation_run
    - build
    - test_result
  event_time_semantics: source_execution_time
  ordering_key: pipeline_run_id
  idempotency_key: source_event_id
  freshness_slo: 2_hours
  completeness_slo: 0.99
  deletion_behavior: tombstone_event
  classification: confidential
  connector_version: 1.0.0
```

```
  status: production
```

G.21. Evidence review agenda

A recurring evidence review should use the following agenda:

1. decisions due before the next review;
2. evidence requests and blocked questions;
3. source and linkage quality exceptions;
4. metric proposals, changes, and retirements;
5. active experiments and guardrails;
6. completed results and required decisions;
7. value-realization variances;
8. governance uses, exceptions, appeals, and incidents;
9. architecture and platform risks;
10. prior decision review dates.

The meeting should not become a general dashboard tour.

G.22. Monthly learning review template

```
learning_review:
  period: 2026_09
  consequential_questions:
    opened: 4
    closed: 3
  decisions:
    adopted: 1
    bounded_adoption: 1
    modified: 1
    stopped: 0
  evidence_quality:
    new_exceptions: 2
    resolved_exceptions: 3
  studies:
    launched: 2
    completed: 1
    replicated: 0
  negative_results_retained: 1
  metrics:
    promoted_to_governed: 2
    deprecated: 1
  realized_value:
    avoided_cost: 500000
    booked_savings: 0
    verified_reallocated_hours: 2300
  governance:
    new_uses: 1
    exceptions: 1
    incidents: 0
  changed_assumptions:
    - ai_review_effect_smaller_for_high_complexity_changes
```

```yaml
  next_actions:
    - launch_high_complexity_replication
    - improve_repository_onboarding
```

G.23. Quarterly portfolio record

```yaml
learning_portfolio:
  period: 2026_Q3
  investments:
    - id: ai_review
      evidence_maturity: randomized
      expected_value: high
      uncertainty: medium
      next_decision: bounded_scale
    - id: ci_cache_expansion
      evidence_maturity: quasi_experimental
      expected_value: medium
      uncertainty: low
      next_decision: expand
    - id: architecture_decomposition
      evidence_maturity: observational
      expected_value: high
      uncertainty: high
      next_decision: pilot
  portfolio_balance:
    short_term_workflow: 0.40
    platform: 0.25
    architecture: 0.20
    long_term_capability: 0.15
  constraints:
    - security_review_capacity
    - data_engineering_capacity
  decisions_required:
    - fund_architecture_pilot
    - retire_low_value_metric_program
```

G.24. Record status vocabulary

Recommended statuses are:

- draft;
- proposed;
- under review;
- approved;
- approved with conditions;
- active;
- degraded;
- suspended;
- superseded;
- deprecated;

- retired;
- rejected;
- completed.

Local systems may map these statuses, but should preserve the canonical meaning.

G.25. Review-date discipline

Every decision, metric, model, governed use, exception, and business case should have a review date.

Review triggers may include:

- elapsed time;
- model or vendor version change;
- population shift;
- metric quality degradation;
- price or contract change;
- architecture migration;
- governance incident;
- guardrail breach;
- evidence contradicting the original assumption.

A record without a review condition tends to become permanent by default.

G.26. Evidence references

References should use durable identifiers rather than presentation URLs alone.

Examples include:

- `metric://ei.review.validation_burden_minutes/1.0.0`;
- `study://exp_01J…/analysis/1.0.0`;
- `evidence://dataset/review_benchmark/2.1.0`;
- `decision://dec_01J…`;
- `model://change_risk_model/3.2.0`;
- `lineage://metric-run/01J…`;
- `business-case://bc_01J….`

The resolver may redirect users to the current governed interface while preserving identity.

G.27. Correction history

Records should not be silently overwritten.

A correction entry should contain:

```
correction:
  corrected_record: dec_01J...
```

```yaml
previous_version: 1.0.0
new_version: 1.0.1
field: evidence.metrics[0]
previous_value: ei.review.validation_burden_minutes@0.9.0
new_value: ei.review.validation_burden_minutes@1.0.0
reason: final_validated_metric_version_was_used_in_analysis
corrected_by: decision_steward
corrected_at: 2026-10-16T09:00:00Z
```

G.28. Minimum operating record set

An organization beginning Engineering Intelligence should implement at least:

1. decision question;
2. evidence request;
3. metric specification and validation;
4. data-quality exception;
5. study charter and result;
6. decision record;
7. governed-use registration;
8. correction request;
9. business-case summary;
10. value-realization record.

This set creates a traceable path from question to evidence, action, and realized outcome.

G.29. Template anti-patterns

G.29.1. Form without ownership

The record is complete, but nobody is accountable for the decision or follow-up.

G.29.2. Required fields with meaningless placeholders

Users enter “N/A” merely to satisfy workflow rather than exposing missing evidence.

G.29.3. Record after the decision

A polished template is created retrospectively to justify a choice already made.

G.29.4. URLs instead of durable references

Evidence disappears when dashboards or documents move.

G.29.5. No review date

Temporary policy becomes permanent.

G.29.6. Templates as bureaucracy

Records are collected but never used in reviews, decisions, corrections, or learning.

G.30. Operating-template checklist

1. Assign durable identifiers and accountable owners.
2. Start from the decision question rather than the available dashboard.
3. Reference versioned metrics, models, studies, and evidence.
4. State uncertainty, limitations, and prohibited uses.
5. Record data quality and linkage exceptions.
6. Separate forecast, modeled value, released capacity, and realized financial impact.
7. Include conditions, reversal criteria, and review dates in decisions.
8. Preserve correction and supersession history.
9. Use registered purposes for access and governance.
10. Retain negative results and stopped interventions.
11. Review records through an operating cadence rather than storing them passively.
12. Keep templates minimal enough to use and complete enough to support accountability.

Appendix H: Glossary and Controlled Vocabulary

***This appendix defines the core vocabulary used throughout** Engineering Intelligence**. The** purpose is not merely editorial consistency. Stable terminology is a prerequisite for valid metrics, interoperable data contracts, reproducible studies, auditable decisions, and trustworthy communication across engineering, product, finance, security, privacy, and executive leadership.*

H.1. Vocabulary principles

H.1.1. Prefer semantic meaning over tool terminology

Use terms such as *Engineering Change*, *validation run*, and *deployment* when describing stable concepts. Preserve tool-specific terms such as pull request, workflow run, or release job as source-system implementations.

H.1.2. Distinguish observation from interpretation

An event is observed evidence. A reconstructed relationship is an inference. A prediction is a model output. A causal effect is an estimate under assumptions. A decision is an organizational choice.

H.1.3. Distinguish people, automation, and agents

Human, deterministic automation, AI assistant, and autonomous agent are separate actor types.

H.1.4. Distinguish assignment from exposure

A unit may be assigned to a treatment without receiving it.

H.1.5. Distinguish capacity from savings

Released engineering time is not automatically cash savings.

H.1.6. Version important semantics

Terms embedded in schemas, metrics, policies, or models should have stable identifiers and versioned definitions.

H.2. A

H.2.1. Acceptance

The explicit or inferred decision that an artifact, finding, recommendation, or Engineering Change is suitable to proceed. Acceptance does not necessarily imply deployment, release, correctness, or realized value.

H.2.2. Acceptance rate

The proportion of eligible suggestions, findings, or artifacts accepted under a defined disposition rule. The denominator,

partial acceptance, later rework, duplicates, and outcome window must be specified.

H.2.3. Accountable owner

The person or role responsible for ensuring a decision, metric, study, model, platform capability, or policy achieves its intended purpose and receives review.

H.2.4. Active time

Estimated time during which a person or system is substantively engaged in an activity. Active time differs from elapsed time and is often imperfectly observed.

H.2.5. Actor

A human, service account, deterministic automation, AI assistant, autonomous agent, vendor system, or organizational unit that initiates or performs an activity.

H.2.6. Agent

A software system that selects and performs actions toward a delegated objective, potentially using models, tools, memory, and other agents. An agent is distinguished from a single model invocation and from fixed automation.

H.2.7. Agentic workflow

A workflow in which one or more agents plan, delegate, execute, validate, or revise actions across multiple steps.

H.2.8. Aggregation

The process of combining observations across units, dimensions, or time. Valid aggregation depends on metric semantics; not all measures are additive.

H.2.9. Analysis unit

The unit represented by each analytical observation or error term. It may differ from the assignment unit in an experiment.

H.2.10. Annotation

A human- or machine-produced label applied to evidence for evaluation, training, classification, or adjudication.

H.2.11. Artifact

A created, modified, reviewed, executed, deployed, or released object, such as source code, configuration, test, package, image, schema, model, document, or deployment manifest.

H.2.12. Assignment

The allocation of an eligible unit to an experimental or policy variant. Assignment is not the same as treatment availability, invocation, completion, or use.

H.2.13. Assignment unit

The entity randomized or allocated to a treatment, such as an Engineering Change, person, repository, team, service, or time window.

H.2.14. Attention

Finite human cognitive capacity applied to understanding, deciding, creating, validating, coordinating, diagnosing, or recovering.

H.2.15. Attention cost

The opportunity cost of human attention consumed by an activity, queue, interruption, tool, or policy.

H.2.16. Attribution

The assignment of an outcome, cost, activity, or benefit to an intervention, actor, team, system, or Engineering Change. Attribution may be descriptive, probabilistic, causal, or accounting-based and should state which.

H.2.17. Audit trail

A durable record of evidence access, transformation, model or metric use, decisions, corrections, overrides, and automated actions.

H.2.18. Automation

A deterministic or rule-based system that performs predefined actions. Automation may use software logic without exercising agent-like planning or delegation.

H.2.19. Autonomous action

An action executed by a system without immediate human approval. Autonomy should be described by scope, consequence, reversibility, evidence threshold, and override controls.

H.2.20. Availability

The degree to which a system, treatment, service, or dataset is operationally accessible when required. Availability does not prove successful use.

H.3. B

H.3.1. Backfill

Historical ingestion or recomputation used to populate evidence from a period before normal processing, restore missing data, or apply corrected logic.

H.3.2. Baseline

The pre-intervention or comparison state used to describe initial conditions, estimate expected outcomes, or improve analytical precision.

H.3.3. Benchmark

A governed set of tasks, examples, artifacts, labels, and evaluation rules used to compare tools, models, systems, or methods.

H.3.4. Benefit

A favorable incremental consequence relative to a credible alternative. Benefits may be cash impact, avoided cost, released capacity, quality improvement, risk reduction, option value, or capability value.

H.3.5. Blast radius

The set and severity of systems, users, data, dependencies, teams, or business outcomes potentially affected by a change or failure.

H.3.6. Blocking time

Elapsed time during which progress cannot continue because a required resource, decision, environment, dependency, or approval is unavailable.

H.3.7. Bot

An imprecise umbrella term for nonhuman activity. Canonical systems should instead identify deterministic automation, service account, AI assistant, or autonomous agent.

H.3.8. Bottleneck

The constrained stage or resource that limits end-to-end system throughput or responsiveness.

H.3.9. Build

The process or result of transforming source artifacts and dependencies into a deployable or distributable artifact.

H.3.10. Business case

A structured comparison of alternatives using incremental costs, benefits, uncertainty, risks, realization conditions, and decision criteria.

H.4. C

H.4.1. Calibration

Agreement between predicted probability or confidence and observed frequency. A calibrated 70 percent prediction should be correct approximately 70 percent of the time in the relevant population.

H.4.2. Canonical event

A source-independent, versioned representation of an observed activity or state transition with preserved provenance.

H.4.3. Canonical identity

A stable identity that connects multiple source-system identities belonging to the same actor while preserving temporal validity and confidence.

H.4.4. Capacity

The amount of work or attention a system or resource can provide over a period under defined operating conditions.

H.4.5. Capacity realization

The proportion of released capacity that becomes valuable productive work, avoided cost, or financial impact.

H.4.6. Carryover

An effect of an earlier treatment period that persists into a later comparison period.

H.4.7. Causal claim

A claim that an intervention changed an outcome relative to the outcome that would otherwise have occurred.

H.4.8. Causal effect

A contrast between potential outcomes under different interventions for a defined population and estimand.

H.4.9. Change failure

A failure, degradation, rollback, incident, or material correction associated with a delivered Engineering Change under a specified attribution rule.

H.4.10. Change type

A governed classification of an Engineering Change, such as feature, defect fix, reliability, security, compliance, platform, experiment, migration, or maintenance.

H.4.11. Closed-loop automation

A system that observes, decides, acts, and evaluates outcomes repeatedly without requiring human approval for every cycle.

H.4.12. Cluster

A group assigned or analyzed together, such as a team, repository, service, or organization.

H.4.13. Cohort

A population grouped by a shared entry time, exposure, characteristic, or lifecycle condition.

H.4.14. Comparator

The alternative condition against which an intervention is evaluated.

H.4.15. Completeness

The degree to which expected evidence is present for the intended population and period.

H.4.16. Complexity

The cognitive, structural, behavioral, dependency, coordination, or operational difficulty associated with understanding and changing a system.

H.4.17. Confidence

A quantified or categorized estimate of how strongly evidence supports a relationship, classification, prediction, or conclusion. Confidence must identify its meaning and calibration.

H.4.18. Confidence band

A controlled category such as confirmed, high, medium, low, or rejected used to communicate relationship strength.

H.4.19. Confidence interval

A frequentist interval produced by a method that would contain the true parameter at the stated long-run rate under repeated sampling and model assumptions.

H.4.20. Confounder

A variable that influences both treatment or exposure and outcome, potentially producing a misleading association.

H.4.21. Consequence

The magnitude and type of impact resulting from an outcome, including human, technical, financial, regulatory, customer, or reputational effects.

H.4.22. Construct

An underlying concept the organization seeks to understand, such as review effort, maintainability, cognitive load, quality, or delivery responsiveness.

H.4.23. Construct validity

The degree to which a measure represents the intended construct rather than a related but different concept.

H.4.24. Contamination

Treatment received by a control unit or otherwise crossing the intended experiment boundary.

H.4.25. Continuity factor

An economic adjustment representing how much released time appears in usable blocks rather than fragmented moments.

H.4.26. Control group

The comparison group that does not receive the intervention of interest or receives the baseline condition.

H.4.27. Correction

A versioned change to erroneous or incomplete evidence, identity, linkage, metric, model output, or decision record that preserves prior state and reason.

H.4.28. Cost of delay

The value lost while an outcome, capability, risk reduction, or decision remains unavailable.

H.4.29. Counterfactual

The outcome that would have occurred under an alternative intervention or decision.

H.4.30. Credible interval

A Bayesian interval containing the parameter with a stated posterior probability under the model and prior assumptions.

H.4.31. Cycle time

Elapsed time between explicitly defined lifecycle events for a defined unit. The start, end, pauses, population, and calendar basis must be specified.

H.5. D

H.5.1. Data contract

A versioned agreement defining source semantics, schemas, identifiers, timing, quality expectations, ownership, change policy, and permitted use.

H.5.2. Data lineage

Traceable relationships from a published result back through models, metrics, transformations, canonical records, and source evidence.

H.5.3. Data minimization

Collection, retention, and processing of only the evidence necessary for an approved purpose.

H.5.4. Data-quality gate

A rule that determines whether a metric, model, study, or decision product may be published or used.

H.5.5. Decision-grade

Sufficiently valid, reliable, timely, relevant, and governed for a specified consequential decision. Decision-grade is use-specific, not universal.

H.5.6. Decision owner

The accountable person or role authorized to choose among alternatives and responsible for reviewing the outcome.

H.5.7. Decision record

A durable statement of the question, alternatives, evidence, assumptions, uncertainty, chosen action, owner, conditions, and review date.

H.5.8. Delegation

The transfer of an objective, authority, or task from one actor to another.

H.5.9. Delegation graph

A representation of human and machine actors, delegated objectives, authority boundaries, actions, and acceptance responsibility.

H.5.10. Deployment

The movement or activation of one or more artifacts in an environment. Deployment is distinct from release to an intended population.

H.5.11. Deployment frequency

The number or rate of eligible deployments for a defined unit, environment, result, and period.

H.5.12. Descriptive claim

A claim about observed or reconstructed patterns without asserting prediction or causation.

H.5.13. Digital twin

A continuously updated computational representation of an engineering system used to simulate or estimate behavior under alternative conditions.

H.5.14. Direct cost

A cost traceable to the intervention, such as license, infrastructure, implementation, or support expense.

H.5.15. Distributional effect

Variation in benefit, burden, risk, or outcome across populations, subgroups, teams, or units.

H.5.16. Drift

A change over time in population, source semantics, feature distribution, model behavior, metric interpretation, treatment, or outcome relationship.

H.6. E

H.6.1. Economic value

The incremental benefit of an intervention relative to a credible alternative after accounting for incremental cost, uncertainty, timing, and risk.

H.6.2. Effect size

The magnitude of a relationship or intervention effect, expressed in absolute, relative, standardized, or distributional terms.

H.6.3. Eligibility

The rules determining which units may enter a metric population, study, treatment, policy, or decision.

H.6.4. Engineering Change

A vendor-independent analytical unit representing an intended modification and its implementation, validation, delivery, operation, and observed outcomes.

H.6.5. Engineering Change graph

The entities and relationships connecting intent, actors, artifacts, activities, validations, builds, deployments, releases, incidents, and outcomes for Engineering Changes.

H.6.6. Engineering Intelligence

The discipline and supporting system for connecting engineering evidence, human attention, architecture, economics, experimentation, governance, and outcomes to improve organizational decisions and learning.

H.6.7. Estimand

The exact quantity a study intends to estimate, including population, treatment contrast, outcome, and handling of events such as noncompliance or missingness.

H.6.8. Event time

The time an event occurred in the source domain.

H.6.9. Evidence

Observed or derived information used to support a claim or decision. Evidence should identify source, provenance, quality, and interpretation level.

H.6.10. Evidence graph

A network linking questions, constructs, metrics, datasets, studies, results, decisions, and replications.

H.6.11. Evidence maturity

The strength and operational readiness of evidence, ranging from exploratory signal to replicated causal evidence and monitored policy.

H.6.12. Exposure

The treatment actually made available, invoked, completed, or used by an eligible unit.

H.6.13. External validity

The degree to which a result applies beyond the studied population, setting, intervention version, and period.

H.7. F

H.7.1. Fairness

The evaluation and governance of whether benefits, burdens, errors, access, and consequences are acceptably distributed across relevant populations.

H.7.2. False negative

An eligible issue, outcome, or condition that exists but is not detected or classified as present.

H.7.3. False positive

A finding, prediction, or classification indicating an issue or condition that is not supported under the adjudication rule.

H.7.4. Feature

A measured or derived input used by a model, rule, analysis, or policy.

H.7.5. Finding

A specific concern, defect, recommendation, policy violation, or observation produced during review or analysis.

H.7.6. Finding acceptance

The disposition that a finding is sufficiently valid and actionable to warrant remediation, acknowledgment, or follow-up.

H.7.7. Finding recall

The proportion of eligible ground-truth findings detected by a reviewer or system.

H.7.8. Finding precision

The proportion of eligible produced findings judged valid and actionable.

H.7.9. Flow

The movement of Engineering Changes through planning, implementation, validation, delivery, and operation.

H.7.10. Fragmentation

The division of attention or released capacity into small intervals that reduce usability and increase switching cost.

H.7.11. Freshness

The delay between source occurrence and evidence or metric availability.

H.8. G

H.8.1. Gaming

Behavior that improves a measured value without improving the intended construct or outcome.

H.8.2. Generated artifact

An artifact produced wholly or partly by an AI model or agent. Generated status does not imply quality or acceptance.

H.8.3. Generated explanation

A machine-produced narrative or summary derived from evidence or other model outputs. It should remain distinguishable from observed evidence.

H.8.4. Governance

The roles, policies, controls, reviews, audits, and accountability mechanisms governing evidence, metrics, models, experiments, automated actions, and decisions.

H.8.5. Governance incident

A material failure involving access, purpose limitation, prohibited use, privacy, metric misuse, unsafe automated action, or missing accountability.

H.8.6. Ground truth

A reference label or outcome treated as authoritative for an evaluation. Ground truth may itself contain uncertainty and should document adjudication.

H.8.7. Guardrail

A measure, threshold, or condition intended to detect unacceptable harm or degradation while optimizing another outcome.

H.9. H

H.9.1. Hazard

The instantaneous rate at which an event occurs among units that have not yet experienced it.

H.9.2. Human-in-the-loop

A system in which a person participates in review, approval, correction, exception handling, or supervision. The phrase should specify the person's actual authority and timing.

H.9.3. Human-on-the-loop

A system that acts autonomously within bounds while a person monitors, audits, or intervenes when necessary.

H.9.4. Human validation burden

The attention and effort required to establish sufficient confidence in machine- or human-produced output.

H.9.5. Hypothesis

A specific, falsifiable statement about a relationship, mechanism, or intervention effect.

H.10. I

H.10.1. Identification strategy

The assumptions and design features allowing an observed contrast to be interpreted causally.

H.10.2. Identity resolution

The process of mapping source identities to canonical actors with temporal validity and confidence.

H.10.3. Immutable evidence

Source evidence preserved in a form that cannot be silently overwritten and can support replay, audit, or correction.

H.10.4. Implementation fidelity

The degree to which an intervention was delivered as designed.

H.10.5. Incident

An operational failure, degradation, security event, or significant near miss requiring coordinated response or investigation.

H.10.6. Incremental cost

The difference in cost between an intervention and the credible alternative.

H.10.7. Incremental value

The difference in value between an intervention and the credible alternative.

H.10.8. Inference

A conclusion, estimate, reconstruction, or prediction derived from evidence rather than directly observed.

H.10.9. Ingestion time

The time evidence entered the Engineering Intelligence platform.

H.10.10. Instrumentation

The technical and procedural mechanisms that create, collect, timestamp, and transmit evidence.

H.10.11. Instrumental variable

A variable used to estimate causal effects when it changes treatment, affects outcome only through treatment under assumptions, and satisfies other identification conditions.

H.10.12. Intention-to-treat effect

The effect of assignment to treatment regardless of whether treatment was received or completed.

H.10.13. Interference

A condition in which one unit’s treatment affects another unit’s outcome.

H.10.14. Internal validity

The credibility of a causal or analytical conclusion for the studied population and setting.

H.10.15. Intervention

A deliberate change to tool, workflow, policy, architecture, platform, staffing, or operating conditions intended to

influence outcomes.

H.11. J

H.11.1. Join coverage

The proportion of eligible records that can be linked across source or canonical entities under the declared rules.

H.12. L

H.12.1. Latency

Elapsed time between defined events. The term should specify whether it refers to system response, queue delay, cycle time, or evidence freshness.

H.12.2. Learning organization

An organization that repeatedly observes, explains, intervenes, evaluates, decides, adapts, and preserves knowledge.

H.12.3. Learning velocity

The rate at which consequential uncertainty is reduced and translated into better decisions and retained knowledge.

H.12.4. Lifecycle state

A semantic state in the progress of an Engineering Change, such as proposed, active, validating, accepted, integrated, deployed, released, operating, completed, reverted, abandoned, or superseded.

H.12.5. Linkage confidence

The strength of evidence that two entities or events belong to the same Engineering Change or relationship.

H.12.6. Loaded labor cost

Compensation plus applicable benefits, payroll costs, facilities, equipment, management, and shared overhead used for economic modeling.

H.12.7. Longitudinal study

A study that observes units or populations across time to measure change, adaptation, retention, or delayed effects.

H.13. M

H.13.1. Maintainability

The ease, safety, and cost with which a system can be understood, modified, validated, operated, and evolved.

H.13.2. Mechanism

The process through which an intervention or condition produces an outcome.

H.13.3. Measurement invariance

The degree to which a measure represents the same construct across groups, contexts, or time.

H.13.4. Measurement model

The explicit relationship between a construct and its observable indicators.

H.13.5. Metric

A versioned measure computed for a defined unit, population, interval, formula, and purpose.

H.13.6. Metric bundle

A coordinated set containing a primary outcome, guardrails, mechanism indicators, economic measures, adoption or exposure measures, and data-quality measures.

H.13.7. Metric incident

A material failure in metric correctness, timeliness, semantics, access, or permitted use.

H.13.8. Metric owner

The person or role accountable for a metric's construct, definition, interpretation, quality, permitted use, review, and retirement.

H.13.9. Missingness

The absence of an expected value or relationship. Missingness should distinguish true zero, not applicable, unobserved, unavailable, unresolved, and not yet mature.

H.13.10. Model

A statistical, machine-learning, simulation, or rule-based representation used to predict, classify, rank, estimate, or recommend.

H.13.11. Model monoculture

Concentration on one model family, provider, or shared failure mode across consequential workflows.

H.13.12. Model provenance

The model identity, version, configuration, training or evaluation context, features, prompt, tools, and policy associated with an output.

H.13.13. Multi-tenancy

An architecture in which multiple organizational or regulatory tenants share some platform components while preserving defined isolation boundaries.

H.14. N

H.14.1. Natural experiment

A situation in which external or operational variation creates a treatment contrast that can support causal inference under defensible assumptions.

H.14.2. Net attention change

Human attention reduced by an intervention minus validation, rework, coordination, and exception-handling attention added.

H.14.3. Net present value

The discounted value of future incremental benefits minus future incremental costs.

H.14.4. Noncompliance

A mismatch between assigned treatment and treatment actually received or followed.

H.14.5. Noninferiority

A study objective establishing that a treatment is not worse than a comparator by more than a predeclared acceptable margin.

H.14.6. Novelty

The degree to which a change introduces unfamiliar concepts, technologies, dependencies, patterns, or operating conditions.

H.15. O

H.15.1. Observable

A value, event, artifact, or behavior that can be directly recorded or reconstructed with stated evidence.

H.15.2. Observability

The ability to infer relevant internal system state from available evidence. Engineering Intelligence observability includes data quality and semantic coverage, not only runtime telemetry.

H.15.3. Observational study

A study in which treatment or exposure is not assigned by the investigator.

H.15.4. Opportunity cost

The value of the best alternative use of money, attention, time, infrastructure, or organizational capacity.

H.15.5. Option value

The value created by preserving or enabling useful future choices under uncertainty.

H.15.6. Outcome

An observed consequence related to user value, business performance, quality, reliability, security, cost, capability, or learning.

H.15.7. Outcome maturity

The point at which sufficient time has elapsed for an outcome to be observed reliably.

H.15.8. Ownership

Responsibility for a system, artifact, metric, source, decision, or policy during a defined interval.

H.16. P

H.16.1. Policy

A rule or decision framework governing eligibility, access, routing, approval, deployment, treatment, or action.

H.16.2. Policy-grade metric

A metric sufficiently mature for formal or automated policy use under continuous quality, monitoring, appeal, and rollback controls.

H.16.3. Population

The complete set of eligible units to which a metric, model, study, or conclusion applies.

H.16.4. Population drift

A change in the composition or characteristics of the eligible or observed population.

H.16.5. Precision

For finding or classification systems, the proportion of positive outputs that are valid under the adjudication rule.

H.16.6. Prediction

A model estimate of an unknown or future value, class, rank, or probability.

H.16.7. Predictive claim

A claim that available evidence can forecast an outcome for a defined population.

H.16.8. Prescriptive claim

A claim that one action should be selected over alternatives to optimize defined objectives under constraints.

H.16.9. Primary outcome

The predeclared outcome carrying the main confirmatory interpretation of a study.

H.16.10. Privacy

The appropriate control of personal or sensitive information, including collection, purpose, access, inference, retention, correction, and disclosure.

H.16.11. Probabilistic linkage

A relationship inferred from multiple uncertain signals rather than an explicit or deterministic reference.

H.16.12. Production function

The relationship between inputs such as labor, attention, tools, capital, and infrastructure and resulting engineering outcomes.

H.16.13. Prompt provenance

The identity and version of instructions, context policy, retrieved evidence, and configuration used for a model invocation.

H.16.14. Propensity score

The estimated probability of receiving treatment given observed pre-treatment characteristics.

H.16.15. Provenance

Information describing the origin, transformation, version, actor, time, and evidence chain of a record or result.

H.16.16. Proxy

An observable measure used to represent another construct. A proxy is not the construct itself.

H.16.17. Purpose limitation

The principle that evidence is collected and used only for declared, reviewed purposes.

H.17. Q

H.17.1. Quality

Fitness of an Engineering Change or system for its intended purpose, including correctness, reliability, security, maintainability, usability, and compliance.

H.17.2. Quality gate

A condition that must be satisfied before an artifact, metric, model, treatment, or decision may proceed.

H.17.3. Queue

Work waiting for a constrained resource or process stage.

H.17.4. Queue delay

Elapsed time spent waiting before service begins.

H.17.5. Quasi-experiment

A nonrandomized study using a defensible identification strategy to estimate a causal effect.

H.18. R

H.18.1. Randomization

Assignment by a chance mechanism designed to make treatment independent of potential outcomes within the target population.

H.18.2. Realized value

Benefit observed and verified after implementation. It should be labeled as booked financial impact, avoided cost, reallocated capacity, or observed outcome value.

H.18.3. Recall

For finding or classification systems, the proportion of eligible true cases detected.

H.18.4. Reconstruction

The derivation of entities, relationships, or lifecycle state from source evidence.

H.18.5. Registered use

An approved purpose-bound use of evidence, metrics, models, or automated policy with defined owner, population, access, controls, and review date.

H.18.6. Regression discontinuity

A causal design comparing units near a treatment-assignment threshold under continuity assumptions.

H.18.7. Reliability

The consistency and reproducibility of a measure, label, relationship, model, or process.

H.18.8. Release

The act of making a deployed change available to an intended user or consuming-system population.

H.18.9. Released capacity

Human or system capacity made available by an intervention. It is not automatically equivalent to savings or additional output.

H.18.10. Replay

Reprocessing preserved evidence through the same or revised transformations, reconstruction, metrics, or models.

H.18.11. Replication

Repeating or independently reproducing a study, measure, or finding to test robustness and boundary conditions.

H.18.12. Reproducibility

The ability to obtain the same result using the same evidence, code, versions, configuration, and analytical procedure.

H.18.13. Retention

The period and conditions under which evidence, identities, model outputs, metrics, and records are preserved.

H.18.14. Reversibility

The degree to which an intervention or automated action can be undone with limited cost or harm.

H.18.15. Review

A structured validation activity in which one or more actors evaluate an Engineering Change, artifact, finding, or decision.

H.18.16. Review effort

Human attention spent performing review. It should distinguish active effort from elapsed review time and queue delay.

H.18.17. Risk

The combination of uncertainty, probability, exposure, and consequence associated with an adverse outcome.

H.18.18. Risk class

A governed category representing expected consequence or control requirements for an Engineering Change or system.

H.18.19. Rollback

An action that removes, disables, or restores the effect of a deployment, release, policy, model, or metric version.

H.19. S

H.19.1. Sample

The observed subset of a population used for measurement or inference.

H.19.2. Schema registry

A governed repository of versioned event, entity, and payload definitions with compatibility and ownership rules.

H.19.3. Sensitivity analysis

Evaluation of how conclusions change under alternative assumptions, parameters, populations, definitions, or missing-data mechanisms.

H.19.4. Service time

Time during which a resource actively processes a unit, excluding queue wait.

H.19.5. Shadow mode

Operation of a tool, model, metric, or policy without allowing it to affect the primary production decision, used for validation and comparison.

H.19.6. Source evidence

The original event, entity, payload, or state obtained from a source system before canonical interpretation.

H.19.7. Spillover

An effect of treatment on units other than those directly assigned or exposed.

H.19.8. Standard error

An estimate of the sampling variability of an estimator.

H.19.9. Statistical significance

A result meeting a predefined probability threshold under a statistical model. It does not imply practical importance, validity, causation, or economic value.

H.19.10. Stratification

Partitioning units into groups before assignment or analysis to improve balance, precision, or interpretation.

H.19.11. Study

A structured investigation designed to answer a descriptive, predictive, causal, or evaluative question.

H.19.12. Study charter

The record defining a study's decision, ownership, hypothesis, population, intervention, outcomes, guardrails, duration, and governance.

H.19.13. Subgroup

A defined subset of a population used to evaluate heterogeneity, fairness, or boundary conditions.

H.19.14. Surveillance

Monitoring people or behavior in a way that exceeds a legitimate, proportionate, transparent, and governed purpose.

H.19.15. Synthetic control

A weighted combination of untreated units used to approximate the counterfactual outcome of a treated unit.

H.19.16. Synthetic evidence

Machine-generated, simulated, or transformed material that resembles observed evidence but must remain labeled as generated or inferred.

H.19.17. System of record

The authoritative source for a defined entity or fact. Authority may be split across domains and time.

H.20. T

H.20.1. Technical debt

Future cost or constrained choice created by technical decisions that make change, validation, operation, or understanding more difficult.

H.20.2. Temporal join

A relationship resolved using values valid at the time of the event rather than current values.

H.20.3. Temporal ownership

Ownership represented as effective intervals so historical evidence is attributed to the correct team or role.

H.20.4. Treatment

The intervention condition assigned or delivered in a study or policy.

H.20.5. Treatment fidelity

The degree to which the actual treatment matches the declared configuration and operational behavior.

H.20.6. Treatment-on-treated effect

The estimated effect among units that actually received treatment, requiring assumptions beyond simple randomized assignment when noncompliance exists.

H.20.7. Throughput

The rate at which eligible units complete a defined process or lifecycle stage.

H.20.8. Tool usage

Observed invocation, interaction, or activity involving a tool. Usage is not proof of effectiveness or value.

H.20.9. Traceability

The ability to follow evidence, artifacts, activities, models, metrics, and decisions through their relationships and versions.

H.20.10. Transportability

The degree to which a causal result from one population or setting can be applied to another using evidence and assumptions.

H.21. U

H.21.1. Uncertainty

Incomplete knowledge about evidence, measurement, model, relationship, outcome, effect, or future state.

H.21.2. Unit of analysis

See *analysis unit*.

H.21.3. Unit of consequence

The entity at which meaningful outcomes and risk are evaluated. In this framework, the Engineering Change is often the principal unit of consequence.

H.21.4. Utilization

The proportion of available service capacity currently consumed. High utilization can produce nonlinear queue growth.

H.22. V

H.22.1. Validation

The process of establishing whether an artifact, finding, measure, model, intervention, or decision is sufficiently correct and fit for its intended use.

H.22.2. Validation burden

Human or system effort required to establish trust in output before acceptance or action.

H.22.3. Validation run

An execution that produces evidence about correctness, quality, safety, performance, compliance, or fitness.

H.22.4. Validity

The degree to which evidence, measurement, inference, or study supports the intended interpretation and use.

H.22.5. Value realization

The process by which forecast benefits become observed capacity, avoided cost, financial impact, risk reduction, or outcomes.

H.22.6. Vendor-neutral evidence

Evidence stored and defined independently of the vendor product that generated it, enabling comparison, audit, migration, and longitudinal analysis.

H.22.7. Version

A stable identifier for the semantics or implementation of a schema, metric, model, prompt, policy, treatment, study, or record.

H.23. W

H.23.1. Wait time

Elapsed time during which a unit is ready but has not begun receiving the required service.

H.23.2. Work in progress

The number of units started but not completed within a defined lifecycle boundary.

H.23.3. Work mix

The distribution of change types, complexity, risk, domains, technologies, and other characteristics in a population.

H.23.4. Work-mix adjustment

Analytical control for differences in population composition when making comparisons.

H.23.5. Workflow

The activities, states, decisions, actors, and transitions through which an Engineering Change or related unit progresses.

H.24. Controlled distinctions

H.24.1. Engineering Change versus pull request

An Engineering Change is the semantic unit of intended consequence. A pull request is one possible implementation artifact or review container.

H.24.2. Deployment versus release

A deployment places artifacts into an environment. A release exposes functionality to an intended population.

H.24.3. Activity versus outcome

Activity records work performed. Outcome records consequence or value.

H.24.4. Elapsed time versus active effort

Elapsed time includes waiting and inactivity. Active effort attempts to represent substantive attention or processing.

H.24.5. Usage versus effectiveness

Usage establishes interaction. Effectiveness establishes improvement in a defined outcome.

H.24.6. Adoption versus exposure

Adoption describes broader use or acceptance of a tool or practice. Exposure records whether a specific unit received a treatment.

H.24.7. Correlation versus causation

Correlation describes association. Causation claims an intervention changed the outcome under an identification strategy.

H.24.8. Prediction versus causal estimate

Prediction estimates an unknown outcome. A causal estimate compares potential outcomes under interventions.

H.24.9. Confidence versus probability

Confidence may refer to relationship strength, model score, or interval procedure. Probability must identify the random event or posterior statement.

H.24.10. Precision versus accuracy

Precision in information retrieval is the proportion of positive findings that are valid. Accuracy is the proportion of all classifications that are correct and can be misleading with class imbalance.

H.24.11. Capacity value versus cash savings

Capacity value estimates potential economic usefulness of released time. Cash savings require an actual or committed financial change.

H.24.12. Avoided cost versus savings

Avoided cost prevents a planned future expenditure. Savings reduce an existing or committed expense.

H.24.13. Generated explanation versus observed evidence

A generated explanation interprets evidence. It is not itself the original event or outcome.

H.24.14. Automation versus agent

Automation follows predefined logic. An agent selects and sequences actions toward a delegated objective.

H.24.15. Review finding versus defect

A review finding is an assertion requiring disposition. A defect is a validated undesirable condition.

H.24.16. Quality signal versus quality outcome

A signal is an indicator such as test failure or review finding. An outcome is an observed consequence such as escaped defect, rollback, or user impact.

H.24.17. Metric quality versus engineering quality

Metric quality describes whether measurement is trustworthy. Engineering quality describes the fitness of the system or change.

H.24.18. Data freshness versus event latency

Freshness is delay before evidence becomes available. Event latency may describe operational processing or user-facing response time.

H.24.19. Governance approval versus scientific validity

Approval authorizes a use under controls. It does not establish that the metric, model, or causal claim is scientifically valid.

H.24.20. Reproducibility versus replication

Reproducibility reruns the same method and evidence. Replication tests the result in new data, populations, periods, or

implementations.

H.24.21. Internal validity versus external validity

Internal validity concerns correctness for the studied setting. External validity concerns generalization beyond it.

H.24.22. Policy versus treatment

A policy is an ongoing decision rule. A treatment is a condition evaluated or delivered to eligible units.

H.24.23. Guardrail versus target

A target is an outcome to improve. A guardrail defines unacceptable degradation or harm.

H.24.24. Correction versus deletion

Correction preserves and amends history. Deletion removes evidence according to legal or policy requirements.

H.25. Vocabulary governance

Terms used in canonical schemas, metric specifications, model registries, experiment protocols, and governance records should be governed through:

- stable term identifier;
- preferred label;
- definition;
- permitted synonyms;
- deprecated terms;
- owner;
- semantic version;
- affected schemas and products;
- effective date;
- review date.

Example:

```
term:
  id: ei.term.engineering_change
  preferred_label: Engineering Change
  version: 1.0.0
  definition: Vendor-independent unit representing an intended modification and its
implementation, validation, delivery, operation, and observed outcomes
  deprecated_synonyms:
    - universal_pull_request
  owner: engineering_intelligence_semantics
  effective_date: 2026-07-01
  next_review: 2027-01-01
```

H.26. Vocabulary checklist

1. Use stable semantic terms rather than source-vendor objects where possible.
2. Define every metric term independently of its current implementation.
3. Distinguish observed evidence, reconstruction, prediction, causal estimate, and decision.
4. Distinguish human, automation, assistant, and autonomous-agent activity.
5. Distinguish assignment, availability, invocation, exposure, adoption, and outcome.
6. Distinguish elapsed time, queue delay, service time, and active effort.
7. Distinguish released capacity, avoided cost, savings, and realized value.
8. Version definitions used in schemas, metrics, models, policies, and records.
9. Deprecate ambiguous terms rather than silently changing their meaning.
10. Review vocabulary when tools, workflows, architecture, or organizational practice changes.

Appendix I: Evidence Base and Bibliography

***This appendix provides a curated evidence base for the concepts developed throughout** Engineering Intelligence. It includes foundational research in software delivery, developer productivity, socio-technical systems, causal inference, measurement, queueing, organizational learning, AI evaluation, governance, and engineering economics. The bibliography is not intended to imply that every claim in this book has already been settled. It identifies the intellectual foundations, established methods, and important areas of active research that support or challenge the framework.*

I.1. How to use this evidence base

The sources are grouped by purpose rather than alphabetically.

They can be used to:

- support chapter-level citations;
- design metric and experiment standards;
- distinguish established results from emerging claims;
- identify original methods rather than secondary summaries;
- guide further reading and replication;
- maintain a versioned source registry for the book.

Where possible, entries include a DOI, official publication page, standards body, or publisher reference.

Software delivery and organizational performance

[] **Forsgren, Nicole; Humble, Jez; and Kim, Gene.** *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution Press, 2018. ISBN 978-1942788331.

A foundational synthesis of research connecting software-delivery capabilities, organizational culture, delivery performance, and organizational outcomes.

[] **Forsgren, Nicole; Storey, Margaret-Anne; Maddila, Chandra; Zimmermann, Thomas; Houck, Brian; and Butler, Jenna.** "The SPACE of Developer Productivity: There's More to It Than You Think." *ACM Queue*, volume 19, issue 1, 2021. ACM Queue [https://queue.acm.org/detail.cfm?id=3454124].

Defines developer productivity as multidimensional across satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow.

[] **Noda, Abi; Storey, Margaret-Anne; Forsgren, Nicole; and Greiler, Michaela.** "DevEx: What Actually Drives Productivity." *ACM Queue*, volume 21, issue 2, 2023. ACM Queue [https://queue.acm.org/detail.cfm?id=3595878].

Develops a practical developer-experience framework centered on feedback loops, cognitive load, and flow state.

[] **Fagerholm, Fabian, and Münch, Jürgen.** "Developer Experience: Concept and Definition." *2012 International Conference on Software and System Process*, 2012. DOI [https://doi.org/10.1109/ICSSP.2012.6225984].

An early formal treatment of developer experience as a distinct construct.

[] **Kim, Gene; Humble, Jez; Debois, Patrick; Willis, John; and Forsgren, Nicole.** *The DevOps Handbook*, second edition. IT Revolution Press, 2021. ISBN 978-1950508402.

Provides a broad operating model for flow, feedback, experimentation, reliability, and organizational learning.

[] **Humble, Jez, and Farley, David.** *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley, 2010. ISBN 978-0321601919.

A foundational reference for deployment pipelines, automation, validation, and reliable release practices.

Socio-technical systems and software architecture

[] **Conway, Melvin E.** "How Do Committees Invent?" *Datamation*, volume 14, number 4, 1968.

Introduces the observation now known as Conway's Law: system structures tend to reflect the communication structures of the organizations that design them.

[] **Cataldo, Marcelo; Wagstrom, Patrick A.; Herbsleb, James D.; and Carley, Kathleen M.** "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools." *Proceedings of CSCW 2006*, 2006. DOI [https://doi.org/10.1145/1180875.1180929].

Develops socio-technical congruence as a way to analyze alignment between technical dependencies and coordination.

[] **Herbsleb, James D.** "Global Software Engineering: The Future of Socio-technical Coordination." *Future of Software Engineering*, 2007. DOI [https://doi.org/10.1109/FOSE.2007.11].

Reviews coordination, distance, communication, and dependency challenges in distributed software engineering.

[] **Parnas, David L.** "On the Criteria To Be Used in Decomposing Systems into Modules." *Communications of the ACM*, volume 15, issue 12, 1972. DOI [https://doi.org/10.1145/361598.361623].

Establishes information hiding and design-decision-based modularity.

[] **Martin, Robert C.** *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Prentice Hall, 2017. ISBN 978-0134494166.

A practitioner-oriented treatment of dependency direction, boundaries, and architectural stability.

[] **Skelton, Matthew, and Pais, Manuel.** *Team Topologies: Organizing Business and Technology Teams for Fast Flow.* IT Revolution Press, 2019. ISBN 978-1942788812.

Connects organizational structure, cognitive load, interaction modes, and software-delivery flow.

Human attention, cognition, and interruption

[] **Sweller, John.** "Cognitive Load During Problem Solving: Effects on Learning." *Cognitive Science*, volume 12, issue 2, 1988. DOI [https://doi.org/10.1207/s15516709cog1202_4].

A foundational treatment of cognitive load and its effects on learning and problem solving.

[] **Mark, Gloria; Gudith, Daniela; and Klocke, Ulrich.** "The Cost of Interrupted Work: More Speed and Stress." *Proceedings of CHI 2008*, 2008. DOI [https://doi.org/10.1145/1357054.1357072].

Examines the performance and stress effects of interruption.

[] **Meyer, André N.; Fritz, Thomas; Murphy, Gail C.; and Zimmermann, Thomas.** "Software Developers' Perceptions of Productivity." *Proceedings of FSE 2014*, 2014. DOI [https://doi.org/10.1145/2635868.2635892].

Studies how developers perceive productive and unproductive work.

[] **Müller, Sebastian C., and Fritz, Thomas.** "Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress." *Proceedings of ICSE 2015*, 2015. DOI [https://doi.org/10.1109/ICSE.2015.334].

Connects affective state, progress, and software-development work.

Measurement theory and metrics

[] **Stevens, S. S.** "On the Theory of Scales of Measurement." *Science*, volume 103, issue 2684, 1946. DOI [https://doi.org/10.1126/science.103.2684.677].

A foundational account of nominal, ordinal, interval, and ratio measurement scales.

[] **Cronbach, Lee J.** "Coefficient Alpha and the Internal Structure of Tests." *Psychometrika*, volume 16, 1951. DOI [https://doi.org/10.1007/BF02310555].

Introduces coefficient alpha as a measure of internal consistency.

[] **Campbell, Donald T.** "Assessing the Impact of Planned Social Change." *Evaluation and Program Planning*, volume 2, issue 1, 1979. DOI [https://doi.org/10.1016/0149-7189(79)90,048-X].

Discusses how indicators become corrupted when heavily used for decision and control.

[] **Goodhart, Charles A. E.** "Problems of Monetary Management: The U.K. Experience." In *Monetary Theory and Practice*, Macmillan, 1984.

The source associated with Goodhart's Law: a measure can cease to be useful when it becomes a target.

[] **Messick, Samuel.** "Validity of Psychological Assessment: Validation of Inferences from Persons' Responses and Performances as Scientific Inquiry into Score Meaning." *American Psychologist*, volume 50, issue 9, 1995. DOI [https://doi.org/10.1037/0003-066X.50.9.741].

A major unified treatment of validity as evidence supporting interpretation and use.

[] **Hand, David J.** *Measurement Theory and Practice: The World Through Quantification.* Arnold, 2004. ISBN 978-0340692486.

A broad reference on quantification, measurement design, and interpretation.

Queueing, flow, and operations

[] **Little, John D. C.** "A Proof for the Queuing Formula: L = ¬W." *Operations Research*, volume 9, issue 3, 1961. DOI [https://doi.org/10.1287/opre.9.3.383].

Provides the formal proof of Little's Law relating work in progress, throughput, and flow time.

[] **Kingman, J. F. C.** "The Single Server Queue in Heavy Traffic." *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 57, issue 4, 1961. DOI [https://doi.org/10.1017/S0305004100036094].

Provides a foundational approximation for waiting time under variable arrival and service processes.

[] **Hopp, Wallace J., and Spearman, Mark L.** *Factory Physics*, third edition. Waveland Press, 2011. ISBN 978-

1577667391.

A comprehensive reference on flow, variability, capacity, queues, work in progress, and operating-system behavior.

[] **Goldratt, Eliyahu M., and Cox, Jeff.** *The Goal: A Process of Ongoing Improvement.* North River Press, 1984.

A practitioner-oriented introduction to constraints and system-level optimization.

Causal inference and experimentation

[] **Rubin, Donald B.** "Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies." *Journal of Educational Psychology*, volume 66, issue 5, 1974. DOI [https://doi.org/10.1037/h0037350].

A foundational statement of the potential-outcomes framework.

[] **Holland, Paul W.** "Statistics and Causal Inference." *Journal of the American Statistical Association*, volume 81, issue 396, 1986. DOI [https://doi.org/10.1080/01621459.1986.10478354].

Clarifies the fundamental problem of causal inference and the role of statistical design.

[] **Pearl, Judea.** *Causality: Models, Reasoning, and Inference*, second edition. Cambridge University Press, 2009. ISBN 978-0521895606.

A foundational treatment of structural causal models and graphical causal inference.

[] **Hernán, Miguel A., and Robins, James M.** *Causal Inference: What If.* Chapman & Hall/CRC, 2020. Official author page [https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/].

A modern, accessible reference on target trials, confounding, selection, time-varying treatment, and causal estimation.

[] **Imbens, Guido W., and Rubin, Donald B.** *Causal Inference for Statistics, Social, and Biomedical Sciences.* Cambridge University Press, 2015. ISBN 978-0521885881.

A rigorous treatment of potential outcomes, assignment mechanisms, experiments, and observational studies.

[] **Shadish, William R.; Cook, Thomas D.; and Campbell, Donald T.** *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Houghton Mifflin, 2002. ISBN 978-0395615560.

A core reference for randomized and quasi-experimental design and validity.

[] **Angrist, Joshua D., and Pischke, Jörn-Steffen.** *Mostly Harmless Econometrics: An Empiricist's Companion.* Princeton University Press, 2009. ISBN 978-0691120355.

A practical reference for regression, instrumental variables, regression discontinuity, and difference-in-differences.

[] **Kohavi, Ron; Tang, Diane; and Xu, Ya.** *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing.* Cambridge University Press, 2020. ISBN 978-1108724265.

A practical guide to experimentation infrastructure, guardrails, metrics, power, and organizational adoption.

Ranking and information-retrieval evaluation

[] **Järvelin, Kalervo, and Kekäläinen, Jaana.** "Cumulated Gain-Based Evaluation of IR Techniques." *ACM Transactions on Information Systems*, volume 20, issue 4, 2002. DOI [https://doi.org/10.1145/582415.582418].

Introduces discounted cumulative gain and normalized discounted cumulative gain for ranking evaluation.

[] **Manning, Christopher D.; Raghavan, Prabhakar; and Schütze, Hinrich.** *Introduction to Information Retrieval*. Cambridge University Press, 2008. Official book site [https://nlp.stanford.edu/IR-book/].

A standard reference for precision, recall, ranking, indexing, and evaluation.

Organizational learning, safety, and resilience

[] **Argyris, Chris, and Schön, Donald A.** *Organizational Learning: A Theory of Action Perspective.* Addison-Wesley, 1978.

Develops single-loop and double-loop organizational learning.

[] **Senge, Peter M.** *The Fifth Discipline: The Art and Practice of the Learning Organization.* Doubleday, 1990. ISBN 978-0385260954.

A foundational practitioner text on systems thinking and learning organizations.

[] **Weick, Karl E., and Sutcliffe, Kathleen M.** *Managing the Unexpected: Resilient Performance in an Age of Uncertainty*, second edition. Jossey-Bass, 2007. ISBN 978-0787996499.

Examines high-reliability organizing, attention to weak signals, and resilience.

[] **Dekker, Sidney.** *The Field Guide to Understanding "Human Error"*, third edition. CRC Press, 2014. ISBN 978-1472439055.

Provides a systems-oriented alternative to individual blame in incident analysis.

[] **Woods, David D.** "Four Concepts for Resilience and the Implications for the Future of Resilience Engineering." *Reliability Engineering & System Safety*, volume 141, 2015. DOI [https://doi.org/10.1016/j.ress.2015.03.018].

Defines resilience concepts including rebound, robustness, graceful extensibility, and sustained adaptability.

Economics and productivity

[] **Solow, Robert M.** "We'd Better Watch Out." *New York Times Book Review*, July 12, 1987.

Contains the observation later called the productivity paradox: computers were visible everywhere except in measured productivity statistics.

[] **Brynjolfsson, Erik.** "The Productivity Paradox of Information Technology." *Communications of the ACM*, volume 36, issue 12, 1993. DOI [https://doi.org/10.1145/163298.163309].

Analyzes measurement, learning, redistribution, and organizational explanations for delayed information-technology productivity gains.

[] **Brynjolfsson, Erik; Rock, Daniel; and Syverson, Chad.** "Artificial Intelligence and the Modern Productivity Paradox: A Clash of Expectations and Statistics." NBER Working Paper 24001, 2017. DOI [https://doi.org/10.3386/w24,001].

Connects general-purpose technology adoption to complementary innovation and delayed measured productivity.

[] **Boardman, Anthony E.; Greenberg, David H.; Vining, Aidan R.; and Weimer, David L.** *Cost-Benefit Analysis: Concepts and Practice*, fifth edition. Cambridge University Press, 2018. ISBN 978-1108401296.

A comprehensive reference for economic evaluation, discounting, risk, and sensitivity analysis.

AI-assisted software engineering

[] **Peng, Sida; Kalliamvakou, Eirini; Cihon, Peter; and Demirer, Mert.** "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot." 2023. arXiv [https://arxiv.org/abs/2302.6,590].

Reports a controlled study of task-completion time with and without an AI coding assistant. Interpretation should remain bounded to the task, population, model, and study design.

[] **Vaithilingam, Priyan; Zhang, Tianyi; and Glassman, Elena L.** "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models." *CHI Extended Abstracts 2022*, 2022. DOI [https://doi.org/10.1145/3491101.3519665].

Examines usability, understanding, debugging, and trust in AI code generation.

[] **Barke, Shraddha; James, Michael B.; and Polikarpova, Nadia.** "Grounded Copilot: How Programmers Interact with Code-Generating Models." *Proceedings of the ACM on Programming Languages*, volume 7, OOPSLA1, 2023. DOI [https://doi.org/10.1145/3586030].

Studies interaction patterns between programrs and code-generating models.

[] **Jimenez, Carlos E.; Yang, John; Wettig, Alexander; Yao, Shunyu; Pei, Kexin; Press, Ofir; and Narasimhan, Karthik.** "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *ICLR 2024*. arXiv [https://arxiv.org/abs/2310.6,770].

Introduces a benchmark using real repository issues and patches for evaluating software-engineering agents.

[] **Afroz, Sadia; Feng, Zixuan; Kimura, Katie; Trinkenreich, Bianca; Steinmacher, Igor; and Sarma, Anita.** "Developer Productivity with GenAI." 2025. arXiv [https://arxiv.org/abs/2510.24,265].

Examines perceived productivity effects across SPACE dimensions and highlights variation by usage and outcome dimension.

AI risk, governance, and standards

[] **National Institute of Standards and Technology.** *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* NIST AI 100-1, January 2023. DOI [https://doi.org/10.6028/NIST.AI.100-1].

Defines the Govern, Map, Measure, and Manage functions for voluntary AI risk management.

[] **National Institute of Standards and Technology.** *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile.* NIST AI 600-1, July 2024. DOI [https://doi.org/10.6028/NIST.AI.600-1].

Extends the AI RMF with generative-AI-specific risks and actions.

[] **National Institute of Standards and Technology.** *NIST Privacy Framework: A Tool for Improving Privacy through Enterprise Risk Management*, version 1.0, January 2020. Official NIST page [https://www.nist.gov/privacy-framework].

Provides a risk-based framework for identifying and managing privacy risk.

[] **National Institute of Standards and Technology.** *The NIST Cybersecurity Framework (CSF) 2.0.* February 2024. DOI [https://doi.org/10.6028/NIST.CSWP.29].

Defines the Govern, Identify, Protect, Detect, Respond, and Recover functions for cybersecurity risk management.

[[001]] **International Organization for Standardization and International Electrotechnical Commission.** *ISO/IEC 42,001:2023 — Information Technology — Artificial Intelligence — Management System.* 2023. ISO publication page [https://www.iso.org/standard/81,230.html].

Defines requirements for an organizational AI management system.

[[894]] **International Organization for Standardization and International Electrotechnical Commission.** *ISO/IEC 23,894:2023 — Information Technology — Artificial Intelligence — Guidance on Risk Management.* 2023. ISO publication page [https://www.iso.org/standard/77,304.html].

Provides guidance for AI-specific risk management.

Data systems, provenance, and observability

[] **Moreau, Luc, and Missier, Paolo, editors.** *PROV-DM: The PROV Data Model.* W3C Recommendation, April 30, 2013. W3C Recommendation [https://www.w3.org/TR/prov-dm/].

Defines a general model for entities, activities, agents, and provenance relationships.

[] **Kleppmann, Martin.** *Designing Data-Intensive Applications.* O'Reilly Media, 2017. ISBN 978-1449373320.

A broad reference for event logs, stream processing, replication, consistency, batch processing, and data-system architecture.

[] **Akidau, Tyler; Chernyak, Slava; and Lax, Reuven.** *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing.* O'Reilly Media, 2018. ISBN 978-1491983874.

Develops event-time, watermarks, triggers, and correctness concepts for stream processing.

[] **Beyer, Betsy; Jones, Chris; Petoff, Jennifer; and Murphy, Niall Richard, editors.** *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, 2016. Official online edition [https://sre.google/sre-book/table-of-contents/].

A foundational reference for service-level objectives, toil, reliability engineering, and operational learning.

[] **Beyer, Betsy; Murphy, Niall Richard; Rensin, David K.; Kawahara, Kent; and Thorne, Stephen, editors.** *The Site Reliability Workbook.* O'Reilly Media, 2018. Official online edition [https://sre.google/workbook/table-of-contents/].

Provides operational practices for SLOs, monitoring, incident response, and reliability programs.

Research design, reproducibility, and reporting

[] **Open Science Collaboration.** "Estimating the Reproducibility of Psychological Science." *Science*, volume 349, issue 6251, 2015. DOI [https://doi.org/10.1126/science.aac4716].

A large collaborative replication project illustrating the importance and difficulty of reproducibility.

[] **Nosek, Brian A.; Ebersole, Charles R.; DeHaven, Alexander C.; and Mellor, David T.** "The Preregistration Revolution." *Proceedings of the National Academy of Sciences*, volume 115, issue 11, 2018. DOI [https://doi.org/10.1073/pnas.1708274114].

Explains how preregistration separates confirmatory and exploratory analysis.

[] **Wilkinson, Mark D., et al.** "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data*, volume 3, 2016. DOI [https://doi.org/10.1038/sdata.2016.18].

Defines principles for making research data findable, accessible, interoperable, and reusable.

[] **Association for Computing Machinery.** *ACM Code of Ethics and Professional Conduct.* 2018. ACM [https://www.acm.org/code-of-ethics].

Provides professional principles concerning harm, fairness, privacy, honesty, competence, and accountability.

I.2. Source-evidence classification

The book uses the following evidence categories:

I.2.1. Foundational theory

Long-standing conceptual or mathematical work, such as measurement scales, causal inference, modularity, queueing, or organizational learning.

I.2.2. Peer-reviewed empirical evidence

Published studies with explicit data, methods, and review.

I.2.3. Standards and official frameworks

Normative or advisory publications from recognized standards bodies and public institutions.

I.2.4. Practitioner synthesis

Books and articles integrating research and field practice.

I.2.5. Emerging preprint

Current research that may be useful but has not necessarily completed peer review or replication.

I.2.6. Organizational case evidence

Evidence from one company, tool, or operating context. Such evidence may be highly relevant but should not be assumed universally transportable.

I.3. Citation-quality rules

When adding citations to the body of the book:

1. cite the original source where available;
2. distinguish peer-reviewed work from preprints and practitioner material;
3. avoid treating standards as proof of empirical effectiveness;
4. avoid using one study to support a universal claim;
5. include study population and setting when material;
6. cite methods separately from substantive findings;
7. identify sources with conflicts of interest or vendor sponsorship;
8. preserve negative or conflicting evidence;
9. review time-sensitive sources and standards before publication;
10. use stable citation keys and durable publication identifiers.

I.4. Evidence gaps and active research areas

Important areas remain unsettled or underdeveloped:

- long-term effects of AI assistance on engineering skill formation;
- causal effects of AI tools on production quality and business outcomes;
- reliable measurement of active cognitive effort;
- transportability of developer-productivity studies across organizations;
- architecture metrics with strong construct and predictive validity;
- economic realization of released engineering capacity;
- evaluation of multi-agent software-engineering systems;
- correlated failure and model-monoculture risk;
- privacy-preserving cross-organization benchmarks;
- governance mechanisms that protect trust without blocking useful learning;
- digital twins and prescriptive policies for engineering organizations.

These gaps should be treated as a research agenda rather than filled with unsupported certainty.

I.5. Bibliography maintenance checklist

1. Prefer original papers, official standards, and authoritative editions.
2. Store stable citation keys and publication identifiers.
3. Label preprints and emerging evidence clearly.
4. Review standards and recent AI research before each edition.
5. Add citations to chapter claims rather than leaving references only in this appendix.
6. Include evidence that challenges the preferred interpretation.
7. Preserve historical editions when definitions or standards change.
8. Record retrieval or verification dates for web-only sources.
9. Avoid vendor marketing as the sole source for effectiveness claims.
10. Treat the bibliography as a governed, versioned part of the Engineering Intelligence evidence system.

Appendix J: Maturity Assessment and Adoption Roadmap

This appendix provides a structured method for assessing an organization's current *Engineering Intelligence capability and planning its next stages. It is designed for organizations that already possess substantial engineering telemetry but lack a coherent evidence-to-decision system. The assessment evaluates evidence quality, semantics, measurement, experimentation, economics, governance, platform architecture, organizational learning, and AI readiness. It also provides a practical first-90-day launch plan and a staged roadmap for advancing without attempting an enterprise-wide transformation all at once.*

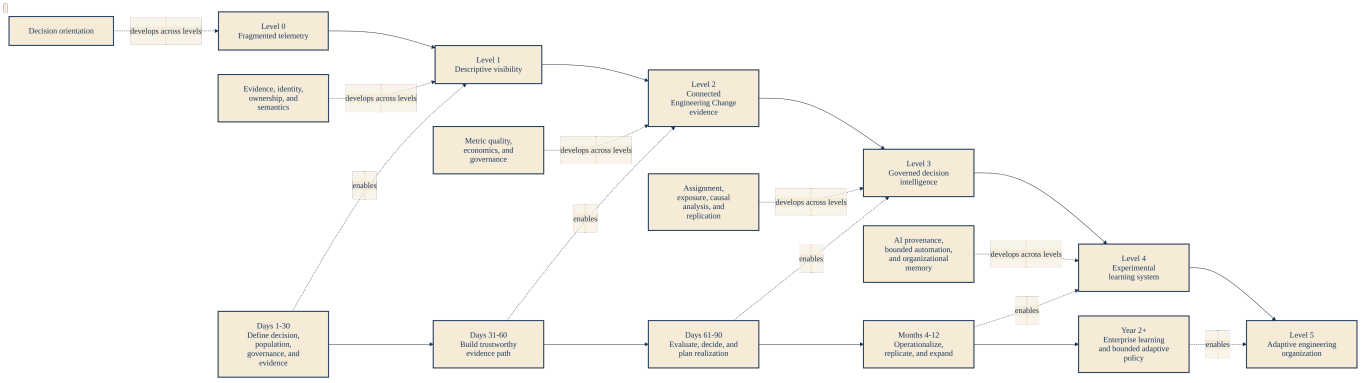
J.1. Assessment philosophy

A maturity model should not become a ranking exercise or a ceremonial scorecard.

Its purpose is to answer three questions:

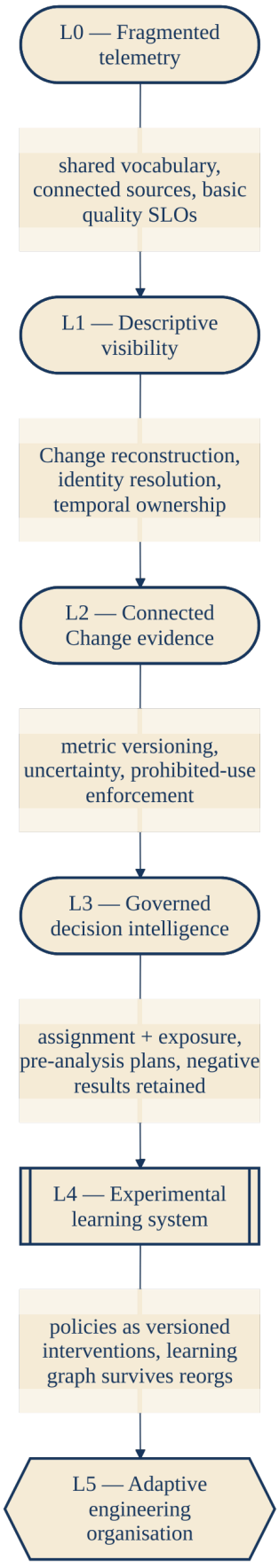
1. Which capabilities are currently strong enough to support consequential decisions?
2. Which missing or weak capabilities create the greatest decision risk?
3. What is the smallest next investment that closes a complete evidence-to-decision loop?

The assessment should evaluate demonstrated operating capability rather than the existence of policies, dashboards, or architecture diagrams.



J.2. Maturity levels

The six levels below describe what an Engineering Intelligence capability looks like as it accumulates. The transitions between levels are as important as the levels themselves — each ratchet closes a specific gap that was previously invisible, and reversing course is possible when a level's investment fails to hold. Read the diagram below as capabilities that must exist before the next level's practices can be sustained, not as a linear timeline.


L0 — Fragmented telemetry
shared vocabulary, connected sources, basic quality SLOs
L1 — Descriptive visibility
Change reconstruction, identity resolution, temporal ownership
L2 — Connected Change evidence
metric versioning, uncertainty, prohibited-use enforcement
L3 — Governed decision intelligence
assignment + exposure, pre-analysis plans, negative results retained
L4 — Experimental learning system
policies as versioned interventions, learning graph survives reorgs
L5 — Adaptive engineering organisation

J.2.1. Level 0: fragmented telemetry

The organization has source-specific dashboards and reports but no stable cross-system model.

In practice, this level looks like teams answering the same question three different ways depending on which tool they open first. Cross-team performance discussions rely on screenshots and story-time rather than shared evidence, and a leadership request for "the number" produces a two-week investigation whose result cannot be reproduced next quarter.

Typical characteristics:

- metrics are defined locally;
- pull requests, tickets, deployments, and incidents cannot be connected reliably;
- current ownership is applied to historical evidence;
- activity counts dominate;
- AI tools are evaluated through usage or anecdote;
- decisions are not linked to evidence versions;
- data quality is mostly implicit.

Decision risk is high because evidence is incomplete, inconsistent, and difficult to reproduce.

J.2.2. Level 1: descriptive visibility

The organization has several useful operational metrics and a basic shared vocabulary.

At this level a leader can open a dashboard and see how flow, quality, and reliability move over time without arguing about definitions. What remains missing is mechanism: nobody can yet say whether a change in the numbers reflects a change in the underlying system, a change in the workload, or a change in how the tools reported it.

Typical characteristics:

- major source systems are ingested;
- some team and repository metrics are standardized;
- basic lineage and data-quality monitoring exist;
- leadership can describe broad flow and reliability patterns;
- privacy and access rules are documented;
- measurement remains mostly descriptive.

The organization can identify patterns but cannot consistently explain mechanisms or causal effects.

J.2.3. Level 2: connected Engineering Change evidence

The organization can connect intent, artifacts, review, validation, deployment, release, and outcomes for a meaningful population.

Reaching this level is the first point at which *end-to-end* questions become answerable — "which changes actually reached production and stayed there," "which repository boundaries added coordination delay to changes that spanned them," "how much reviewer time did the top 5% of high-blast-radius changes consume." Reconstruction is imperfect and confidence bands appear on every joined dataset, but analysts stop reasoning about pull requests and start reasoning about Engineering Changes.

Typical characteristics:

- Engineering Change relationships are reconstructed;

- linkage confidence is visible;
- identity and ownership are temporal;
- source contracts and quality SLOs exist;
- metrics are computed from shared evidence;
- correction and replay are operational;
- decision products begin to replace dashboard browsing.

The organization can analyze end-to-end flow and consequence rather than isolated tool objects.

J.2.4. Level 3: governed decision intelligence

Metrics and analyzes are treated as versioned, governed products.

At Level 3 a metric change is a release, not a query edit. Consequential measures carry semantic versions, published assumptions, and uncertainty. Decision records reference the specific metric version that was used, so a decision made a year ago can be re-inspected under the same rules. Prohibited uses are enforceable, not aspirational, and dashboards that fail their governance criteria are retired rather than tolerated.

Typical characteristics:

- metric specifications include purpose, population, formula, quality, and prohibited use;
- uncertainty accompanies consequential measures;
- governance uses are registered;
- privacy, access, correction, and appeal operate in practice;
- economic evaluations distinguish capacity from savings;
- decision records include evidence, assumptions, and review dates;
- negative results are retained.

The organization can make defensible portfolio and operational decisions from governed evidence.

J.2.5. Level 4: experimental learning system

The organization routinely evaluates interventions causally.

At this level, "we rolled it out and things got better" is no longer treated as evidence that a tool worked. Interventions are launched with an assignment mechanism, an exposure signal, guardrails, and a pre-analysis plan; results carry intervals and are validated against realized long-term outcomes rather than immediate metric movement. Negative results are shipped alongside positive ones because they remove interventions that would otherwise linger in policy by default.

Typical characteristics:

- deterministic assignment and observed exposure are available;
- guardrails and stop rules are operational;
- randomized and quasi-experimental methods are used appropriately;
- pre-analysis plans and reproducibility packages are standard;
- findings are replicated across contexts;
- value realization is measured after adoption;
- intervention versions are tracked over time.

The organization can distinguish plausible improvement from measured causal effect.

J.2.6. Level 5: adaptive engineering organization

Engineering Intelligence is embedded in operating policy and organizational learning.

The distinguishing property of Level 5 is that the organization's own operating model is a subject of measurement. Policies, thresholds, review routing, capacity allocation, and even the metrics themselves are treated as interventions with lifecycles: they are versioned, evaluated for continued fit, and retired when the evidence changes. A reorganization does not reset the learning; the graph of decisions, evidence, and outcomes survives the org chart.

Typical characteristics:

- decisions, evidence, metrics, models, and outcomes form a persistent graph;
- portfolio investment adapts to evidence maturity and realized value;
- bounded automated policies operate with audit, appeal, and rollback;
- AI agents have identity, authority, and validation provenance;
- long-term capability and skill effects are monitored;
- learning is preserved across teams and reorganizations;
- the organization continuously revises assumptions and operating models.

The organization is capable of learning how it engineers, not merely measuring what it did.

J.3. Assessment dimensions

The maturity level should be scored separately for each dimension. A single average can hide critical weaknesses.

J.4. Decision orientation

J.4.1. Diagnostic questions

- Are major measurement efforts tied to named decisions?
- Does every consequential analysis have a decision owner?
- Are credible alternatives identified?
- Are decisions recorded with assumptions, conditions, and review dates?
- Can the organization show cases where evidence changed or stopped a decision?

J.4.2. Evidence of maturity

- active decision-question registry;
- bounded evidence requests;
- decision records;
- review-date compliance;
- examples of reversal or modification based on later evidence.

J.4.3. Common failure

Dashboards exist without a clear user, decision, or action.

J.5. Source evidence and data contracts

J.5.1. Diagnostic questions

- Are source semantics documented?
- Are event time and ingestion time both preserved?
- Can data be replayed?
- Are connector completeness and freshness measured?
- Are deletions and corrections propagated?
- Are source owners accountable for changes?

J.5.2. Evidence of maturity

- source inventory;
- versioned data contracts;
- immutable raw evidence;
- connector SLOs;
- backfill and replay records;
- schema-change review.

J.5.3. Common failure

Source APIs are treated as stable facts while tool semantics change silently.

J.6. Engineering Change semantics

J.6.1. Diagnostic questions

- Can the organization connect intent to production outcome?
- Are relationships explicit, deterministic, or probabilistic?
- Is linkage confidence visible?
- Are changes spanning multiple repositories or services represented?
- Can corrections be applied without deleting history?

J.6.2. Evidence of maturity

- canonical Engineering Change identifier;
- relationship model;
- confidence bands;
- reconstruction version;
- known linkage-coverage metrics;
- correction workflow.

J.6.3. Common failure

The pull request becomes the permanent analytical unit even when consequence spans multiple systems.

J.7. Identity, ownership, and organizational context

J.7.1. Diagnostic questions

- Are people, service accounts, bots, assistants, and agents distinguished?
- Is team and service ownership resolved as of event time?
- Are reorganizations preserved historically?
- Can analysts work with pseudonymous identities?
- Is re-identification restricted and audited?

J.7.2. Evidence of maturity

- canonical actor model;
- temporal ownership intervals;
- identity-confidence measures;
- isolated identity mapping;
- access logs;
- correction process.

J.7.3. Common failure

Current organizational structure is applied retrospectively, distorting historical comparison.

J.8. Metric quality

J.8.1. Diagnostic questions

- Does each metric identify its construct and decision purpose?
- Are unit, population, interval, formula, and aggregation explicit?
- Are missing data and bot activity handled deliberately?
- Are uncertainty, validity, and prohibited uses documented?
- Are metric versions visible in dashboards and APIs?

J.8.2. Evidence of maturity

- metric registry;
- validation records;
- quality gates;
- versioned computation;
- interpretation guidance;

- deprecation and retirement history.

J.8.3. Common failure

A familiar metric name conceals several incompatible implementations.

J.9. Human attention and work-system measurement

J.9.1. Diagnostic questions

- Can active effort be separated from elapsed delay?
- Are queues, interruption, rework, and validation burden measured?
- Does analysis account for work mix and complexity?
- Are local experience and qualitative evidence included?
- Are measures prohibited from unsupported individual ranking?

J.9.2. Evidence of maturity

- active-time methods with known limitations;
- queue-state instrumentation;
- review and validation sessions;
- experience sampling or practitioner interviews;
- work-mix adjustment;
- acceptable-use controls.

J.9.3. Common failure

Visible activity is treated as a substitute for cognitive work or productivity.

J.10. Experimentation and causal inference

J.10.1. Diagnostic questions

- Can treatment assignment be reproduced?
- Is actual exposure observed?
- Are primary outcomes and guardrails predeclared?
- Are clustering, spillover, and noncompliance considered?
- Are observational claims distinguished from causal claims?
- Are negative and null results preserved?

J.10.2. Evidence of maturity

- experiment registry;
- assignment and exposure service;

- readiness review;
- pre-analysis plans;
- reproducibility packages;
- replication records.

J.10.3. Common failure

Before-and-after movement is reported as causal improvement.

J.11. Economic evaluation

J.11.1. Diagnostic questions

- Are alternatives and counterfactuals explicit?
- Are implementation, validation, migration, support, and exit costs included?
- Is released capacity separated from savings?
- Are realization and fragmentation assumptions visible?
- Are bottlenecks and downstream constraints considered?
- Are forecast and actual value tracked?

J.11.2. Evidence of maturity

- decision-grade business cases;
- scenario and sensitivity analysis;
- break-even models;
- benefit-dependency map;
- realization ledger;
- renewal review.

J.11.3. Common failure

Minutes saved are multiplied by salary and presented as financial savings.

J.12. Architecture and complexity intelligence

J.12.1. Diagnostic questions

- Are repository, service, dependency, deployment, and ownership graphs connected?
- Can blast radius be estimated and calibrated?
- Are architectural boundaries linked to change effort and outcomes?
- Can coupling and ownership fragmentation be analyzed over time?
- Are architectural decisions linked to later evidence?

J.12.2. Evidence of maturity

- service and repository catalogue;
- dependency graph;
- change-coupling analysis;
- incident-change linkage;
- architecture decision records;
- longitudinal architecture metrics.

J.12.3. Common failure

Static code-complexity measures are interpreted as complete architecture quality.

J.13. Governance, privacy, and trust

J.13.1. Diagnostic questions

- Are permitted, restricted, and prohibited uses explicit?
- Is access purpose-bound and time-limited?
- Can practitioners inspect and correct attributed evidence?
- Are small groups and re-identification protected?
- Are vendor data use and model training constrained contractually?
- Are governance incidents detectable and handled?

J.13.2. Evidence of maturity

- governed-use registry;
- access recertification;
- practitioner notice;
- correction and appeal records;
- audit logs;
- incident and exception management.

J.13.3. Common failure

Governance exists as policy text but not as enforced architecture and operating practice.

J.14. AI and agent readiness

J.14.1. Diagnostic questions

- Are model, prompt, retrieval, tool, and configuration versions captured?
- Can human and machine contribution be attributed appropriately?
- Is validation burden measured?

- Are quality, security, cost, and capability guardrails present?
- Are agent authority and delegation boundaries explicit?
- Can a model or agent version be rolled back rapidly?

J.14.2. Evidence of maturity

- model and prompt registries;
- AI activity provenance;
- contribution and rework measures;
- continuous evaluation;
- delegation graph;
- bounded action policies;
- rollback tests.

J.14.3. Common failure

AI adoption, generated output, or acceptance is reported as productivity or value.

J.15. Platform and operational reliability

J.15.1. Diagnostic questions

- Are freshness, completeness, linkage, identity, and serving SLOs defined?
- Can source or metric incidents be detected quickly?
- Are environments isolated appropriately?
- Is cost tracked by source, pipeline, and product?
- Can the platform recover from corruption or regional failure?
- Are semantic changes run in shadow mode?

J.15.2. Evidence of maturity

- platform SLOs;
- quality dashboards;
- incident procedures;
- disaster-recovery tests;
- lineage impact analysis;
- workload and cost controls.

J.15.3. Common failure

The system measures engineering reliability without operating as a reliable system itself.

J.16. Organizational learning

J.16.1. Diagnostic questions

- Are decisions revisited after implementation?
- Are negative results and failed interventions preserved?
- Can learning survive team turnover and reorganization?
- Are practitioner observations incorporated?
- Are assumptions and boundary conditions recorded?
- Does portfolio allocation change based on realized evidence?

J.16.2. Evidence of maturity

- learning reviews;
- decision follow-up;
- evidence registry;
- replication agenda;
- value-realization review;
- retired or reversed policy records.

J.16.3. Common failure

Reports are produced, presented once, and then lost as organizational memory.

J.17. Scoring method

Score each dimension from 0 to 5 using demonstrated evidence.

Score	Meaning
0	Capability is absent or entirely informal.
1	Local examples exist, but semantics and ownership are inconsistent.
2	A repeatable process exists for a limited population or workflow.
3	The capability is governed and used for consequential decisions.
4	The capability is validated, monitored, and reproducible across several contexts.
5	The capability is adaptive, integrated, and improves through accumulated evidence.

A score should cite specific evidence and limitations.

Example:

```
assessment:
```

```yaml
  dimension: experimentation_and_causal_inference
  score: 2
  evidence:
    - repository_level_randomization_available_for_two_platforms
    - primary_metrics_versioned
  limitations:
    - treatment_exposure_not_consistently_recorded
    - no_standard_reproducibility_package
    - limited_guardrail_latency
  next_capability:
    - implement_exposure_service
    - standardize_readiness_and_analysis_records
```

J.18. Weighting by decision risk

Not every dimension has equal priority for every organization.

A tool-evaluation program may prioritize:

- experimentation;
- metric quality;
- AI provenance;
- economics;
- governance.

An architecture modernization program may prioritize:

- Engineering Change linkage;
- service and dependency catalogues;
- complexity and blast radius;
- longitudinal outcomes;
- portfolio economics.

A high-risk automated policy should require strong scores in evidence, governance, causal learning, platform reliability, and rollback before deployment.

J.19. Critical-capability floor

An overall maturity score should never compensate for a dangerous weakness.

Examples:

- strong analytics do not compensate for prohibited-use risk;
- excellent dashboards do not compensate for missing source lineage;
- randomized assignment does not compensate for unobserved exposure;
- high adoption does not compensate for absent quality guardrails;
- economic upside does not compensate for unacceptable security risk.

For consequential decisions, define minimum dimension scores before use.

J.20. Assessment process

A credible assessment should include:

1. executive and decision-owner interviews;
2. practitioner and domain interviews;
3. source and platform architecture review;
4. metric-definition review;
5. sample lineage walkthroughs;
6. experiment and decision record review;
7. governance and access review;
8. economic-model review;
9. live demonstration of correction and replay;
10. evidence-based scoring workshop.

The assessment should not rely solely on self-reported survey scores.

J.21. Assessment outputs

The assessment should produce:

- dimension scores with evidence;
- decision risks;
- capability dependencies;
- quick wins;
- critical controls;
- recommended pilot decision;
- first-90-day plan;
- 12- to 24-month roadmap;
- owners and investment assumptions;
- review date.

J.22. Selecting the first decision

The best first use case is consequential enough to matter but bounded enough to complete.

Strong candidates have:

- a named decision owner;
- two or more credible alternatives;
- meaningful cost or risk;
- an observable workflow;
- available source evidence;
- measurable outcomes within months;
- reversible or bounded implementation;

- practitioner interest;
- no need for individual performance use.

Examples include:

- selecting a pull-request review tool;
- improving CI queue capacity;
- changing review routing;
- evaluating a merge queue;
- prioritizing an architecture bottleneck;
- comparing an AI coding-assistant configuration;
- changing deployment policy for low-risk services.

J.23. Poor first decisions

Avoid beginning with:

- universal engineering productivity;
- individual performance scoring;
- enterprise-wide agent autonomy;
- a complete engineering ontology;
- every source system at once;
- long-term architecture value with no short-term outcomes;
- a decision with no accountable owner;
- a politically predetermined vendor selection.

J.24. First 30 days: define and bound

J.24.1. Outcomes

By day 30, the program should have:

- one consequential decision question;
- an executive decision owner;
- a cross-functional working group;
- a mapped Engineering Change lifecycle;
- a source and evidence inventory;
- initial governance and prohibited-use agreement;
- candidate primary metrics and guardrails;
- identified data-quality gaps;
- a written pilot charter.

J.24.2. Activities

1. Interview decision owners and practitioners.

2. Document alternatives and decision timing.
3. Map actors, artifacts, queues, validation, deployment, and outcomes.
4. Identify source systems and source owners.
5. Review privacy, security, and employment-use boundaries.
6. Select the smallest eligible population.
7. Draft metric and experiment specifications.
8. Establish a weekly evidence review.

J.24.3. Exit criteria

Do not proceed merely because instrumentation is available.

Proceed when:

- the decision and alternatives are clear;
- evidence requirements are bounded;
- owners accept the governance constraints;
- outcomes and guardrails are observable or can become observable;
- the pilot can produce a decision.

J.25. Days 31 to 60: build the evidence path

J.25.1. Outcomes

By day 60, the program should have:

- immutable ingestion for the required sources;
- canonical event mapping;
- actor, repository, service, and ownership references;
- Engineering Change linkage for the pilot population;
- linkage and source-quality reporting;
- draft governed metrics;
- correction and replay capability;
- a shadow decision product.

J.25.2. Activities

1. Implement or harden connectors.
2. Preserve event and ingestion time.
3. Establish source contracts.
4. Reconstruct Engineering Changes.
5. Calibrate linkage confidence on a labeled sample.
6. Implement metric formulas and quality gates.
7. Conduct practitioner validation.
8. Review identity and access controls.

9. Run historical backfills and reconciliation.

J.25.3. Exit criteria

Proceed when:

- source completeness and freshness are known;
- linkage quality is adequate for the declared use;
- metric semantics are reviewable;
- missingness is visible;
- corrections can propagate;
- practitioners recognize the workflow represented.

J.26. Days 61 to 90: evaluate and decide

J.26.1. Outcomes

By day 90, the program should have:

- an approved study or evaluation protocol;
- deterministic assignment or a defensible comparison design;
- observed treatment exposure;
- operational guardrails;
- an economic model with scenarios;
- a decision record template;
- an implementation and value-realization plan.

J.26.2. Activities

1. Conduct readiness and power review.
2. Register the pre-analysis plan.
3. Begin or complete the bounded evaluation.
4. Monitor data quality and safety.
5. Estimate direct cost, validation burden, and released capacity.
6. Identify realization conditions and bottlenecks.
7. Present effect size, uncertainty, limitations, and alternatives.
8. Record an adopt, modify, extend, reject, or reverse decision.

J.26.3. Exit criteria

The first 90 days succeed when the organization closes one complete loop from decision question to governed evidence and recorded action.

A large platform without a decision is not a successful first quarter.

J.27. Months 4 to 6: operationalize

Focus on:

- stabilizing source and metric contracts;
- expanding to adjacent populations;
- completing outcome follow-up;
- tracking value realization;
- implementing correction and appeal;
- documenting negative results;
- onboarding domain stewards;
- improving publication and decision products;
- reviewing platform cost and reliability.

The main objective is repeatability, not connector count.

J.28. Months 7 to 12: expand causal and economic capability

Focus on:

- reusable assignment and exposure services;
- standard experiment templates;
- quasi-experimental methods for nonrandomized rollouts;
- business-case and realization ledgers;
- architecture and blast-radius analysis;
- cross-team replication;
- evidence and decision registries;
- quarterly learning portfolio reviews.

By the end of year one, several decisions should have complete evidence-to-outcome histories.

J.29. Year 2: enterprise learning system

Potential priorities include:

- broader Engineering Change coverage;
- temporal organizational and service models;
- governed metric and model marketplaces;
- multi-region and regulated-domain controls;
- AI model, prompt, and agent provenance;
- continuous tool evaluation;
- architecture digital twins and simulations;
- bounded automated policy;
- cross-organization benchmarking under privacy controls.

These capabilities should be justified by demonstrated decision value from earlier stages.

J.30. Capability dependency map

Some capabilities must precede others.

Examples:

- causal analysis depends on stable outcomes, eligibility, assignment, and exposure;
- economic realization depends on measured effects and operational ownership;
- automated policy depends on metric quality, model validation, governance, and rollback;
- architecture intelligence depends on service, dependency, deployment, and ownership data;
- AI evaluation depends on model and prompt provenance plus human validation evidence;
- portfolio optimization depends on comparable business cases and realized outcomes.

Skipping dependencies creates false maturity.

J.31. Team and sponsorship model

A launch team should include:

- accountable executive sponsor;
- Engineering Intelligence product owner;
- data or platform engineering;
- measurement or causal expertise;
- developer-experience or workflow expertise;
- domain representatives;
- security and privacy;
- finance partner for economic evaluation;
- practitioner representatives.

A large permanent organization is not required initially. Clear ownership and access to these capabilities are required.

J.32. Funding model

Initial funding should be tied to a bounded decision portfolio.

A useful investment structure is:

- foundational platform funding for common evidence and governance;
- domain funding for source integration and semantic stewardship;
- decision-owner funding for studies and interventions;
- central measurement expertise shared across programs;
- explicit operational ownership after pilot completion.

The platform should not become an unfunded central reporting service with unlimited demand.

J.33. Adoption signals

Healthy adoption is visible when:

- teams request help framing decisions rather than requesting dashboards;
- metric definitions are challenged constructively;
- practitioners use correction workflows;
- decision owners accept uncertainty and bounded rollout;
- negative results affect investment;
- finance distinguishes capacity from savings;
- governance incidents are surfaced rather than hidden;
- platform priorities follow decision value;
- prior decisions are revisited using realized outcomes.

J.34. Warning signals

The program is drifting when:

- connector count is the primary progress metric;
- leaders ask for individual rankings;
- local teams rebuild metrics outside the registry;
- experiments lack exposure or guardrails;
- every result is positive;
- modeled time becomes reported savings;
- practitioners do not recognize the evidence model;
- governance review occurs only after deployment;
- old decisions are never revisited;
- AI usage is treated as AI value.

J.35. Maturity reassessment cadence

Reassess at least every six months during the first two years.

Also reassess after:

- major organizational change;
- new source-control or deployment platform;
- significant AI model or agent adoption;
- governance incident;
- metric or reconstruction redesign;
- expansion into regulated populations;
- transition from advisory to automated policy.

Each reassessment should compare promised and demonstrated capability.

J.36. Executive maturity summary template

```yaml
engineering_intelligence_maturity:
  assessment_date: 2026-11-01
  accountable_owner: vp_engineering_platform
  target_decisions:
    - enterprise_review_tool_selection
    - ci_capacity_investment
  dimensions:
    decision_orientation: 3
    source_evidence_and_contracts: 2
    engineering_change_semantics: 2
    identity_and_ownership: 2
    metric_quality: 2
    attention_measurement: 1
    experimentation: 2
    economics: 2
    architecture_intelligence: 1
    governance_and_privacy: 3
    ai_and_agent_readiness: 1
    platform_reliability: 2
    organizational_learning: 1
  critical_gaps:
    - treatment_exposure
    - capacity_realization_tracking
    - temporal_service_ownership
  next_90_day_outcome: complete_review_tool_decision_loop
  review_date: 2027-05-01
```

J.37. Roadmap decision rules

Prioritize work that:

- closes a complete decision loop;
- improves evidence quality for several decisions;
- reduces a critical governance or validity risk;
- enables causal evaluation;
- creates reusable semantics rather than vendor lock-in;
- supports correction and reproducibility;
- can demonstrate value within a bounded horizon.

Deprioritize work that:

- adds data without a purpose;
- optimizes dashboard aesthetics before semantics;
- builds universal abstractions before a working use case;
- automates decisions before governance maturity;
- requires unsupported individual surveillance;
- has no owner for realized value.

J.38. Assessment anti-patterns

J.38.1. Maturity theater

Scores are inflated to satisfy leadership expectations rather than grounded in evidence.

J.38.2. Average-score optimism

Strong dimensions conceal one critical weakness.

J.38.3. Tool-based maturity

Owning a warehouse, graph database, experimentation platform, or AI gateway is mistaken for operating capability.

J.38.4. Roadmap by connector

The program plans source integrations without specifying which decisions each enables.

J.38.5. Enterprise-wide first step

The first release attempts to model every team and workflow.

J.38.6. Permanent pilot team

The central team continues running local processes that should have operational owners.

J.38.7. Governance last

Privacy and acceptable-use constraints are introduced after detailed individual evidence is already broadly available.

J.38.8. No realization review

The program demonstrates metric movement but never determines whether the decision produced value.

J.39. Maturity and adoption checklist

1. Assess demonstrated capabilities by dimension rather than relying on one maturity score.
2. Cite concrete evidence, limitations, and decision risks for every score.
3. Define minimum capability floors for consequential uses.
4. Choose one bounded decision with alternatives, an owner, observable outcomes, and reversibility.
5. Complete a full evidence-to-decision loop in the first 90 days.
6. Build only the source, semantic, metric, governance, and analytical capabilities required for that loop.
7. Measure source quality, linkage confidence, uncertainty, and prohibited uses explicitly.
8. Separate released capacity, realized value, and financial savings.
9. Expand through replication and reuse rather than immediate enterprise breadth.
10. Reassess after major technical, organizational, model, or governance change.

11. Judge progress by decisions improved, harmful choices avoided, and learning retained.

Appendix K: Chapter Source Notes and Citation Map

This appendix maps the principal claims, constructs, and methods in each chapter to the *foundational and empirical sources collected in Appendix I. It provides a publication-ready citation plan without implying that every concept is established to the same degree. The map distinguishes mature foundations, practitioner syntheses, standards, and emerging research, and identifies where the book advances an original synthesis or proposes a research agenda.*

K.1. Citation conventions

The citation map follows several rules.

- Original papers and official standards are preferred over secondary summaries.
- A citation supports only the claim for which it is relevant; it does not endorse every conclusion in the source.
- Practitioner books are used for operating models and synthesis, not as substitutes for causal evidence.
- Preprints and recent AI studies are labeled as emerging evidence.
- The Engineering Change reference model, attention economics, integrated platform design, and several agent-governance concepts are original syntheses in this book and should not be presented as established external standards.
- Time-sensitive AI research and standards should be reverified before each publication edition.

K.2. Chapter 1 — Introduction

K.2.1. Established foundations

The argument that developer productivity is multidimensional is supported by the SPACE framework and the DevEx framework. See [forsgren2021-space] and [noda2023-devex].

The discussion of software delivery performance and organizational capability builds on the research synthesized in [forsgren2018] and the operating practices described in [kim2021-devops] and [humble2010].

The warning against treating easily counted activity as productivity is consistent with measurement-validity principles in [messick1995], indicator corruption in [campbell1979], and Goodhart's Law in [goodhart1984].

The socio-technical framing is grounded in Conway's organizational observation, socio-technical congruence, and distributed coordination research. See [conway1968], [cataldo2006], and [herbsleb2007].

The discussion of human attention and interruption is informed by cognitive-load theory and interruption research. See [sweller1988] and [mark2008].

The discussion of experimentation draws on the potential-outcomes framework, experimental design, and practical controlled-experiment methods. See [rubin1974], [holland1986], [shadish2002], and [kohavi2020].

K.2.2. Original synthesis

The Engineering Change as the principal unit of consequence, the explicit treatment of AI as an engineering participant, and the integration of attention, economics, architecture, causal inference, and governance form the book's original synthesis.

K.3. Chapter 2 — Engineering Economics

K.3.1. Established foundations

The distinction between technology adoption and realized productivity is supported by the productivity-paradox literature. See [solow1987], [brynjolfsson1993], and [brynjolfsson2017].

The treatment of cost-benefit analysis, discounting, risk, and sensitivity follows the methods summarized in [boardman2018].

The system-level emphasis on constraints and bottlenecks is informed by [goldratt1984] and the operational treatment of flow and capacity in [hopp2011].

The distinction between gross local efficiency and end-to-end value is consistent with queueing and operations research, particularly [little1961], [kingman1961], and [hopp2011].

K.3.2. Original synthesis

The capacity-realization factor, fragmentation discount, validation-burden accounting, and explicit taxonomy separating cash savings, avoided cost, released capacity, modeled outcome value, and option value are operational constructs developed for Engineering Intelligence.

K.4. Chapter 3 — Engineering Change

K.4.1. Established foundations

The modularity and information-hiding basis for reasoning about change boundaries is grounded in [parnas1972].

The relationship between organizational communication and system structure is supported by [conway1968] and [cataldo2006].

The provenance-oriented entity, activity, and agent model is compatible with the W3C PROV data model in [moreau2013].

The event-log, replay, temporal, and data-intensive architectural principles draw on [kleppmann2017] and [akidau2018].

K.4.2. Original synthesis

The Engineering Change lifecycle, graph vocabulary, confidence-aware reconstruction model, multi-repository aggregation, and unit-of-consequence framing are original reference-model contributions of this book.

K.5. Chapter 4 — Human Attention

K.5.1. Established foundations

Cognitive-load theory provides the basis for treating working cognitive capacity as constrained. See [sweller1988].

The cost of interruption and context switching is supported by [mark2008].

Developer perceptions of productive and unproductive work are examined in [meyer2014].

The relationship between developer affect, progress, and flow is informed by [muller2019] and the developer-experience

frameworks in [fagerholm2012] and [noda2023-devex].

K.5.2. Original synthesis

The four-part attention model—production, verification, coordination, and recovery effort—and the concepts of attention amplification, attention yield, cognitive offload, and cognitive reload are book-specific operational constructs.

K.6. Chapter 5 — Measuring Engineering Systems

K.6.1. Established foundations

The treatment of measurement scales and the limitations of arithmetic operations on different scale types is grounded in [stevens1946].

Reliability and internal consistency are informed by [cronbach1951].

Validity as evidence supporting a particular interpretation and use follows [messick1995].

The broader treatment of measurement design is supported by [hand2004].

Indicator corruption and behavioral response to targets are grounded in [campbell1979] and [goodhart1984].

The multidimensional nature of engineering productivity is supported by [forsgren2021-space] and [noda2023-devex].

K.6.2. Original synthesis

The metric specification lifecycle, policy-grade metric concept, explicit prohibited-use field, quality-state publication contract, and integration of measurement validity with governance are original operational extensions.

K.7. Chapter 6 — Experimentation and Causal Inference

K.7.1. Established foundations

The potential-outcomes formulation follows [rubin1974], [holland1986], [imbens2015], and [hernan2020].

Structural causal reasoning and causal graphs are grounded in [pearl2009].

Randomized and quasi-experimental design principles are supported by [shadish2002], [angrist2009], and [kohavi2020].

Preregistration and separation of confirmatory and exploratory analysis are supported by [nosek2018].

K.7.2. Original synthesis

The Engineering Intelligence experiment protocol combines software delivery events, Engineering Change assignment, actual tool exposure, model and prompt versioning, guardrails, economic outcomes, and value-realization follow-up into one operational study model.

K.8. Chapter 7 — AI-Assisted Engineering

K.8.1. Established and emerging evidence

Controlled task-level evidence on AI coding-assistant completion time is represented by [peng2023].

Usability, comprehension, debugging, and trust challenges are examined in [vaithilingam2022].

Programmer interaction patterns with code-generating models are studied in [barke2023].

Repository-level software-agent benchmarking is represented by [jimenez2024].

Broader GenAI productivity perceptions across several dimensions are discussed in the emerging evidence represented by [afroz2025].

AI risk-management principles are supported by [nist2023-airmf], [nist2024-genai], 001, and 894.

K.8.2. Caution

These studies do not establish universal effects across organizations, model versions, engineering tasks, or production outcomes. Claims should remain bounded to population, task, treatment, comparator, and observation period.

K.8.3. Original synthesis

Net attention effect, validation burden, accepted contribution share, machine-participation provenance, delegation depth, autonomous-action reversal, model monoculture, and continuous Engineering Change evaluation are original or extended constructs in this book.

K.9. Chapter 8 — Enterprise Instrumentation

K.9.1. Established foundations

Event-time, late-data, watermark, and stream-processing concepts draw on [akidau2018].

Event logs, replay, consistency, batch and stream architecture, and data-system tradeoffs draw on [kleppmann2017].

Provenance semantics are aligned with [moreau2013].

Reliability objectives, operational toil, and service-level thinking are informed by [beyer2016] and [beyer2018].

Privacy-risk management is informed by [nist2020-privacy]; cybersecurity governance is informed by [nist2024-csf].

K.9.2. Original synthesis

The source-to-canonical Engineering Change instrumentation model, confidence-aware reconstruction quality, metric-quality SLOs, and purpose-bound evidence-serving architecture are original platform integrations.

K.10. Chapter 9 — Complexity and Blast Radius

K.10.1. Established foundations

Modular decomposition and information hiding are grounded in [parnas1972].

The relationship between organizational and technical structure is supported by [conway1968], [cataldo2006], and

[herbsleb2007].

Team cognitive load and interaction modes are informed by [skelton2019].

K.10.2. Original synthesis

The multidimensional blast-radius model combining dependency reach, service and repository span, ownership entropy, validation surface, data and financial consequence, and incident evidence is a book-specific framework.

K.11. Chapter 10 — The Engineering Intelligence Platform

K.11.1. Established foundations

The architectural treatment of durable logs, stream and batch processing, replay, and data-system tradeoffs draws on [kleppmann2017] and [akidau2018].

Provenance modeling is supported by [moreau2013].

Service-level objectives and operational platform reliability are informed by [beyer2016] and [beyer2018].

Privacy, cybersecurity, and AI risk management are informed by [nist2020-privacy], [nist2024-csf], [nist2023-airmf], and [nist2024-genai].

K.11.2. Original synthesis

The connected evidence, metric, model, experiment, and decision registries; purpose-bound APIs; Engineering Change reconstruction service; and evidence-to-decision architecture are integrated reference designs developed in this book.

K.12. Chapter 11 — Engineering Metrics Catalogue

K.12.1. Established foundations

Delivery-performance measures build on the research synthesized in [forsgren2018].

Multidimensional developer-productivity and experience measures are informed by [forsgren2021-space] and [noda2023-devex].

Queue, throughput, work-in-progress, and elapsed-time relationships are grounded in [little1961], [kingman1961], and [hopp2011].

Precision, recall, and ranking measures are supported by [manning2008] and [jarvelin2002].

Measurement validity and target distortion are informed by [messick1995], [campbell1979], and [goodhart1984].

K.12.2. Original synthesis

The catalogue’s attention, economics, AI, architecture, linkage, governance, and value-realization metrics extend established measurement ideas into a unified Engineering Intelligence system.

K.13. Chapter 12 — Experiment Playbooks

K.13.1. Established foundations

The playbooks are grounded in experimental and quasi-experimental methods from [shadish2002], [angrist2009], [imbens2015], [hernan2020], and [kohavi2020].

Ranking evaluation uses [jarvelin2002] and [manning2008].

Preregistration and confirmatory-analysis discipline are informed by [nosek2018].

K.13.2. Original synthesis

The vendor, AI reviewer, coding assistant, CI, architecture, and policy playbooks adapt causal methods to Engineering Change evidence, machine provenance, human validation burden, and enterprise guardrails.

K.14. Chapter 13 — Governance, Ethics, and Adoption

K.14.1. Established foundations

Professional obligations concerning harm, fairness, privacy, honesty, competence, and accountability are informed by [acm2018-ethics].

AI risk governance is supported by [nist2023-airmf], [nist2024-genai], 001, and 894.

Privacy-risk management is informed by [nist2020-privacy], and cybersecurity governance by [nist2024-csf].

Measurement misuse and behavioral distortion are supported by [campbell1979], [goodhart1984], and [messick1995].

K.14.2. Original synthesis

The use-classification model, presumption against individual employment use, purpose registration, metric-use review, generated-explanation controls, and governance-incident model are operational standards developed in this book.

K.15. Chapter 14 — Portfolio and Executive Decision-Making

K.15.1. Established foundations

Cost-benefit and uncertainty methods draw on [boardman2018].

Technology-productivity realization and complementary organizational investment are informed by [brynjolfsson1993] and [brynjolfsson2017].

Constraint and flow reasoning is informed by [goldratt1984] and [hopp2011].

Organizational learning and systems thinking are supported by [argyris1978] and [senge1990].

K.15.2. Original synthesis

The evidence-maturity portfolio, decision-record system, learning investment portfolio, value-realization ledger, and executive distinction between activity, capacity, and realized value are book-specific integrations.

K.16. Chapter 15 — Implementation and Rollout

K.16.1. Established foundations

Continuous delivery and incremental operational change are informed by [humble2010] and [kim2021-devops].

Reliability objectives and operational ownership are supported by [beyer2016] and [beyer2018].

Learning-organization and double-loop concepts are grounded in [argyris1978] and [senge1990].

K.16.2. Original synthesis

The decision-first rollout sequence, maturity floors, complete evidence-to-decision loop, and first-90-day Engineering Intelligence implementation are operational methods developed in this book.

K.17. Chapter 16 — The Science of Engineering Intelligence

K.17.1. Established foundations

Measurement validity and reliability draw on [stevens1946], [cronbach1951], [messick1995], and [hand2004].

Causal inference draws on [rubin1974], [holland1986], [pearl2009], [imbens2015], and [hernan2020].

Preregistration and reproducibility are informed by [nosek2018] and [open-science2015].

Reusable evidence and stewardship principles are informed by [wilkinson2016].

K.17.2. Original synthesis

Engineering Intelligence as a distinct scientific discipline, its unit hierarchy, evidence maturity model, open evidence graph, and combined research agenda are proposed in this book.

K.18. Chapter 17 — The Learning Engineering Organization

K.18.1. Established foundations

Single-loop and double-loop learning are grounded in [argyris1978].

Systems thinking and learning-organization practices are informed by [senge1990].

High-reliability organizing and attention to weak signals are supported by [weick2007].

Systems-oriented incident learning is informed by [dekker2014] and resilience engineering by [woods2015].

Delivery feedback and learning practices are informed by [kim2021-devops] and [beyer2016].

K.18.2. Original synthesis

Learning velocity, capability preservation, evidence memory, decision follow-up, negative-result retention, and the Engineering Intelligence function are integrated organizational constructs developed in this book.

K.19. Chapter 18 — The Future of Engineering Intelligence

K.19.1. Established foundations

AI risk and management-system concepts are informed by [nist2023-airmf], [nist2024-genai], 001, and 894.

General-purpose-technology adoption and delayed productivity effects are informed by [brynjolfsson2017].

Resilience and adaptive capacity are informed by [woods2015] and [weick2007].

Provenance and agent-activity representation are compatible with [moreau2013].

K.19.2. Emerging and original material

The treatment of intent as a first-class artifact, agent economic participation, delegation graphs, machine-speed governance, software-production digital twins, synthetic evidence risk, executable architecture policy, and bounded closed-loop engineering automation is forward-looking synthesis and research agenda, not established consensus.

K.20. Appendix source relationships

K.20.1. Appendix A — Engineering Change Reference Model

Conceptually compatible with provenance entities, activities, and agents in [moreau2013], but defines a domain-specific model and relationship vocabulary.

K.20.2. Appendix B — Metric Specification Standard

Grounded in measurement validity and reliability from [stevens1946], [cronbach1951], [messick1995], and [hand2004], with governance extensions informed by [campbell1979] and [goodhart1984].

K.20.3. Appendix C — Experiment and Causal Study Protocol

Grounded in [shadish2002], [imbens2015], [hernan2020], [angrist2009], [kohavi2020], and [nosek2018].

K.20.4. Appendix D — Governance, Privacy, and Acceptable Use Standard

Informed by [acm2018-ethics], [nist2020-privacy], [nist2024-csf], [nist2023-airmf], [nist2024-genai], 001, and 894.

K.20.5. Appendix E — Economic Evaluation and Business Case Standard

Grounded in [boardman2018], [brynjolfsson1993], [brynjolfsson2017], [little1961], [kingman1961], and [hopp2011].

K.20.6. Appendix F — Technical Reference Architecture

Informed by [moreau2013], [kleppmann2017], [akidau2018], [beyer2016], and [beyer2018].

K.20.7. Appendix G — Operating Templates and Records

Operationalizes the standards and original synthesis developed throughout the book.

K.20.8. Appendix H — Glossary and Controlled Vocabulary

Defines the book's controlled semantics; external concepts retain their established meaning where cited.

K.20.9. Appendix I — Evidence Base and Bibliography

Contains full source details and evidence classifications.

K.20.10. Appendix J — Maturity Assessment and Adoption Roadmap

Combines organizational learning from [argyris1978] and [senge1990], delivery and reliability practices from [kim2021-devops] and [beyer2016], and the book's original maturity framework.

K.21. Citation integration roadmap

The next publication pass should:

1. add inline cross-references from major empirical and methodological claims to the bibliography anchors;
2. add source notes to figures whose structure derives from published frameworks;
3. distinguish direct empirical support from analogy or synthesis;
4. verify every DOI and official publication URL;
5. remove or replace sources that cannot be verified from an authoritative publication record;
6. add conflicting or null evidence where material;
7. record verification dates for web-only standards and institutional publications;
8. generate a citation-coverage report during the publication build.

K.22. Citation map checklist

1. Cite primary and official sources wherever available.
2. Bound claims to the population and method of the cited evidence.
3. Label practitioner synthesis, standards, and preprints accurately.
4. Do not imply that a framework or standard proves empirical effectiveness.
5. Identify original Engineering Intelligence concepts explicitly.
6. Preserve unresolved questions and conflicting evidence.
7. Reverify recent AI studies and standards before each edition.
8. Treat citation coverage as a quality attribute of the publication.

Index

A

acceptable use, 247
agent task loop, 129
attention residue, 75
attention scarcity, 37
autonomous engineering, 345
autonomy tier, 129
average treatment effect, 109

B

benchmark, 299
blast radius, 171

C

capacity allocation, 265
carrying cost, 37
causal graph, 109
cognitive load, 75
cognitive offload, 129
Cognitive Offload Index, 75
cognitive reload, 75
composite metric, 91
confounder, 109
construct validity, 91
context switch, 75
cost of delay, 37
cost of quality, 37

D

data minimization, 247
decision workbench, 191
difference-in-differences, 109

E

Engineering Change, 57
Engineering Intelligence, 23
event contract, 149
evidence explorer, 191
experiment playbook, 229

F

feature store, 191

G

guardrail, 229
guardrail metric, 91

I

identity resolution, 149
immutable log, 149
intrinsic complexity, 171

L

learning loop, 323
Little’s Law, 37

M

measurement, 23
metric API, 191
metric bundle, 209
metric lifecycle, 209
metric specification, 209
metric taxonomy, 91
metric versioning, 91
minimum detectable effect, 109
multi-agent, 345

O

opportunity cost, 265
option value, 265

P

phased rollout, 283
pilot selection, 283
portfolio, 265
potential outcomes, 109
provenance, 57
purpose limitation, 247

Q

queueing, 37

R

reconstruction, 57
reliability, 91
replication, 299
repository topology, 171
reproducibility, 299
retrospection, 323
review burden, 75

S

sensitivity, 91
socio-technical system, 23
statistical power, 109

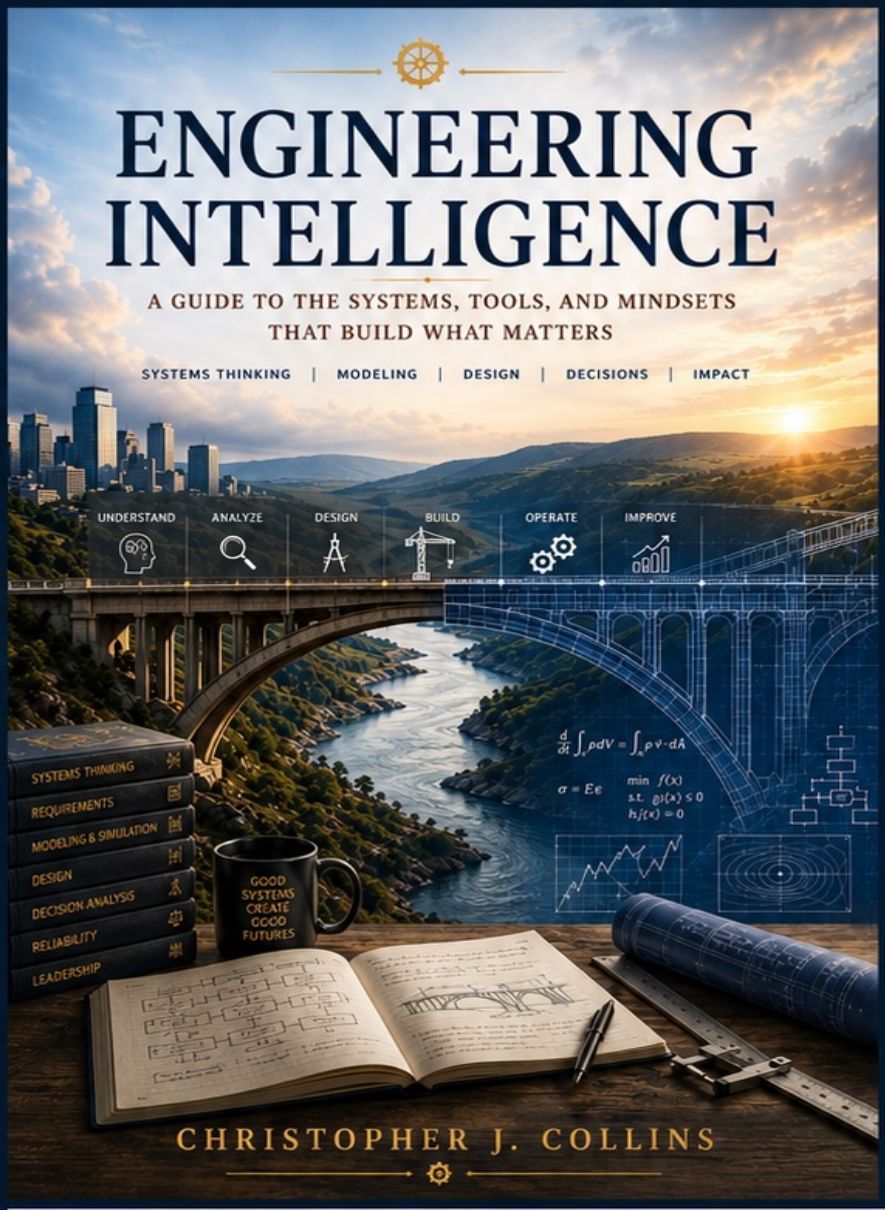
T

temporal ownership, 149
temporal property graph, 57
topological complexity, 171

treatment exposure, 229

U

use classification, 247



BUILD INTELLIGENCE. BUILD WHAT MATTERS.

Engineering today is more complex—and more consequential—than ever before. Success belongs to those who can think in systems, model what matters, design with purpose, and make decisions that create lasting impact.

Engineering Intelligence is a comprehensive guide to the principles, practices, and tools that empower engineers, builders, and leaders to solve hard problems and deliver outcomes that endure. From systems thinking and modeling to decision-making, operations, and continuous improvement, this book connects timeless engineering wisdom with modern methods and real-world application.

INSIDE YOU WILL DISCOVER:

- How to think in systems and see the big picture
- How to model, simulate, and validate complex systems
- How to design solutions that are robust, reliable, and scalable
- How to operate, monitor, and improve what you build
- How to make decisions with clarity, data, and judgment
- How to lead engineering efforts that create lasting impact

Whether you are an engineer, architect, product builder, or technical leader, this book will help you build the capabilities—and the mindset—to turn ideas into intelligent systems that make the world better.

GOOD SYSTEMS CREATE GOOD FUTURES.

ENGINEERING INTELLIGENCE
is a publication of OrionWorks
orion.works

ISBN 978-1-963412-00-1
90000
9 781963 412001

ENGINEERING INTELLIGENCE

A GUIDE TO THE SYSTEMS, TOOLS, AND MINDSETS THAT BUILD WHAT MATTERS

SYSTEMS THINKING | MODELING | DESIGN | DECISIONS | IMPACT



CHRISTOPHER J. COLLINS

TO THE READER

Engineering is more than building things. It is the disciplined application of knowledge, creativity, and judgment to solve problems that matter.

The systems you build shape how we live, work, and thrive. Build them with clarity, intelligence, and purpose.

May this book help you think deeper, design better, and create a positive impact that lasts.

C. J. Collins

CHRISTOPHER J. COLLINS