Systems Engineering

from First Principles

Part Two: Derivations, Questions and Failures

Matt Polly

DERIVED PRINCIPLES

These are not first principles. They are consequences that follow when the subject of the engineering is a system — what happens when the S-principles and E-principles meet. Each is derivable from those foundations, but important enough to state explicitly because violating any of them produces a characteristic failure.

D1. Pay attention to the interactions, not the parts. Systems engineering addresses behaviors that emerge from the interactions between parts, not in the parts themselves. Emergent behavior requires more information to describe than the set of parts separately. In most cases this behavior is derivable in principle but computationally overwhelming in practice. The engineering consequence: model and constrain the interactions with at least the same rigor as the parts.

D2. Decomposition is necessary and lossy. Reasoning requires reduction, but any decomposition is a model, not a fact. The resulting components relate outside their declared contracts through implicit dependencies — timing, state, coupling, shared resources. These hidden interactions are not eliminated by the decomposition; they merely go unmanaged, or worse, unnoticed. The engineering consequence: reference the same "shared temporal context" in each component.

D3. System state is history-dependent. Interactions unfold over time and their sequence matters. A system is not simply a configuration — it is a configuration plus a trajectory. Models calibrated at one state may fail at another even when nothing in the environment has changed. Behavior observed during testing may not be behavior exhibited in operation. The engineering consequence: track system state as a series of events.

D4. Some decisions are asymmetrically irreversible. Architectural commitments, technology selections, and interface contracts vary enormously in how costly they are to reverse. The cost of reversal is itself path-dependent and often invisible until attempted. Systems engineering must distinguish between decisions that can be deferred and decisions that foreclose future options. The engineering consequence: identify asymmetric decision points, and mitigate risk through late binding and narrow interfaces.

D5. Verification does not compose across levels. Systems nest without fixed depth. Behavior that is elementary at one level is composite at another. A system whose

components all pass their tests can still fail as a system, and a system that passes integration testing can still fail in deployment. Green tests are evidence bounded by the level at which they were conducted.

D6. Requirements are negotiations, not discoveries. Decomposition creates constituencies with competing interests. Optimizing one component can de-optimize another. Systems engineering makes the cost of each decision legible so that tradeoffs are contractual rather than accidental. Which preferences are treated as constraints versus tradeoffs is a contextual and political decision, not a technical one.

D7. Insufficient knowledge is the baseline. Engineering knowledge is quantitative, conditional, and bounded. The domain may shift beneath you; verification doesn't compose across levels. Systems engineering responds through two distinct mechanisms: *learning* (iteration, testing, feedback) to reduce uncertainty where possible, and *buffering* (redundancy, margin, graceful degradation) to absorb uncertainty that remains. These are complementary — a system that only learns is fragile to surprises, and a system that only buffers never improves its model.

D8. Deployed artifacts co-evolve with their environments. Engineered systems integrate into larger supersystems, mutually altering interactions, dependencies, and available states. Each design is both enabler and limiter for subsequent configurations. The system becomes part of the world that future engineering must contend with.

D9. Stability is an active, ongoing cost. Because systems are dissipative and knowledge is incomplete, a system is never done. Maintenance, feedback, adaptation — these aren't aftercare. They are the ongoing provision of energy and information to arrest the drift toward entropy. A system that isn't being actively sustained is a system in the process of failing.

SIX QUESTIONS

These six questions are generalized from the first principles of part one. They are designed to be **fractal** — they work at every level. Ask them about the system. Ask them about a subsystem, an interface, a requirement, a team, a single design decision. Same six questions, different answers at each level.

Q1. *What is this for?* Reveals function, validates decomposition, traces parts to purpose. Asked of a part, does it earn its place? Asked of a process or meeting, is the organization spending energy on coherence or ritual? Asked of a requirement, it points to a stakeholder preference. If you can't answer this clearly and traceably, that thing may be wasteful or a risk you don't understand.

Q2. *What depends on what?* Maps interactions, information flows, boundaries, nesting, and organizational coupling. Every dependency is a link in a chain that must endure the stress of the world. These links are the channels through which change propagates or failure cascades. Hidden properties of dependencies are often the source of integration failure.

Q3. *What is the agreement?* Asks whether alignment exists — among stakeholders, between teams, across interfaces. If you can't state the agreement, there isn't one. Covers interface contracts, requirements as stakeholder agreements, and the political dimension of whose preferences prevail. Unstated agreements become accidental tradeoffs.

Q4. *What would break this?* Forces adversarial thinking. Think about quantity, quality, timing, and context. Where might resource-exhaustion occur? Do we validate all they way up to our business rules? How do you handle missing data or out-of-order events? Find the failure modes before they find you.

Q5. *How can I make this more resilient?* Responds to fragility, but goes beyond it to include sustainment. What energy and attention are necessary to keep this coherent over time? Includes robustness — are we prepared for surprise? Includes recovery — what happens after failure? Nothing survives without investment.

Q6. *How do I know?* The meta-question. Challenges the quality of knowledge behind every other answer. Probes for bounded models, perishable assumptions, unverified claims, and confidence that hasn't been earned. Can be asked of every answer to every other question. It is the question that enforces honesty across all the others.

FAILURE MODES

Each derived principle, when violated, produces a characteristic failure. These are not random — they are predictable consequences of ignoring what is true about systems and engineering. The six questions are designed to reveal them before they reveal themselves.

F1. → **The Integration Failure.** Over-reliance on part-level analysis leaves interactions under-modeled. The signature: every component passes its tests and the system doesn't work. Surprises concentrate at the interactions that part-level testing couldn't see. *Caught by: What depends on what? What would break this?*

F2. → **The Interface Failure.** Treating decomposition as objective truth rather than a chosen model means implicit dependencies go unmanaged. They surface as integration shocks, security vulnerabilities, or failures at the seams between teams. Because the contract said those interactions didn't exist, nobody was watching. *Caught by: What is the agreement? What depends on what?*

F3. → **The Trajectory Failure.** Modeling states as static rather than path-dependent. The system is validated in one state but deploys or drifts into another. It doesn't fail because it is wrong — it fails because the system state moved and the models and tests didn't. *Caught by: What would break this? How do I know?*

F4. → **The Commitment Failure.** Treating all decisions as equally reversible means high-cost commitments get made prematurely. The architecture gets locked before requirements are stable. The technology gets selected before constraints are understood. By the time reversal is attempted, the cost exceeds the budget. *Caught by: What is the agreement? How do I know?*

F5. → **The Verification Illusion.** Expecting verification to propagate across levels leads to systems that work in isolation but fail as a whole. Green tests are evidence, not proof. Emergent behavior at each level is invisible to the level below, and the cost of discovering this late is refactoring and re-integration. *Caught by: How do I know? What depends on what?*

F6. → **The Accidental Tradeoff.** Constituencies treat their preferences as requirements. Tradeoffs get made implicitly — without awareness or intent — rather than through explicit negotiation. The system reflects a set of tradeoffs that nobody knew had been made. *Caught by: What is the agreement? What is this for?*

F7. → The Model Confidence Failure. Assuming completeness without building in tolerance exposes the system to unbuffered uncertainty. Edge cases become disasters not because they were unforeseeable but because the system was over-confident. *Caught by: How can I make this more resilient?*

F8. → The Supersystem Surprise. Designing artifacts in isolation ignores their co-evolutionary impact. Dependencies form that weren't designed. Adjacent systems couple in ways that limit future options. The system becomes a legacy trap — not because it stopped working, but because the ecosystem solidified. *Caught by: What depends on what?*

F9. → The Delivery Illusion. Treating stability as passive after deployment withdraws the energy required to hold the system in its designed state. Maintenance is underfunded, feedback channels atrophy, and the system drifts. By the time the failure is visible, dissolution has been underway for years. *Caught by: How can I make this more resilient? What would break this?*