The Discipline of Decision Design

How Engineers Design Systems That Survive Reality

Viktor Jevdokimov, Vilnius, Lithuania

Table of contents

INTRODUCTION & QUICK START

System Design Lens (SDL) — The Discipline of Decision Design

How Engineers Design Systems That Survive Reality

> ⚠️ **ATTENTION: You may be viewing a downloaded version.**
> The living, latest version of this documentation is always available online: SDL Official Documentation

This book is a manual for **replicating a reasoning discipline**—a way to analyze, compare, design, and validate systems and frameworks **without AI**.

It is not a catalog of frameworks, and it is not a "best practices" guide. It treats systems as **decision machines**: engineered constructs that reduce a specific, observable failure by constraining thinking and producing inspectable artifacts.

What This Book Is

A practical doctrine for designing, selecting, and governing systems such as frameworks, methods, operating models, and decision processes.

What This Book Is Not

- Not a creativity book
- Not a productivity book
- Not a leadership motivation book
- Not "framework literacy" for its own sake

If you are looking for a new framework to adopt, this book will mostly tell you when **not** to adopt one.

Who This Book Is For (Non-Negotiable)

This book exists to prevent a specific failure:

Engineers, product leaders, and strategists adopt systems (frameworks, methods, models) without understanding what decisions they optimize, how they break, or when they should not be used.

This failure shows up as:

- "Alignment" meetings that create more disagreement
- Process changes that add ritual but reduce throughput
- Frameworks used as justification instead of diagnosis
- Systems that scale vocabulary but not coordination
- Teams optimizing local metrics while global outcomes degrade

Audience

You will get value from this book if you:

- Design or adopt how work is done (engineering, product, ops, strategy, leadership)
- Facilitate planning, prioritization, discovery, or delivery decisions
- Need to evaluate "should we use X?" (OKRs, DDD, Shape Up, SAFe, ITIL, etc.)
- Want to invent a lightweight system that is *valid*, not fashionable

Non-Negotiable Reading Contract

You should not proceed through this book unless you accept these rules:

- A system must be anchored in an **observable failure** (not a vibe).
- A system must optimize a **specific decision** (not general "clarity").
- A system must produce at least one **inspectable artifact**.
- A system must enforce at least one **constraint** (or it is just vocabulary).
- A system must include a **misuse model** (how it breaks when misapplied).

High-Level Structure (Yes, Your Instinct Is Correct)

Your instinct to organize this as **Introduction / Theory / Practice / Reference** is right, with one adjustment:

The sections must be designed to prevent the most common misuse: treating the book itself like a framework to adopt.

So the book uses **four layers**, each with a distinct job:

Layer	Purpose	Capability Gained
Orientation	Prevent misuse from page 1	Know when *not* to proceed
Design Theory	Install mental primitives	See systems as decision machines
Operational Practice	Train execution	Perform analysis & invention manually
Reference Doctrine	Enable repeatability	Apply under pressure, solo or in groups

How to Read This Book

- If you are about to adopt a new framework: start with **Orientation**, then use **Operational Practice** to evaluate it.
- If you are diagnosing recurring failures: start with **Design Theory**, then use **Operational Practice** to decompose your current system.
- If you are inventing something new: go directly to the **System Contract** in **Operational Practice** and use **Design Theory** only as needed.
- If you already know what you're doing but need consistency: treat **Reference Doctrine** as your checklist library.

The Book's Operating Style

This book is written to be usable:

- **Solo** (engineer/lead reasoning at a whiteboard)
- **In a team** (facilitated session)
- **In leadership contexts** (governance and system review)

The emphasis is on **inspectable outputs** and **constraints that prevent self-deception**, not persuasion.

Quick Start Use Case

If you only do one thing after reading this page:

1. Write a 3–5 sentence **Observable Failure Statement** describing what is going wrong right now.
2. Name the **decision** you want to make safer/faster (priority, scope, sequencing, ownership, investment, diagnosis, repair).
3. Identify one **artifact** that would make that decision inspectable (map, table, score, contract, rule set).
4. Add one **constraint** that forbids a common failure pattern (timebox, scope cap, authority boundary).

If you can't do step (1), you are not ready to design or adopt a system—yet.

Why This Works Without AI

This book doesn't rely on intelligence. It relies on **discipline**.

The core logic is a set of forced moves that humans can execute with a whiteboard and a checklist:

- **Failure anchoring:** you must name an observable breakdown, not a vague aspiration.
- **Decision focus:** you must name the decision being optimized (priority, scope, ownership, sequencing, investment, diagnosis, repair).
- **Object of control:** you must choose what the system can actually manipulate (not outcomes).
- **Inspectable artifacts:** you must produce something others can challenge and revise.
- **Constraints with defaults:** you must force tradeoffs and define what happens when people avoid them.
- **Misuse modeling:** you must predict how the system will be gamed or ritualized and design mitigations.
- **Adoption realism:** you must fit authority, unit of analysis, and time-to-first-value.

AI can help generate content, but it cannot replace these constraints. When applied consistently, they prevent the most common failure mode in system work: **cargo-culting systems that look rigorous but don't change decisions.**

PART I — ORIENTATION (Anti-Cargo-Cult Layer)

Chapter 1 — What This Book Is Not

This chapter exists to prevent the book itself from becoming the kind of problem it is trying to solve: a framework adopted for comfort, status, or ritual.

If you misunderstand what this book is for, you will use it to justify decisions you already want to make. That failure mode is common, socially rewarded, and hard to notice from the inside.

The Failure This Chapter Prevents

Observable failure: people adopt systems without understanding what they optimize or how they break.

Common symptoms:

- A framework is chosen because "strong teams use it"
- People argue about terminology instead of outcomes
- Process changes increase meeting load but not decision quality
- "Alignment" becomes a synonym for "agreement we can't explain"
- Systems are layered on top of systems until nothing is accountable

This book is designed to reduce that failure by teaching you to treat systems as engineered constructs with explicit decision logic, constraints, and failure modes.

This Book Is Not a Framework Catalog

You will not find "the best framework for X" lists.

Why:

- Lists optimize for **recognition** ("I've heard of that"), not for **fit**
- They invite cargo-culting: adopting the visible structure without the operating logic
- They ignore the core truth: the same framework can succeed or fail depending on the failure it is meant to address and the constraints of the environment

The goal is not to memorize systems. The goal is to be able to **evaluate** any system.

This Book Is Not a Theory Book

It includes theory, but only as needed to operate.

Theory is here to support decisions like:

- Is this problem more linear or feedback-driven?
- Is this primarily a flow/constraint issue or a cooperation issue?
- Is this a domain boundary problem or a prioritization problem?

If a concept does not change what you do next, it does not belong.

This Book Is Not a Template Library

Templates can help, but they also produce a predictable failure:

- People fill boxes without changing thinking
- Artifacts become performative
- The system becomes “the form” rather than the decision logic

This book provides structures, but it does not promise that filling them out equals correctness.

Rule: a system is not the template. A template is only a container for decision logic.

This Book Is Not a Process Improvement Manual

Most process improvement fails for two reasons:

1. It treats local friction as the problem, rather than the decisions that create the friction.
2. It “adds process” without adding constraints or artifacts that make decisions inspectable.

This book is about **decision quality under constraints**, not about polishing routines.

This Book Is Not a Leadership Philosophy

It will not tell you how to be inspiring, empathetic, or visionary.

It will tell you how to:

- Make assumptions explicit
- Force inspectability
- Prevent misuse
- Avoid designing systems that collapse under social reality

Leadership matters, but this is not that book.

What This Book Refuses to Do

This refusal list is a constraint. It defines the boundary of the system you are learning.

This book refuses to:

- Substitute vocabulary for clarity
- Treat “alignment” as a goal without specifying the decision it serves
- Recommend a system without first naming the failure it prevents
- Assume adoption is the same as effectiveness
- Ignore misuse because it is inconvenient or politically sensitive
- Pretend systems work the same across units of analysis (team vs org vs ecosystem)

If you want a method you can apply without thinking, this book will disappoint you.

The Core Reframe

A system is not a set of steps.

A system is:

- a **decision machine**
- designed to reduce a **specific failure**
- by controlling a **specific object**
- producing **inspectable artifacts**
- under explicit **constraints**
- with an explicit **misuse model**

Everything else is decoration.

The Non-Negotiable Rule Introduced Here

You may not apply or design a system until you can answer:

1. What is the observable failure?
2. Which decision is being optimized?
3. What artifact will make that decision inspectable?
4. What constraint prevents the most likely misuse?

If you cannot answer these, stop. You do not have a system yet.

Misuse Warnings for This Book

This book can be misused in predictable ways. Naming them is part of the system.

Misuse 1: Turning this book into a new vocabulary

Symptom: you start labeling everything (“unit of analysis”, “object of control”) but decisions do not improve.

Antidote: require an artifact that changes a real decision within a week.

Misuse 2: Using the lens as a debate weapon

Symptom: you use “misuse model” and “constraints” to win arguments rather than to improve reality.

Antidote: the lens is not a judge. It is an engineering discipline. Apply it to your own system first.

Misuse 3: Designing systems you don’t have authority to enforce

Symptom: you invent constraints you can’t actually hold.

Antidote: adoption path is part of validity. If enforcement is impossible, redesign the system.

Misuse 4: Over-designing

Symptom: you keep refining the system instead of using it to make decisions.

Antidote: the minimum viable system is: one decision, one artifact, one constraint, one misuse warning.

Exit Condition for This Chapter

You are ready to proceed if you can write:

- A one-paragraph **Observable Failure Statement** from your environment
- The **decision type** you want to optimize
- One **artifact** that would make the decision inspectable
- One **constraint** that forces the system to have teeth

If you can’t, the right next step is not “read more.” The right next step is to go observe the failure.

Chapter 2 — Systems as Decision Machines

Most frameworks fail in practice because people treat them as knowledge, culture, or ceremony.

This chapter installs the core operating model of the book:

A system is a decision machine.

If the system does not reliably improve a specific decision under real constraints, it is not functioning—no matter how elegant its concepts appear.

The Failure This Chapter Prevents

Observable failure: teams adopt systems that increase activity but do not improve decisions.

Symptoms:

- More meetings, same uncertainty
- More artifacts, less accountability
- More terminology, less shared understanding
- "Process compliance" replaces problem-solving
- Decisions are delayed until urgency forces them

Underlying pattern:

- People optimize for *feeling organized* rather than *making decisions inspectable*.

What "Decision Machine" Means

A **decision machine** is anything that:

1. Takes ambiguous reality as input
2. Applies constraints and rules of interpretation
3. Produces an output that commits people to action

That output might look like:

- a priority order
- an approved scope boundary
- a diagnosis of a failure
- an ownership assignment
- an investment choice
- a sequence / roadmap
- a repair plan

If you cannot name the output decision, you are not looking at a system. You are looking at a language, a ritual, or a set of preferences.

The Core Test

A system is valid only if it passes this test:

> **Which decision becomes safer, faster, or harder to avoid because this system exists?**

"Safer" means

- fewer irreversible mistakes

- fewer unexamined assumptions
- fewer hidden tradeoffs
- reduced variance (more predictable outcomes)

“Faster” means

- reduced time-to-commit (not reduced time-to-talk)
- fewer loops caused by ambiguity or missing information
- fewer stalled decisions waiting for consensus theater

“Harder to avoid” means

- the system forces confrontation with reality
- avoidance strategies (politics, vagueness, deferral) become costly
- accountability is structurally embedded

If a system makes decisions easier to *delay*, it is anti-functional.

Why People Confuse Systems with Other Things

Mistake 1: Treating a vocabulary as a system

Vocabulary helps thinking, but it doesn’t force decisions.

A vocabulary becomes a system only when:

- it constrains interpretation, and
- it produces an inspectable output that changes action.

Otherwise it is just a shared set of labels.

Mistake 2: Treating a ritual as a system

Ritual can coordinate behavior, but it often lacks enforcement.

A weekly ceremony is not a system unless:

- it yields a decision output, and
- skipping the decision output is treated as failure, not “we’ll revisit.”

Mistake 3: Treating a tool as a system

Tools store and display information. They do not create decision logic.

If a tool is doing the work, what’s really happening is:

- the tool is imposing hidden constraints, or
- people are outsourcing thinking to defaults.

This book prefers explicit constraints over accidental ones.

The Decision Types Systems Typically Optimize

Use this set to prevent vague goals like “clarity” or “alignment.”

- **Priority:** what matters most now?
- **Scope:** what is in vs out?
- **Ownership:** who is responsible for what?
- **Sequencing:** in what order should we do things?
- **Investment:** what gets time/money/attention?
- **Diagnosis:** what is the real failure and why?
- **Repair:** what changes will remove the constraint?

Most dysfunction is a disguised failure in one of these seven decisions.

Systems Don’t “Create Alignment” — They Create Commitments

“Alignment” is not a decision. It is at best a side effect.

If you hear:

- “We need alignment”
- “We need clarity”
- “We need to get on the same page”

Translate it into a decision question:

- Priority: “What do we do first?”
- Scope: “What are we not doing?”
- Ownership: “Who owns the interface?”
- Sequencing: “What blocks what?”
- Investment: “What are we funding?”
- Diagnosis: “What’s causing the miss?”
- Repair: “What changes are we making?”

If you cannot translate it, you are not ready to select a system.

The Minimal Anatomy of a Decision Machine

A functioning system includes:

1. **Trigger:** when the system runs (a cadence, an event, a threshold, a failure)
2. **Inputs:** what information is considered valid
3. **Rules:** constraints and interpretation logic
4. **Artifact:** an inspectable representation of the decision
5. **Enforcement:** what happens if the decision is avoided
6. **Feedback:** how outputs are evaluated and corrected

If any of these are missing, the system will drift toward ritual.

Inspectability Is the Source of Power

A decision machine must leave evidence.

Without inspectability:

- there is no disagreement surface
- there is no learning loop
- there is no accountability trail
- the system becomes politics by other means

Inspectable artifacts are not bureaucracy. They are the mechanism that turns thinking into a shared object.

Misuse Model: How “Decision Machine” Thinking Breaks

Misuse 1: Over-optimizing one decision type

Example: optimizing priority so aggressively that ownership and sequencing collapse.

Warning sign: decisions get made quickly, but delivery becomes chaotic.

Correction: systems must acknowledge adjacent decisions they stress.

Misuse 2: Confusing decisiveness with decision quality

Fast decisions with hidden assumptions are just fast mistakes.

Correction: “faster” must come from reduced ambiguity, not reduced scrutiny.

Misuse 3: Building a machine without enforcement

If people can avoid the output without consequence, you have a suggestion engine, not a decision machine.

Correction: define what “failure to decide” triggers (escalation, default rule, timebox expiry).

Operational Checklist

Before you adopt or design any system, write these sentences:

1. **The decision this system optimizes is: __**
2. **The artifact that makes it inspectable is: __**
3. **The constraint that gives it teeth is: __**
4. **The failure mode if misused is: __**

If you can’t fill these out, the next step is not choosing a framework. The next step is defining the decision you keep failing to make.

Chapter 3 — Why Smart People Design Bad Systems

Bad systems are rarely designed by incompetent people.

They are designed by capable people responding to real pressure, using mental shortcuts that work locally but fail structurally. The problem is not intelligence—it is that system design punishes the same behaviors that are rewarded in most organizations: speed, confidence, persuasive narratives, and visible activity.

This chapter makes the failure mechanisms explicit so you can recognize them while they are happening.

The Failure This Chapter Prevents

Observable failure: teams install systems that feel rigorous but degrade decision quality over time.

Symptoms:

- The system is “adopted” but rarely changes hard decisions
- Meetings become more frequent; commitments become less reliable
- Everyone can explain the process; no one can explain why it works
- The system produces artifacts that look official but cannot be challenged
- People defend the system’s identity rather than evaluating its outputs

Underlying pattern:

- People optimize for legitimacy and coherence instead of inspectability and constraint.

The Core Dynamic: Intelligence Is Not the Constraint

System design fails because it is a socio-technical engineering problem:

- Social: incentives, status, power, conflict avoidance, identity
- Technical: interfaces, constraints, feedback loops, scaling limits

Smart people often fail here because:

- they are rewarded for persuasive abstraction, and
- punished for exposing uncertainty.

Systems require the opposite:

- explicit assumptions
- explicit constraints
- explicit failure modes

Failure Mechanism 1: Abstraction Drift

Abstraction is useful until it replaces contact with reality.

How it happens:

1. A real failure occurs (missed deadlines, churn, conflict)
2. People create a broad label (“alignment”, “execution”, “ownership”)
3. The label becomes the problem definition
4. The system is built to satisfy the label, not the failure

Warning signs:

- the problem statement contains only abstract nouns
- nobody can cite a specific incident
- “we all know what that means” appears frequently

Countermeasure:

- require an **Observable Failure Statement** that includes:
- a concrete situation
- a repeated symptom
- a measurable consequence
- who is affected

Failure Mechanism 2: Vocabulary Substitution

People replace decisions with words because words feel safer than commitments.

Examples:

- “Let’s align” instead of “Let’s pick a priority”
- “Let’s clarify scope” instead of “Let’s say no to X”
- “We need ownership” instead of “Name an owner and their authority”

Why it’s attractive:

- vocabulary scales socially
- decisions create winners and losers

What it causes:

- semantic debates
- ritualized artifacts
- avoidance of accountability

Countermeasure:

- translate abstract language into one of seven decision types:
- priority, scope, ownership, sequencing, investment, diagnosis, repair

If you can’t translate it, you don’t have a decision problem yet—you have a political or emotional problem.

Failure Mechanism 3: Framework Stacking

When a system doesn’t work, people often add another system on top.

Typical chain:

- OKRs + Agile + Architecture Review Board + “Alignment” cadence + dashboards

Why it fails:

- the systems optimize different decisions
- constraints conflict
- artifacts compete
- teams learn to perform compliance, not thinking

Warning signs:

- multiple artifacts represent the “same truth” differently
- teams spend more time reconciling artifacts than shipping or learning
- each system has defenders; none has measurable decision improvement

Countermeasure:

- insist on a single “source decision” for each recurring domain
- treat new systems as replacements unless proven otherwise

Failure Mechanism 4: Local Optimization Disguised as Strategy

Smart people can make a system that works for their team and harms the whole.

Examples:

- a prioritization system that optimizes feature delivery but breaks platform reliability
- a velocity system that improves predictability but increases long-term coupling
- a discovery system that generates insights but starves delivery

Why it persists:

- local wins are visible
- systemic costs are delayed and diffused

Countermeasure:

- explicitly name the **unit of analysis**
- state what the system does not optimize
- add a misuse warning for “success that causes harm elsewhere”

Failure Mechanism 5: Unpriced Tradeoffs

Every system makes tradeoffs; bad systems hide them.

Common hidden tradeoffs:

- speed vs correctness
- local autonomy vs global coherence
- innovation vs stability
- predictability vs adaptability
- throughput vs quality

Warning sign:

- the system is described as “best practice” with no cost model

Countermeasure:

- require a “cost of correctness” statement:
- what you are willing to lose to gain the decision improvement

Failure Mechanism 6: Confusing Legibility with Truth

Systems often become tools to make work legible to leadership rather than effective.

Legibility pressure creates:

- simplified metrics
- sanitized narratives
- artifacts that look stable even when reality is volatile

Result:

- decision machines become reporting machines
- teams learn to optimize optics

Countermeasure:

- define whether the artifact is for:
- decision-making
- coordination
- reporting

If you try to make one artifact do all three, it will be gamed.

Failure Mechanism 7: The Comfort of Completion

Smart people love closed forms: canvases, diagrams, tables.

The danger is that “filled out” feels like “done.”

Warning signs:

- the artifact is celebrated more than the decision it supports
- teams spend time perfecting format
- people ask for templates instead of constraints

Countermeasure:

- enforce an “artifact usefulness test”:
- Did this artifact change a real decision within a week?
- If not, delete it or redesign it.

Failure Mechanism 8: Hidden Authority Mismatch

A system can be logically correct and socially impossible.

Authority mismatches happen when:

- the system requires enforcing constraints no one can enforce
- ownership is declared without real decision rights
- escalation paths are undefined or ignored

Warning signs:

- “We all agreed but nothing changed”
- “That’s not my call”
- “Leadership will never go for that”

Countermeasure:

- adoption path is part of validity:
- who can run it
- who can enforce it
- what default applies when enforcement fails

Failure Mechanism 9: No Misuse Model

Smart people avoid naming misuse because it implies distrust.

But systems without misuse models will be misused immediately, because:

- misuse aligns with incentives
- misuse reduces friction
- misuse is socially safer than confrontation

Countermeasure:

- every system must include:
- how it degrades when misapplied
- what behaviors indicate misuse
- one mitigation per misuse

The Non-Negotiable Rule Introduced Here

A system is not “good” because it is coherent.

A system is good only if:

- it improves a specific decision,
- under real constraints,
- while resisting predictable misuse.

Coherence without enforcement produces folklore.

A Practical Diagnostic: The Bad System Smell Test

If you notice three or more of these, treat the system as suspect:

- It can be applied without naming a failure
- It produces artifacts that are not challenged
- It claims to optimize everything
- It relies on “buy-in” more than constraints
- It scales by vocabulary rather than by interfaces
- It measures activity more than decision outcomes
- It becomes identity (“we are an OKR company”)
- It cannot name how it breaks

Exit Condition for This Chapter

You are ready to proceed if you can name, from your environment:

1. One instance of **abstraction drift**
2. One instance of **vocabulary substitution**
3. One instance of **framework stacking**
4. One system that lacks a **misuse model**

If you can’t find examples, don’t assume you’re healthy. Assume you’re not looking closely enough.

PART II — DESIGN THEORY (Mental Model Installation)

Chapter 4 — Problem Frames and Observable Failure

Systems are built to prevent failures. If you can't name the failure, you will design a system that optimizes the wrong thing—usually something socially convenient, like "alignment," "visibility," or "consistency."

This chapter teaches the entry move of System Design Lens:

1. **Locate the failure**
2. **Make it observable**
3. **Choose the correct problem frame**
4. **Refuse to proceed without evidence**

The Failure This Chapter Prevents

Observable failure: systems are designed from abstract motivations rather than concrete breakdowns.

Symptoms:

- A new process is introduced with no baseline ("we need to improve")
- Different stakeholders mean different things by the same goal word
- Teams debate methods instead of diagnosing failures
- "Success" is defined after the fact
- The system becomes permanent even after the original issue disappears

Root cause:

- Problem statements are written to be agreeable, not accurate.

What a Problem Frame Is

A **problem frame** is the boundary you draw around "what kind of failure this is" so that:

- you don't solve the wrong problem well
- you don't apply the wrong causality model
- you don't pick artifacts that can't represent reality

Problem frames are not labels. They determine:

- what evidence counts
- what decisions matter
- what object of control is realistic

In this book, failures tend to cluster into five frames.

The Five Failure Locations

Strategy failures

What breaks:

- priorities drift
- investment choices don't match intent
- "important" work can't defeat urgent work

Observable symptoms:

- frequent re-prioritization with no learning
- roadmaps that change without triggering a decision review
- teams building things that leadership later calls "not the goal"

Decisions commonly failing:

- priority, investment, scope

Discovery failures

What breaks:

- learning is slow or untrusted
- teams build based on assumptions that aren't tested
- users are understood through proxy opinions

Observable symptoms:

- months of build work with surprise outcomes
- repeated "we thought users wanted…" postmortems
- research artifacts no one uses in decisions

Decisions commonly failing:

- diagnosis, investment, scope

Delivery failures

What breaks:

- flow, predictability, quality, throughput

Observable symptoms:

- chronic missed dates
- work items stuck in-progress
- quality debt accumulating faster than it can be paid down
- firefighting as the default mode

Decisions commonly failing:

- sequencing, repair, scope

Cooperation failures

What breaks:

- interfaces, ownership, coordination across boundaries

Observable symptoms:

- cross-team friction dominates cycle time
- unclear ownership of systems, APIs, or outcomes
- escalation replaces collaboration
- “we’re blocked” becomes a permanent status

Decisions commonly failing:

- ownership, sequencing, repair

Evolution / scaling failures

What breaks:

- the system stops working when context changes
- growth increases coupling and coordination cost

Observable symptoms:

- practices that worked for one team fail at 5–10 teams
- architectural boundaries erode
- governance expands to compensate for lack of clarity
- the organization becomes slow to adapt

Decisions commonly failing:

- ownership, investment, repair

Observable Failure vs Abstract Dissatisfaction

Abstract dissatisfaction is language like:

- “We need alignment”
- “We need better execution”
- “We need to move faster”
- “We need clarity”
- “We need accountability”

These phrases are not failures. They are requests for safety.

An **observable failure** is something that:

- is repeatedly happening
- could be witnessed by an outsider
- has a measurable cost (time, money, risk, customer impact)
- can be stated without moral judgment

The Observable Failure Statement format

Write it in 3–5 sentences:

1. **Situation:** where/when it occurs
2. **Symptom:** what repeatedly happens
3. **Consequence:** what it costs
4. **Who is impacted:** team, org, users
5. **Frequency:** how often / how long

Example (delivery frame):

- “Over the last 6 weeks, items labeled ‘small’ routinely take 2–3 weeks to ship. Work sits in review and QA with unclear handoffs. This causes planned releases to slip and forces weekend stabilization. Engineers are increasingly reluctant to pick up work outside their area because it amplifies cycle time.”

The point is not perfection. The point is **inspectability**.

Why “Alignment” Is a Smell

“Alignment” is usually a symptom of one of these real failures:

- priority is not explicit
- scope boundaries are porous
- ownership is unclear
- sequencing dependencies are hidden
- investment choices are not committed

“Alignment” becomes a goal when people don’t want to name the real decision, because naming it creates conflict.

In this book, “alignment” is acceptable only when you can complete:

> “Alignment about __ **decision, using _** ***artifact, under*** ___ constraint.”

If you can’t, drop the word.

Problem Frames Determine Causality Assumptions

This matters because the wrong causality model creates the wrong system.

Examples:

- Delivery bottlenecks often require **constraints & flow** thinking (queues, WIP, bottlenecks).
- Strategy ambiguity often requires **feedback loops** (hypotheses, metrics, learning).
- Cooperation failures often require **socio-technical** thinking (authority, incentives, boundary clarity).
- Scaling failures often require **evolutionary** thinking (selection pressures, drift, local adaptation).

If you choose a linear plan for a feedback-dominant problem, you will get false confidence and real surprises.

Evidence: What Counts, What Doesn’t

Evidence that counts

- cycle times, queue sizes, defect rates
- incident timelines
- decision logs (or absence of them)
- recurring escalation paths

- handoff points and wait states
- repeated reversals ("we decided X, then we decided not-X")

Evidence that does not count (by itself)

- "people feel misaligned"
- "leadership wants visibility"
- "teams are frustrated"
- "communication is bad"

These can be useful signals, but they are not failure definitions.

A Simple Frame Selection Tool

When you're unsure which frame you're in, ask:

1. **Are we failing to choose what matters?** → Strategy
2. **Are we failing to learn what's true?** → Discovery
3. **Are we failing to deliver reliably?** → Delivery
4. **Are we failing at cross-boundary coordination?** → Cooperation
5. **Are we failing to adapt as we scale?** → Evolution

Most real situations involve multiple frames, but you must choose the dominant one to avoid designing a system that "optimizes everything" and enforces nothing.

Misuse Model: How This Chapter Gets Misapplied

Misuse 1: Treating "observable" as "quantitative only"

Some failures are observable through consistent narratives and incidents even before metrics exist.

Correction: use incident examples and timelines as evidence until metrics stabilize.

Misuse 2: Over-framing and analysis paralysis

People keep refining the failure statement instead of acting.

Correction: timebox diagnosis. Your goal is a usable frame, not a perfect one.

Misuse 3: Choosing the frame that avoids conflict

Teams pick "delivery" because it feels technical, when the real problem is strategy or ownership.

Correction: ask "Which decision are we refusing to make?" That usually reveals the real frame.

The Non-Negotiable Rule Introduced Here

You may not select, adopt, or design a system until you can produce:

- one **Observable Failure Statement**
- one **dominant problem frame**
- one **decision type** that is failing repeatedly

If you can't do that, you don't need a system. You need observation.

Exit Condition for This Chapter

Before moving on, write:

1. Your Observable Failure Statement (3–5 sentences)
2. The dominant frame (strategy / discovery / delivery / cooperation / evolution)
3. The primary decision type currently failing (priority / scope / ownership / sequencing / investment / diagnosis / repair)

You now have the minimum input required to do real system work.

Chapter 5 — Objects of Control

Outcomes are what you want. Objects of control are what you can actually manipulate.

Most bad systems fail because they are designed to control outcomes directly ("increase innovation," "improve alignment," "move faster") without specifying what is being controlled to produce those outcomes.

This chapter makes a hard distinction:

- **Outcomes** are goals and results.
- **Objects of control** are the levers a system touches every time it runs.

If you don't name the object of control, your system will default to controlling something accidental—often meetings, documents, or reporting.

The Failure This Chapter Prevents

Observable failure: systems are installed that create activity, but the underlying levers never change.

Symptoms:

- Lots of planning, no improvement in delivery predictability
- More dashboards, same surprises
- New "alignment" cadences, same cross-team conflicts
- Repeated strategy resets with no changes in investment behavior
- Artifacts proliferate but decision-making does not improve

Root cause:

- The system has goals, but no controllable mechanism.

The Core Idea: Systems Don't Control Outcomes

You can't directly control:

- customer satisfaction
- innovation
- reliability
- speed
- "alignment"
- quality

You can influence them by controlling proximate objects like:

- constraints (WIP limits, SLAs, timeboxes)
- interfaces (APIs, contracts, ownership boundaries)
- work items (definition, size, sequence, acceptance)
- incentives (what is rewarded, punished, tolerated)
- information flow (what is visible, when, to whom)
- domains (bounded contexts, responsibility maps)

A system that tries to control outcomes directly will drift into moralizing ("we need to care more") or reporting ("let's measure it harder").

What “Object of Control” Means

An object of control is:

- **touched repeatedly** by the system
- **changeable** by the actors running the system
- **causally close enough** to influence the failure
- representable as an **artifact** or rule

A good object of control is something you can point to and edit.

If you can’t edit it, you can’t control it.

The Common Objects of Control

This book uses a deliberately short list to force precision. Most systems manipulate one or two of these.

Goals

What gets controlled:

- goal definition
- goal hierarchy
- goal review cadence
- goal acceptance criteria

Typical systems:

- OKRs, KPI trees, strategy deployment

Misuse risk:

- goals become wishes or slogans
- goals multiply without tradeoffs

Work items

What gets controlled:

- what counts as “work”
- work item size and slicing rules
- entry/exit criteria (DoR/DoD)
- prioritization and sequencing rules

Typical systems:

- agile variants, kanban policies, intake processes

Misuse risk:

- backlog as junk drawer
- prioritization without capacity reality

Interfaces

What gets controlled:

- team boundaries and integration points
- API contracts, SLAs, change protocols
- cross-team dependency handling

Typical systems:

- platform operating models, integration governance, contract testing norms

Misuse risk:

- “coordination” replaces interface clarity
- escalations become the interface

Domains

What gets controlled:

- responsibility boundaries
- language boundaries (what terms mean where)
- ownership of concepts and data

Typical systems:

- domain-driven design, team topologies, operating model definitions

Misuse risk:

- domains become political territories
- boundaries drawn without enforcement

Constraints

What gets controlled:

- WIP limits, timeboxes, approval thresholds
- quality gates
- queueing policies
- default rules when decisions are not made

Typical systems:

- flow-based delivery, incident response, change management

Misuse risk:

- constraints applied without authority
- constraints become bureaucracy rather than throughput protection

Incentives

What gets controlled:

- what is rewarded, punished, ignored
- performance metrics and evaluation criteria
- promotion narratives

Typical systems:

- performance systems, goal systems, compensation-linked measurement

Misuse risk:

- metric gaming
- local optimization

Information flow

What gets controlled:

- what is visible
- when it becomes visible
- to whom
- in what form (artifact)

Typical systems:

- dashboards, reporting cadences, decision logs, review rituals

Misuse risk:

- legibility replaces truth
- reporting replaces decisions

Selecting the Right Object of Control

Use this sequence to avoid choosing an object that is merely convenient.

Step 1: Start from the failure

Ask:

- "What *must change repeatedly* for this failure to reduce?"

If the answer is "people need to communicate better," you don't have an object. Keep going.

Step 2: Identify the decision you need to optimize

Common mappings:

- Priority decisions often control: **work items**, **investment**, **goals**
- Ownership decisions often control: **interfaces**, **domains**
- Sequencing decisions often control: **work items**, **constraints**
- Diagnosis decisions often control: **information flow**, **work item definitions**
- Repair decisions often control: **constraints**, **interfaces**, **ownership boundaries**

Step 3: Run the "edit test"

For each candidate object:

- Can the team change it within their authority?
- Is it updated often enough to matter?
- Can it be represented as an artifact or rule?
- Does changing it plausibly affect the failure within a reasonable time horizon?

If the answer is “not really,” it’s not your object of control.

The Most Common Trap: Controlling Meetings

When people don’t know what to control, they control the calendar.

Symptoms:

- new cadences
- more “syncs”
- longer planning sessions
- more reviews

This creates the illusion of governance but usually does not change:

- ownership boundaries
- sequencing rules
- constraints
- interface contracts

Meetings can be part of a system, but they are not an object of control unless the meeting produces enforceable artifacts and constraints.

Object of Control and Artifact Must Match

You can’t control an object you can’t represent.

Examples:

- If you control interfaces, you need interface artifacts (contracts, SLAs, ownership maps).
- If you control constraints, you need constraint artifacts (WIP policies, gates, default rules).
- If you control domains, you need domain artifacts (maps, glossaries, context boundaries).

If your artifact is a slide deck but your object of control is interfaces, you will get theater.

Misuse Model: How “Objects of Control” Gets Misapplied

Misuse 1: Choosing too many objects

If you control everything, you control nothing.

Failure mode:

- the system becomes complex
- enforcement collapses
- adoption becomes political

Correction:

- choose 1–2 objects of control max for a system. Treat others as secondary effects.

Misuse 2: Choosing an object outside authority

You design a system that requires changing incentives or org boundaries when you can’t.

Correction:

- redesign the system to use objects you can actually control, or define an adoption path that includes the authority holder.

Misuse 3: Picking an object that is too far from causality

Example: controlling dashboards to improve reliability.

Dashboards can improve awareness but rarely change the mechanisms that produce incidents (interfaces, constraints, quality gates).

Correction:

- prefer objects closer to causal mechanisms even if they are less comfortable.

The Non-Negotiable Rule Introduced Here

You may not design or adopt a system until you can state:

- **The object of control is _____**
- **The actors who can change it are _____**
- **The artifact that represents it is _____**
- **The constraint that enforces it is _____**

If you can't complete this, you are trying to control an outcome with vibes.

Exit Condition for This Chapter

Write:

1. Your system's 1–2 objects of control
2. Who has authority to change each object
3. The artifact that will make each object inspectable
4. One constraint that prevents drift or avoidance

If you can do that, you are now operating like a system designer, not a framework consumer.

Chapter 6 — Units of Analysis and Scale Collapse

Many systems “work” and still fail—because they are applied at the wrong scale.

A team-level method used at org-level becomes bureaucracy. An org-level governance model used at team-level becomes control theater. A system designed for one unit of analysis collapses when moved to another without redesign.

This chapter gives you a rule:

A system is only valid for the unit of analysis it was designed to control.

The Failure This Chapter Prevents

Observable failure: a system succeeds locally but fails when scaled, copied, or mandated.

Symptoms:

- A practice that worked for one team becomes painful across many teams
- “Standardization” increases coordination cost and reduces speed
- Teams comply with a system without believing in it
- Leadership adds governance to compensate for drift
- Local autonomy and global coherence fight constantly

Root cause:

- The system’s assumptions match one unit of analysis, but it is applied to another.

What “Unit of Analysis” Means

The **unit of analysis** is the smallest boundary inside which the system’s logic is true and enforceable.

It defines:

- who runs the system
- who can enforce constraints
- what “success” means
- what coupling exists
- how feedback loops work

This book uses five common units:

- Individual
- Team
- Multi-team
- Organization
- Ecosystem / Market

A system that does not name its unit of analysis will be misapplied by default.

The Five Units and What Typically Changes

Individual

What dominates:

- personal attention, habits, cognition
- high autonomy, low coordination

System risks:

- over-structuring
- self-optimization that harms team interfaces

Common decision types optimized:

- sequencing, focus, repair (personal bottlenecks)

Team

What dominates:

- shared execution and local coordination
- stable context and feedback

System risks:

- ignoring external dependencies
- optimizing team throughput while harming adjacent systems

Common decision types optimized:

- scope, sequencing, ownership (inside the team)

Multi-team

What dominates:

- dependency management, interface clarity
- shared standards, integration risk

System risks:

- coordination overload
- governance replacing interface design

Common decision types optimized:

- ownership, sequencing, repair (across boundaries)

Organization

What dominates:

- investment allocation, strategy coherence
- incentives, career ladders, portfolio control

System risks:

- false uniformity
- legibility pressure overriding truth
- politics shaping artifacts

Common decision types optimized:

- investment, priority, scope (portfolio-level)

Ecosystem / Market

What dominates:

- competition, regulation, network effects
- evolution and selection pressures

System risks:

- internal optimization that ignores external reality
- slow adaptation to market shifts

Common decision types optimized:

- investment, diagnosis, repair (strategic adaptation)

Scale Collapse: The Most Common Patterns

Scale collapse is what happens when a system's constraints, artifacts, and enforcement mechanisms do not survive a change in scale.

Pattern 1: The "Copy-Paste Team" Fallacy

A team has a successful system and assumes other teams can copy it.

What breaks:

- different constraints
- different coupling
- different skill distribution
- different external dependencies

Why it happens:

- success creates narrative confidence
- copying is cheaper than redesign

Correction:

- systems must be revalidated at each unit of analysis
- reuse concepts, not rules

Pattern 2: Standardization as a Substitute for Interfaces

Organizations standardize process because interfaces are unclear.

What breaks:

- teams lose local adaptability
- exceptions multiply
- compliance becomes the main activity

Why it happens:

- standardization is legible
- interface design is hard and political

Correction:

- at multi-team scale, prefer controlling **interfaces** over controlling **methods**
- standardize contracts and boundaries, not rituals

Pattern 3: Metrics That Don't Survive Aggregation

Team metrics don't add up cleanly at org scale.

Example failures:

- "velocity" used as org productivity
- "story points" compared across teams
- "utilization" used to predict throughput

Why it happens:

- leadership needs legibility
- aggregation is tempting

Correction:

- change metrics with unit of analysis
- enforce "no cross-team comparability" rules where needed

Pattern 4: Authority Mismatch

A system is mandated at a level where enforcement is weak.

Example:

- an org mandates a team-level practice but cannot enforce quality without creating heavy oversight

Failure mode:

- surface compliance + hidden divergence
- governance inflation

Correction:

- enforcement must match authority:
- either delegate authority to the unit running the system
- or accept the cost of governance (explicitly)

Pattern 5: Artifact Explosion

As scale grows, artifacts multiply.

What breaks:

- teams spend time reconciling representations instead of making decisions
- the system becomes reporting infrastructure

Correction:

- enforce artifact minimalism:
- one artifact per recurring decision domain
- explicit “source of truth” rules

The “Scale Triangle”: Autonomy, Coherence, Legibility

At scale, systems face a three-way tradeoff:

- **Autonomy:** teams can act locally
- **Coherence:** the org moves in compatible directions
- **Legibility:** leadership can understand and steer

Most bad systems try to maximize all three.

That produces:

- heavy reporting (legibility)
- heavy governance (coherence)
- reduced local adaptability (autonomy loss)

You must choose what to sacrifice and admit it.

Choosing the Correct Unit of Analysis

Ask these questions:

1. **Where does the failure actually occur?**
2. inside a team? between teams? in portfolio decisions?
3. **Who has the authority to enforce constraints?**
4. if enforcement requires leadership, this is not a team-level system
5. **Where is the coupling?**
6. if coupling is mostly cross-team, a team-only system will not fix it
7. **What feedback loop matters?**
8. if learning requires market signals, team-only metrics will mislead

Misuse Model: How This Chapter Gets Misapplied

Misuse 1: “We’re unique, so nothing scales”

Teams use scale differences as an excuse to avoid shared constraints.

Correction:

- don’t standardize rituals, standardize interfaces and decision outputs.

Misuse 2: Treating unit of analysis as org chart

The unit of analysis is about coupling and authority, not reporting lines.

Correction:

- map actual dependency networks, not formal structure.

Misuse 3: Forcing identical artifacts across units

Organizations mandate a single artifact format for everyone.

Correction:

- artifacts should be comparable only if the decision type and object of control are comparable.

The Non-Negotiable Rule Introduced Here

A system definition must include:

- its **unit of analysis**
- the **authority boundary** that enforces it
- what happens when it is applied outside that unit

If it can be applied at any scale without modification, it is either trivial or dishonest.

Exit Condition for This Chapter

Before moving on, write:

1. The unit of analysis for the system you are evaluating or designing
2. The authority that enforces its constraints
3. One example of how it would fail if scaled up or down
4. One redesign you would need to make it valid at a different unit

Chapter 7 — Causality Models

A system is a machine built on a belief about causality: how actions produce outcomes.

If that belief is wrong for the situation, the system can be perfectly executed and still fail. You will get confident plans, clean artifacts, and consistent disappointment.

This chapter gives you a practical rule:

Mismatch between causality model and problem reality invalidates the system.

The Failure This Chapter Prevents

Observable failure: teams apply planning and governance systems that assume a predictable world to situations governed by feedback, constraints, or social dynamics.

Symptoms:

- plans “work on paper” and fail in execution
- postmortems repeat the same surprises
- the system produces certainty, not accuracy
- people become cynical about process (“we all know it won’t happen anyway”)
- the organization oscillates between rigid planning and chaotic firefighting

Root cause:

- the system encodes the wrong causality model.

What a Causality Model Is

A **causality model** is the implicit answer to:

- “If we do X, what happens next?”
- “What can we predict, and what must we learn?”
- “What kind of evidence changes our mind?”

Every system picks one model by default. You can either choose it consciously or inherit it accidentally.

This book uses five dominant models:

- Linear planning
- Feedback loops
- Constraints & flow
- Evolution / selection
- Socio-technical dynamics

Most real environments contain more than one, but one usually dominates the failure you’re addressing.

Model 1: Linear Planning

Core assumption

Cause → effect is sufficiently stable that you can plan sequences and expect them to hold.

When it fits

- well-understood work
- stable requirements
- low uncertainty
- repeatable execution
- clear ownership and interfaces

Typical systems built on it

- Gantt-style project planning
- phase gates
- standard operating procedures
- deterministic roadmaps

What it optimizes

- sequencing decisions
- scope control
- predictability under stable conditions

Failure modes

- overconfidence in estimates
- late discovery of errors
- brittle plans that discourage learning

Smell test: if the work keeps changing after you "plan it," linear planning is being used outside its domain.

Model 2: Feedback Loops

Core assumption

You cannot reliably predict outcomes; you must act, observe, and adjust.

When it fits

- product discovery
- strategy under uncertainty
- experimentation
- user behavior change
- market-dependent outcomes

Typical systems built on it

- hypothesis-driven development
- OKR variants with learning cycles
- continuous discovery
- experiment pipelines

What it optimizes

- investment decisions under uncertainty
- diagnosis and learning
- adaptation speed

Failure modes

- endless experimentation without commitment
- metrics theater (“we measure everything, decide nothing”)
- local learning that doesn’t translate into action

Smell test: if “learning” does not change priorities or scope, the loop is decorative.

Model 3: Constraints & Flow

Core assumption

Throughput is governed by bottlenecks, queues, and capacity constraints, not by intention or effort.

When it fits

- delivery throughput problems
- reliability operations
- build/release pipelines
- support queues and escalations
- multi-stage handoffs

Typical systems built on it

- kanban with explicit WIP limits
- SRE incident management
- theory of constraints applied to delivery
- flow metrics and queue policies

What it optimizes

- sequencing decisions around bottlenecks
- repair decisions (remove constraints)
- predictability through reduced WIP and variance

Failure modes

- treating flow metrics as performance evaluation (gaming)
- local flow optimization that increases downstream load
- constraints applied without authority (paper rules)

Smell test: if work is “in progress” everywhere, your real system is uncontrolled WIP.

Model 4: Evolution / Selection

Core assumption

Success emerges through variation, selection, and retention over time. You shape conditions more than you control outcomes.

When it fits

- scaling organizations
- platform ecosystems
- innovation portfolios
- architectural evolution
- competitive environments

Typical systems built on it

- portfolio bets with explicit kill criteria
- evolutionary architecture approaches
- internal platform product models
- innovation funnels with selection gates

What it optimizes

- investment decisions across uncertain futures
- adaptability and resilience
- avoidance of single-point bets

Failure modes

- “innovation theater” without real selection pressure
- too much variation (fragmentation)
- too much selection (premature standardization)

Smell test: if nothing ever dies, you are not selecting; you are accumulating.

Model 5: Socio-Technical Dynamics

Core assumption

Outcomes are shaped by incentives, authority, trust, identity, and power, interacting with technical constraints.

When it fits

- cross-team cooperation failures
- ownership ambiguity
- governance, compliance, risk management
- cultural and incentive-driven behavior
- any environment where conflict is avoided rather than resolved

Typical systems built on it

- RACI-like ownership models (when enforced)

- interface contracts with escalation rules
- decision rights frameworks
- governance mechanisms that define authority and defaults

What it optimizes

- ownership decisions
- conflict resolution pathways
- coordination cost reduction through clear boundaries

Failure modes

- systems that pretend politics doesn't exist
- "consensus" systems that create veto power everywhere
- enforcement collapse when authority is unclear

Smell test: if the main blocker is "getting people to agree," you are in socio-technical territory.

Choosing the Right Model

Start from the failure and ask which statement is most true:

- Linear: "We mostly know what to do; we just need to execute consistently."
- Feedback: "We don't know what will work until we test and learn."
- Flow: "We know what to do, but work doesn't move."
- Evolution: "We need to explore options and let winners emerge."
- Socio-technical: "The hard part is authority, incentives, and coordination."

Then confirm with evidence:

- repeated surprises → feedback or socio-technical
- chronic queues and stuck work → flow
- fragmentation and drift at scale → evolution
- stable repeatable work → linear

Model Mismatch: The Common Failure Combinations

Linear planning applied to feedback problems

Result:

- fixed roadmaps with fragile assumptions
- large batches, late learning, public failures

Correction:

- shorten cycles; make learning artifacts part of the system.

Feedback loops applied to flow problems

Result:

- more experimentation, same bottleneck
- "improvements" that don't change throughput

Correction:

- identify the constraint; set WIP and queue policies.

Flow thinking applied to socio-technical problems

Result:

- dashboards and SLAs added, but ownership conflicts persist
- people route around the system

Correction:

- clarify authority and interfaces; define defaults and escalation.

Evolution thinking applied without selection

Result:

- endless pilots, tool sprawl, architectural fragmentation

Correction:

- add explicit selection gates and kill rules.

The Artifact Must Match the Model

Artifacts encode causality assumptions.

Examples:

- Linear planning artifacts: schedules, dependency graphs, milestones
- Feedback artifacts: hypotheses, experiment results, decision logs
- Flow artifacts: WIP limits, cumulative flow, queue policies, bottleneck maps
- Evolution artifacts: portfolio maps, bet tracking, kill criteria, standards lifecycle
- Socio-technical artifacts: ownership maps, decision rights, interface contracts, escalation rules

If you use the wrong artifact, you will observe the wrong reality.

Misuse Model: How This Chapter Gets Misapplied

Misuse 1: Treating models as categories rather than lenses

People argue “this is a flow problem” as identity.

Correction:

- choose the dominant model for the failure you’re addressing, not for the entire organization.

Misuse 2: Using “socio-technical” as an excuse

Teams label problems political to avoid engineering.

Correction:

- if the bottleneck is measurable work stuck in queues, start with flow.

Misuse 3: Over-fitting to uncertainty

Teams treat everything as unknown to avoid commitment.

Correction:

- uncertainty does not eliminate the need for decisions; it changes the decision cadence and artifact type.

The Non-Negotiable Rule Introduced Here

A system definition must state:

- its dominant causality model
- what evidence is valid within that model
- what artifact represents the model's truth
- how the system fails under model mismatch

If a system can't name its causality assumptions, it will smuggle them in anyway.

Exit Condition for This Chapter

Write:

1. The dominant causality model for your failure (linear / feedback / flow / evolution / socio-technical)
2. One piece of evidence that supports that choice
3. The artifact type that best represents truth under that model
4. One likely mismatch risk if the context changes

Chapter 8 — Constraints as Power

Most “systems” fail because they don’t force anything.

They provide language, meetings, and optional artifacts—but they don’t constrain behavior. When a system allows everything, it becomes a mirror: people see what they want to see, and nothing changes.

This chapter introduces the power source of real systems:

Constraints create force.

A system without a constraint is not a system. It is a suggestion.

The Failure This Chapter Prevents

Observable failure: teams adopt frameworks that produce activity and artifacts but do not change decisions under pressure.

Symptoms:

- The process is followed until urgency appears, then ignored
- Meetings happen, but hard tradeoffs are deferred
- “We’ll revisit” becomes the default escape hatch
- Artifacts are produced, but nothing is enforced
- The system works only when everyone is already cooperative

Root cause:

- the system has no teeth: no forced tradeoffs, no defaults, no consequences.

What a Constraint Is

A **constraint** is a rule that:

- forbids a class of behaviors, or
- forces a decision by limiting options, time, scope, or authority.

Constraints are not bureaucracy. They are the mechanism that makes decision avoidance expensive.

A constraint is valid only if it is:

- enforceable by someone in the unit of analysis
- visible (people know it exists)
- tied to an artifact or rule
- worth its cost

Why Constraints Matter More Than Principles

Principles are interpreted.

Constraints are executed.

Under stress, organizations revert to incentives and defaults, not values. Constraints define defaults.

If you want a system to survive reality, you must specify what happens when:

- people disagree
- time runs out
- priorities conflict
- ownership is contested
- capacity is exceeded

Constraints are how a system answers those situations without improvisation.

The Three Jobs Constraints Do

1) Force tradeoffs

Constraints eliminate “yes to everything” behavior.

Examples:

- WIP limits force “stop starting”
- scope caps force “say no”
- timeboxes force “decide now”

2) Prevent drift

Systems degrade over time due to convenience, politics, and entropy.

Constraints keep the system from slowly becoming optional.

Examples:

- required decision outputs
- explicit entry/exit criteria
- default rules when artifacts are missing

3) Create comparability and learning

Without constraints, every cycle is different, and you can’t tell what caused what.

Constraints stabilize the process enough to learn.

Examples:

- consistent review cadence
- stable definitions of “done”
- standardized interface contracts

Common Types of Constraints

This book uses a small set so you can choose deliberately.

Time constraints

- timeboxes for decisions
- cadence rules (review every N weeks)
- expiry rules (“if not decided by date X, default applies”)

Why they work:

- they stop endless deliberation

Misuse risk:

- timeboxes used to rush without evidence

Scope constraints

- maximum number of active initiatives
- explicit “in / out” boundaries
- capacity allocation limits

Why they work:

- they force prioritization to be real

Misuse risk:

- scope caps applied without revisiting assumptions

Participation constraints

- who must be present to decide
- who is consulted vs informed
- quorum rules

Why they work:

- they prevent decisions from being undone later by “I wasn’t involved”

Misuse risk:

- participation rules become gatekeeping

Sequence constraints

- order-of-operations rules (e.g., define interface before building)
- “no build before hypothesis” (discovery)
- “no release without gate” (reliability)

Why they work:

- they prevent predictable failure patterns

Misuse risk:

- rigid sequencing in fast-changing contexts

Authority constraints

- decision rights (“this person/team decides”)
- escalation rules
- veto rules (and limits)

Why they work:

- they resolve conflict without endless negotiation

Misuse risk:

- authority constraints used to centralize power unnecessarily

Capacity / WIP constraints

- WIP limits
- queue policies
- “stop-the-line” rules

Why they work:

- they convert invisible overload into explicit tradeoffs

Misuse risk:

- constraints used as punishment rather than flow protection

Constraints Must Have Defaults

A constraint without a default invites a workaround.

Example:

- “We must decide by Friday” (no default) → decision gets postponed
- “We must decide by Friday; if not, we ship option A” (default) → decision gets made or accepts consequence

Defaults are not threats. They are the system’s automatic behavior when humans avoid the decision.

Every recurring decision should have a default.

The Constraint Ladder: Soft to Hard

Not all constraints need to be strict. But you must know what you’re choosing.

1. **Norm:** “We should…” (weak, easy to ignore)
2. **Policy:** “We do…” (stronger, still bendable)
3. **Rule:** “We must…” (enforced)
4. **Gate:** “Cannot proceed unless…” (hard)
5. **Automatic default:** “If not X, then Y happens” (hardest)

A system that relies only on norms will fail under stress.

The Cost Model: Constraints Always Cost Something

Constraints trade one kind of pain for another.

Examples:

- WIP limits reduce flexibility but improve throughput
- authority boundaries reduce participation but improve decision speed
- gates reduce speed but reduce risk

Bad systems hide these costs.

Good systems state them:

- “We accept slower approvals to reduce incident risk.”
- “We accept reduced local autonomy to reduce cross-team conflict.”

If the cost is unacceptable, the constraint is invalid.

Constraint-Artifact Coupling

Constraints must attach to something inspectable.

Examples:

- A WIP constraint attaches to a board/queue artifact
- A scope constraint attaches to a portfolio artifact
- An authority constraint attaches to an ownership/decision-rights artifact
- A sequencing constraint attaches to a checklist or definition-of-done artifact

If the constraint lives only in people's heads, it will not survive turnover or stress.

Misuse Model: How Constraints Become Bureaucracy

Constraints create power, but they also attract misuse.

Misuse 1: Constraints without causality

A rule is added because it sounds responsible, not because it addresses a failure.

Result:

- compliance work grows
- outcomes don't change
- cynicism increases

Correction:

- every constraint must map to an observable failure it prevents.

Misuse 2: Constraints without authority

A system declares rules that nobody can enforce.

Result:

- selective enforcement
- politics replaces predictability

Correction:

- enforceability is part of constraint validity.

Misuse 3: Constraints that optimize optics

Constraints are added to make reporting look clean.

Result:

- truth becomes expensive
- people hide problems to avoid punishment

Correction:

- constraints must protect learning and safety, not just legibility.

Misuse 4: Constraints that never expire

Rules accumulate because removing them feels risky.

Result:

- process fossilization
- inability to adapt

Correction:

- add review/expiry rules for constraints (“sunset unless renewed”).

The Non-Negotiable Rule Introduced Here

Every system must specify at least one constraint that:

- forces a real tradeoff,
- has an explicit default,
- is enforceable within the unit of analysis.

If your system can be followed without ever confronting a hard tradeoff, it will become ritual.

Exit Condition for This Chapter

Write:

1. The system’s primary constraint (time / scope / participation / sequence / authority / WIP)
2. The default behavior when the constraint is violated
3. Who enforces it and how
4. The explicit cost you accept in exchange for this constraint

PART III — OPERATIONAL PRACTICE (Executable Logic)

Chapter 9 — System Landscape Identification

Before you invent a system, you should map the systems already operating in the space.

This chapter is about a specific decision:

Should we adopt an existing system, adapt one, or design a new one?

Most organizations skip this step. They either:

- copy what peers are doing, or
- invent something new without realizing they are duplicating existing logic.

System Landscape Identification prevents both.

The Failure This Chapter Prevents

Observable failure: organizations accumulate overlapping frameworks that optimize different decisions and create conflict, drift, and compliance theater.

Symptoms:

- multiple “sources of truth” for priorities
- different teams using different definitions for the same concept
- governance expands because no system is trusted
- teams spend time translating between frameworks
- new initiatives add process rather than removing it

Root cause:

- systems are introduced without a landscape view of what already exists and what decisions are already “owned.”

What “Landscape Identification” Does

Landscape identification is not “research frameworks.” It is an operational diagnostic.

It answers:

1. **What systems are currently shaping decisions?** (official or unofficial)
2. **What decision types do they optimize?**
3. **Where do they overlap or conflict?**
4. **What failures are not addressed by any system?**
5. **What should be removed before adding anything new?**

The output is a map you can use to make rational choices about adoption and design.

Inputs Required

You need only three things:

- an **observable failure statement** (Chapter 4)
- the **unit of analysis** (Chapter 6)
- the **decision type** that keeps failing (Chapter 2)

Everything else can be discovered during mapping.

Step-by-Step Method

Step 1: List the systems actually in play

Include:

- formal frameworks (OKRs, Scrum, ITIL, SAFe, DDD, etc.)
- governance mechanisms (architecture reviews, change approval boards)
- planning cadences (quarterly planning, weekly exec review)
- tooling-driven systems (ticket workflows, deployment gates)
- cultural default systems (“everything must be consensus”, “ship first, fix later”)

A “system” counts if it changes how decisions get made.

If people change behavior because it exists, it’s in the landscape.

Step 2: Classify each system by decision type

For each system, force one primary decision type:

- priority
- scope
- ownership
- sequencing
- investment
- diagnosis
- repair

If you can’t assign a decision type, the system may be:

- pure reporting (legibility system)
- a vocabulary (naming system)
- a ritual (coordination habit)

Those still matter because they consume attention and shape behavior.

Step 3: Locate each system in the lifecycle

Where does it primarily operate?

- strategy
- discovery
- delivery
- cooperation
- evolution / scaling

This prevents “everything everywhere” descriptions.

Step 4: Identify the primary object(s) of control

For each system, write 1–2:

- goals
- work items
- interfaces
- domains
- constraints
- incentives
- information flow

This reveals hidden conflicts (e.g., two systems both controlling work item priority with different rules).

Step 5: Map overlaps and collisions

Look for:

- two systems optimizing the same decision differently (collision)
- one decision optimized by no system (gap)
- a system producing an artifact that another system ignores (disconnect)
- a system that requires authority another system undermines (enforcement conflict)

Typical collision examples:

- OKRs (goals/investment) vs roadmap commitments (scope/sequencing)
- Agile team autonomy vs centralized architecture review vetoes (ownership/authority)
- Platform interface ownership vs product team “move fast” incentives (interfaces/incentives)

Step 6: Identify blind spots (the value of the map)

Blind spots are recurring failures with no supporting system.

Common blind spots:

- ownership of cross-team interfaces
- repair capacity allocation (always sacrificed to features)
- decision logging (no learning loop)
- kill criteria for initiatives (nothing dies)
- standards lifecycle (everything is permanently “recommended”)

Your landscape map should make at least one blind spot obvious.

The Primary Artifact: The Landscape Matrix

Create a matrix with one row per system:

- System name
- Decision type optimized (primary)
- Lifecycle location (strategy/discovery/delivery/cooperation/evolution)
- Unit of analysis
- Object(s) of control
- Artifact produced
- Constraint/enforcement mechanism
- Known misuse/failure mode

You do not need perfect accuracy. You need enough clarity to see overlaps and gaps.

What “Good” Looks Like in a Landscape

A healthy landscape tends to have:

- clear ownership of the major recurring decisions
- minimal overlap (or explicit precedence rules)
- one primary artifact per decision domain
- explicit defaults when decisions are not made
- visible review points where systems can be modified or retired

A sick landscape tends to have:

- many artifacts, few decisions
- many cadences, little enforcement
- unclear precedence (“which system wins?”)
- “special cases” that become the majority

Decision Outcomes From Landscape Identification

At the end, you should be able to choose one of three actions:

Action A: Adopt an existing system

Choose this when:

- the failure matches the system’s intent
- the unit of analysis matches
- enforcement is feasible
- the misuse model is acceptable

Action B: Adapt an existing system

Choose this when:

- the core logic fits, but
- object of control or constraints need adjustment, or
- the system must be made compatible with existing systems

Action C: Design a new system

Choose this only when:

- no system in the landscape optimizes the failing decision, or
- existing systems are structurally incompatible with your constraints, or
- the failure requires a new artifact/constraint combination

New system design is expensive. The map helps you justify the cost.

Misuse Model: How Landscape Identification Gets Misapplied

Misuse 1: Treating it as a literature review

You don't need to know every framework. You need to know what is operating in your environment.

Correction: map reality first, then research only as needed.

Misuse 2: Using it to attack existing systems

People use the map to blame teams ("your system is bad").

Correction: the map is diagnostic, not a weapon. Focus on collisions and gaps.

Misuse 3: Over-mapping

Teams spend weeks perfecting the map.

Correction: timebox the map. If it doesn't reveal collisions and gaps quickly, you're missing the right level of abstraction.

The Non-Negotiable Rule Introduced Here

You may not add a new system until you can answer:

- What existing system should be removed, weakened, or subordinated?
- Which decision will the new system own that is currently unowned or poorly owned?
- What artifact will become the source of truth for that decision?

If you can't answer these, the new system will become "one more layer."

Exit Condition for This Chapter

Produce a first-pass landscape matrix that includes:

1. The top 5–15 systems actually shaping decisions
2. Each system's primary decision type
3. At least two collisions and one blind spot
4. A preliminary choice: adopt / adapt / design new

Chapter 10 — System Decomposition

When someone says “we should use X” (a framework, method, operating model), you need a way to evaluate it without relying on reputation, persuasion, or aesthetics.

System Decomposition is that method.

It forces a system to reveal:

- what failure it targets
- what decisions it optimizes
- what it directly controls
- what it produces as evidence
- how it enforces constraints
- how it breaks when misused

If the system can’t answer these, it’s not robust enough to trust.

The Failure This Chapter Prevents

Observable failure: teams adopt or argue about systems based on identity (“we’re an OKR company”), fashion, or superficial similarity, then discover incompatibilities only after costly rollout.

Symptoms:

- adoption is driven by executive preference or peer imitation
- implementation debates focus on ceremony and tooling
- the system is defended despite weak outcomes (“we’re doing it wrong” forever)
- the organization can’t explain why the system works (or doesn’t)

Root cause:

- systems are chosen without decomposing their operating logic.

What System Decomposition Is

System Decomposition is a structured way to reconstruct a system’s design from first principles.

It works on:

- published frameworks
- internal processes
- “how we do things here”
- implicit cultural defaults

The outcome is not a critique. It’s a specification:

> “This system is a machine that optimizes decision X by controlling object Y, producing artifact Z, under constraint C, and it fails when misused in these ways.”

The Decomposition Dimensions

Decompose every system across these 10 dimensions.

1) Problem frame

- What failure does the system reduce?
- Where does that failure appear (strategy, discovery, delivery, cooperation, evolution)?

2) Primary object of control

- What does it directly manipulate?
- goals, work items, interfaces, domains, constraints, incentives, information flow

3) Unit of analysis

- Individual, team, multi-team, org, ecosystem/market

4) Causality model

- Linear, feedback loops, constraints/flow, evolution/selection, socio-technical

5) Decision type optimized

- priority, scope, ownership, sequencing, investment, diagnosis, repair

6) Artifacts

- What does it produce that can be inspected and challenged?

7) Vocabulary & boundary rules

- What must be named precisely?
- What vague thinking is disallowed?
- What does the system refuse to talk about?

8) Operating mode

- One-off vs continuous
- Slow strategic vs fast operational
- Facilitation-heavy vs solo-usable

9) Failure & misuse model

- How does it degrade when misapplied?
- What anti-patterns does it attract?

10) Adoption path

- Who can use it first successfully?
- Minimum viable use
- Time to first value

Step-by-Step Decomposition Procedure

Step 1: State the system in one sentence

Use this format:

> "System X optimizes __ **decisions by controlling** _, ***producing artifacts, under*** ___ constraints."

If you can't write this, you don't understand the system yet.

Step 2: Identify the target failure

Write the observable failure the system claims to address.

If the system claims to address everything, treat that as a warning sign: it is either a worldview or a consultancy product.

Step 3: Identify the decision output

Force one primary decision type.

If multiple decisions appear, choose the one that most determines outcomes, and treat the others as secondary.

Step 4: Identify the object(s) of control

Name 1–2 objects. If you end up with 5+, you're describing an operating model, not a system—and you should decompose it into smaller subsystems.

Step 5: Extract artifacts and enforcement

Artifacts:

- What is produced every time the system runs?

Enforcement:

- What happens if the artifact is missing or the decision is avoided?

If enforcement is unclear, the system likely survives only by goodwill.

Step 6: Identify the causality model embedded in the system

Look at how it expects truth to be discovered:

- by planning?
- by experimentation?
- by managing bottlenecks?
- by selecting winners?
- by negotiating power and boundaries?

Mismatch is the most common root cause of "it worked somewhere else."

Step 7: List the boundary rules

Every serious system has a "no."

Examples of boundary rules:

- “No work starts without definition-of-done” (scope/quality boundary)
- “No initiative without an owner” (ownership boundary)
- “No decision without a written artifact” (inspectability boundary)

If a system has no boundary rules, it is a vocabulary.

Step 8: Model misuse explicitly

Write at least three ways the system predictably fails.

Examples:

- becomes reporting theater
- becomes an identity (“we do X”) rather than a tool
- becomes punitive measurement
- becomes central control disguised as alignment

A system with no misuse model is unsafe to roll out.

Step 9: Define adoption path and minimum viable use

Specify:

- who can run it first
- what the minimum artifact and cadence is
- what result should be visible quickly if it works

If it requires org-wide buy-in before any value appears, treat it as high-risk.

The Primary Artifact: The Decomposition Table

Create a table with the 10 dimensions as rows and fill them in.

This artifact has two jobs:

- enable rational comparison across systems
- expose hidden assumptions and incompatibilities

It also makes debate productive:

Instead of arguing “is this framework good?”, you argue:

- “Does it optimize the decision we’re failing at?”
- “Can we enforce its constraints at our unit of analysis?”
- “Does its causality model match our environment?”

How to Compare Two Systems Using Decomposition

You can compare systems by:

- decision type optimized (do they solve the same decision?)
- object of control (do they control the same lever?)
- constraint/enforcement (do they have teeth?)
- unit of analysis (are they scale-compatible?)
- causality model (do they assume the same world?)

Two systems can coexist only if their:

- objects of control don't collide, or
- precedence rules are explicit.

Otherwise, you will get silent conflict and drift.

Misuse Model: How Decomposition Gets Misapplied

Misuse 1: Treating it as academic analysis

People decompose frameworks for intellectual satisfaction.

Correction: decomposition should end in an adoption decision: adopt, adapt, replace, or reject.

Misuse 2: Weaponizing decomposition

People use decomposition to discredit opponents.

Correction: decompose your current system first. If you can't, you're not qualified to critique replacements.

Misuse 3: False precision

Teams fill in the table with confident guesses.

Correction: mark uncertain fields explicitly and gather evidence. A "known unknown" is better than a wrong claim.

The Non-Negotiable Rule Introduced Here

No system may be adopted, mandated, or criticized until it has a decomposition table.

If someone can't tolerate decomposition, they're selling identity, not operating logic.

Exit Condition for This Chapter

Decompose one real system you are using or considering.

Your decomposition is complete when you can state:

- the observable failure it targets
- the primary decision it optimizes
- the object(s) of control it manipulates
- the artifact it produces
- the constraint that enforces it
- three predictable misuse modes
- what unit of analysis it is valid for

Chapter 11 — Deliberate System Invention

Most “new systems” are accidental: a meeting becomes a cadence, a document becomes a requirement, and soon everyone is complying with something nobody designed.

This chapter is about designing systems on purpose.

Deliberate system invention is not creativity. It is engineering:

- define the failure
- choose the decision to optimize
- choose what you can control
- produce inspectable artifacts
- enforce constraints
- model misuse
- define adoption

If any of these are missing, you’re not inventing a system. You’re inventing ceremony.

The Failure This Chapter Prevents

Observable failure: organizations invent local processes and frameworks that feel helpful but become fragile, unenforceable, or politically captured as they spread.

Symptoms:

- the “system” works only with its creator present
- adoption requires persuasion rather than enforcement
- artifacts multiply without decision clarity
- the system becomes permanent without review
- people comply superficially while routing around it

Root cause:

- system design happened implicitly, without a contract.

The System Contract (Required)

You may not call something a “system” unless you can fill this contract.

1) Target situation

Where will this system run?

- unit of analysis (team / multi-team / org / ecosystem)
- environment characteristics (stability, coupling, risk profile)
- why now (what changed or became intolerable)

2) Observable failure

Write the failure in 3–5 sentences:

- situation
- symptom
- consequence
- who is impacted
- frequency

If the failure is not observable, the system will optimize appearance.

3) Root-cause assumption

State your belief about why the failure persists.

Examples:

- “Work is stuck because WIP is uncontrolled and handoffs create queues.”
- “We miss strategy because investment decisions are made without kill criteria.”
- “Cross-team conflict persists because interfaces and ownership are ambiguous.”

This assumption is not guaranteed to be true; it is the hypothesis your system encodes.

4) Primary object of control

Choose 1–2 objects:

- goals
- work items
- interfaces
- domains
- constraints
- incentives
- information flow

If you pick more than 2, you’re designing an operating model; decompose into subsystems.

5) Decision to optimize

Pick one primary decision type:

- priority
- scope
- ownership
- sequencing
- investment
- diagnosis
- repair

This is the system’s purpose.

If you can’t choose one, your system will expand until it becomes political.

6) Artifact(s)

Specify what the system produces every time it runs.

Artifacts must be:

- inspectable
- challengeable
- stable enough to compare over time
- clearly tied to the decision

Examples:

- decision log entry
- ownership map
- interface contract
- priority stack with capacity allocation
- bottleneck map + WIP policy
- portfolio bet tracker with kill criteria

7) Non-negotiable rule (constraint)

This is the power source.

Choose at least one constraint that:

- forces a real tradeoff
- has a default if not obeyed
- can be enforced within the unit of analysis

Examples:

- “Max 3 active initiatives; new work displaces old work.”
- “No work starts without an explicit owner and exit criteria.”
- “If a dependency isn’t resolved in 48 hours, escalation path triggers automatically.”
- “If decision is not made by Friday, default option A ships.”

8) Misuse warning

Name how the system will be misused.

At minimum include:

- one misuse that turns it into reporting theater
- one misuse that turns it into ritual
- one misuse that turns it into power capture

Then specify one mitigation per misuse.

9) Adoption path

Define:

- who can successfully use it first
- minimum viable use (smallest real instance)
- time to first value (what changes quickly if it works)
- how it expands (and what redesign is required to scale)

Adoption is part of design. A system that cannot be adopted is not valid.

The Invention Procedure

This is how you fill the contract in practice.

Step 1: Write the failure and refuse to abstract it

Don't write "alignment." Write what happens:

- decisions reversed
- work stuck in queues
- repeated incidents
- conflicting priorities

Step 2: Name the decision you keep failing to make

Most system invention is actually decision invention.

Examples:

- "We are failing to decide what not to do" → scope/priority
- "We are failing to decide who owns the interface" → ownership
- "We are failing to decide what to fix vs tolerate" → repair

Step 3: Choose an object you can control this week

Avoid fantasy objects (incentives, org chart) unless you own them.

Start with objects you can actually edit:

- work item definitions
- interface contracts
- WIP limits
- decision logs
- ownership maps

Step 4: Design the artifact first, then the cadence

Artifacts make thinking inspectable.

Cadence without artifact becomes meeting.

Design:

- what the artifact contains
- how it is reviewed
- how it triggers action
- where it lives (single source of truth)

Step 5: Add the constraint and default

Ask:

- “How will people avoid this decision?”
- “What rule makes avoidance expensive?”
- “What default happens when avoidance occurs?”

A constraint without a default is a request.

Step 6: Simulate misuse

Run three rehearsals:

1. How a busy team will weaken it
2. How leadership will turn it into reporting
3. How a political actor will capture it

If you can’t imagine misuse, you are not done.

Step 7: Define minimal viable use

Describe the smallest run that produces value:

- one decision cycle
- one artifact output
- one enforced constraint

If you can’t do it small, you can’t do it at scale.

Patterns for System Shapes

Most systems fall into a few shapes. Pick one that matches your decision and object of control.

Diagnostic systems

Purpose: improve diagnosis decisions.

Artifacts:

- incident timeline
- causal map
- decision log

Constraints:

- mandatory post-incident review
- required “next change” output

Allocation systems

Purpose: improve investment/priority decisions.

Artifacts:

- portfolio allocation table
- priority stack with capacity

Constraints:

- capacity caps
- explicit tradeoff rules

Boundary systems

Purpose: improve ownership decisions.

Artifacts:

- ownership map
- interface contracts

Constraints:

- no change without owner approval
- escalation defaults

Flow-control systems

Purpose: improve sequencing/repair.

Artifacts:

- WIP policy
- bottleneck map

Constraints:

- WIP limits
- stop-the-line rules

Selection systems

Purpose: manage evolution and adaptation.

Artifacts:

- bet tracker
- kill criteria

Constraints:

- sunset rules
- selection gates

Pick one shape and keep it narrow. Hybrid systems are where complexity hides.

Misuse Model: The Most Common Invention Failures

Misuse 1: The system is just a meeting

Artifact is weak or absent, so the system becomes a cadence.

Mitigation:

- no artifact, no meeting; meeting exists to produce the artifact.

Misuse 2: The system controls everything

It becomes an operating model with no clear decision output.

Mitigation:

- one primary decision type; everything else is secondary.

Misuse 3: The system cannot be enforced

Constraints require authority you don't have.

Mitigation:

- redesign to use controllable objects, or explicitly include the authority holder in adoption path.

Misuse 4: The system is never allowed to die

It becomes permanent because removing it feels risky.

Mitigation:

- add expiry/review rules ("sunset unless renewed").

The Non-Negotiable Rule Introduced Here

A system is invalid unless it contains:

- one optimized decision
- one inspectable artifact
- one enforceable constraint with a default
- one misuse model
- one adoption path

If any is missing, you are designing ceremony.

Exit Condition for This Chapter

Produce a completed System Contract for one new system you might build.

You are done when you can answer:

- What failure does this reduce?
- What decision does it optimize?
- What does it directly control?
- What artifact does it produce?
- What constraint enforces it (and what default applies)?
- How will it be misused, and what mitigates that?
- Who can adopt it first, and what is minimal viable use?

Chapter 12 — System Review & Validation

Designing a system is not the hard part.

Keeping it valid over time is.

Systems drift. Incentives change. People route around constraints. Artifacts become performative. The environment that made the system useful shifts under it.

This chapter provides a review discipline that answers one question:

Should we keep, modify, subordinate, or remove this system?

The Failure This Chapter Prevents

Observable failure: systems persist after they stop improving decisions, because they have identity, inertia, and institutional defenders.

Symptoms:

- "We do X" becomes cultural identity rather than a choice
- the system becomes compliance theater
- exceptions accumulate until the rule is meaningless
- the system expands to cover more territory instead of staying effective
- people fear removing the system because it feels like removing safety

Root cause:

- no explicit validation criteria and no retirement mechanism.

Review Is Part of System Design

A system without review becomes:

- ritual
- ideology
- bureaucracy

Validation must be built in because:

- context changes
- the system gets gamed
- the system's costs grow
- the original failure may no longer be the dominant failure

If a system cannot be questioned, it is no longer a tool.

The Review Outputs (The Only Four Allowed)

A review must end with one of these decisions:

1. **Keep** (system remains as-is)
2. **Modify** (change object of control, artifact, constraints, cadence, or scope)
3. **Subordinate** (keep it, but clarify precedence under another system)
4. **Remove** (retire it; replace only if needed)

If the review ends with "we'll revisit," the system is already winning against you.

Validation Criteria (The Test Suite)

A system is valid only if it passes all five criteria below.

1) Problem fit

- Is the original observable failure still present?
- Is it still the dominant failure?
- Has the failure moved to a different frame (strategy ↔ delivery ↔ cooperation)?

A system that solved yesterday's failure may be today's drag.

2) Decision clarity

- Is the optimized decision still explicit?
- Can people name it consistently?
- Are decisions actually being made, or merely discussed?

If outputs are not decisions, you're paying for a meeting.

3) Artifact inspectability

- Is the artifact produced consistently?
- Can it be challenged by informed participants?
- Does it reflect reality or optics?
- Is it used in subsequent decisions?

Artifacts that are not used are reporting waste.

4) Constraint enforcement

- Are constraints actually enforced?
- Do defaults trigger when decisions aren't made?
- Are exceptions explicit and rare—or invisible and common?

If constraints aren't enforced, the system exists only as narrative.

5) Misuse resistance

- Are predictable misuse modes occurring?
- Is the system being gamed?
- Has it become identity or power leverage?

If misuse dominates, redesign or retire.

The Review Procedure (Step-by-Step)

Step 1: Reconstruct the system's intent

Write a one-sentence spec:

> "This system exists to reduce **__ failure by optimizing _** ***decisions through control of , producing artifacts, enforced by*** **___** constraints."

If you can't write this, you can't review it. You have an ambient ritual.

Step 2: Reconfirm the observable failure with evidence

Collect at least one of:

- incident timelines
- cycle-time / queue evidence
- repeated decision reversals
- escalation logs
- missed commitments and their causes

If you have only opinions, you have politics, not validation.

Step 3: Inspect real artifacts, not descriptions

Bring examples:

- the last 3 artifacts produced by the system
- the last 3 decisions the system claims to have produced

Then ask:

- Were these artifacts used to commit to action?
- Do they reflect what actually happened?
- Can outsiders interpret them?

If artifacts don't survive inspection, the system isn't working.

Step 4: Test enforcement and defaults

Ask:

- What happens when people don't comply?
- Has that happened recently?
- Did the system respond automatically (defaults) or socially (nagging)?

If enforcement relies on reminders and heroics, the system is weak.

Step 5: Run the misuse audit

Use three lenses:

- **Busy-team misuse:** how do people weaken it to save time?
- **Leadership misuse:** how does it become reporting and control?
- **Political misuse:** how does it become a weapon, veto, or shield?

For each misuse observed:

- name the mechanism
- name the incentive that drives it
- name the mitigation (change constraint, artifact, or authority)

Step 6: Check compatibility with the system landscape

Systems rarely fail in isolation. They fail in collision.

Ask:

- Does this system conflict with another system's artifacts or constraints?
- Is it duplicating a decision already owned elsewhere?
- Should it be subordinate rather than primary?

If two systems both claim to be "the source of truth," one will become theater.

Step 7: Decide: keep, modify, subordinate, or remove

Do not leave without a decision output.

What to Modify (The Levers)

Most modifications fall into a small set. Choose deliberately.

Modify the object of control

Example:

- from "work items" to "interfaces" if the real failure is cross-team friction

Modify the artifact

Example:

- from status reports to decision logs
- from goals lists to allocation tables with capacity

Modify the constraint

Example:

- add a WIP cap
- add a default rule when decisions are not made
- add expiry rules to prevent accumulation

Modify the unit of analysis

Example:

- a team-level system moved to multi-team requires interface artifacts and authority boundaries

Modify cadence and triggers

Example:

- shift from time-based reviews to event-based triggers (incidents, thresholds)

Retirement Rules (Prevent Fossilization)

A system needs an exit mechanism. Otherwise it will never die.

Use one or more:

- **Sunset clause:** system expires unless renewed on evidence
- **Kill criteria:** defined conditions that trigger retirement or redesign
- **Cost cap:** if system consumes more than X time/effort, it must justify itself
- **Replacement rule:** introducing a new system requires retiring or subordinating an old one

Retirement is not failure. It is evidence the organization can learn.

Misuse Model: How Review Becomes Theater

Misuse 1: Reviews become compliance audits

People check boxes rather than evaluating decision improvement.

Mitigation:

- require evidence of decision outcomes, not process adherence.

Misuse 2: Reviews become blame sessions

People use reviews to punish teams or protect status.

Mitigation:

- review the system as an engineered object; separate performance evaluation from system evaluation.

Misuse 3: Reviews produce “more process”

When systems fail, organizations add oversight.

Mitigation:

- treat added governance as a cost; prefer changing constraints and artifacts first.

The Non-Negotiable Rule Introduced Here

A system must have a review cadence and a retirement mechanism.

If it cannot be removed, it is not a tool. It is an institution.

Exit Condition for This Chapter

Perform a review of one system currently in use.

You are done when you can produce:

1. The one-sentence system spec (intent reconstruction)
2. Evidence of the current failure state
3. Examples of real artifacts and decisions
4. A misuse audit with at least two observed misuses
5. One decision: keep / modify / subordinate / remove

PART IV — REFERENCE DOCTRINE (Operational Memory)

Chapter 13 — Canonical Dimensions (Reference)

This chapter is a reference sheet you can apply to any framework, method, operating model, or “how we do things here” practice.

It exists to make systems inspectable.

If you cannot decompose a system across these dimensions, you do not understand it well enough to adopt, mandate, or criticize it.

The Canonical Dimension Set

Use these ten dimensions in order. They are designed to force decision clarity, constraint, and misuse resistance.

1) Problem frame

Purpose: anchor the system in a concrete failure.

Answer:

- What observable failure does this system reduce?
- Where does it appear: strategy, discovery, delivery, cooperation, evolution?

Red flags:

- “alignment” without a decision
- “execution” without a failure description
- “best practices” as justification

2) Primary object of control

Purpose: identify what the system directly manipulates.

Choose 1–2:

- goals
- work items
- interfaces
- domains
- constraints
- incentives
- information flow

Red flags:

- controlling outcomes directly
- controlling “communication” without a mechanism
- controlling too many objects (becoming an operating model)

3) Unit of analysis

Purpose: prevent scale collapse.

Choose one:

- individual
- team
- multi-team
- organization
- ecosystem/market

Red flags:

- “works at any scale”
- unclear enforcement authority
- copying team practices org-wide without redesign

4) Causality model

Purpose: match system logic to reality.

Choose one dominant model:

- linear planning
- feedback loops
- constraints & flow
- evolution / selection
- socio-technical dynamics

Red flags:

- deterministic plans in high uncertainty
- experimentation used to avoid commitment
- flow tools applied to authority conflicts
- “innovation” without selection pressure

5) Decision type optimized

Purpose: make the system’s purpose explicit.

Pick one primary decision type:

- priority
- scope
- ownership
- sequencing
- investment
- diagnosis
- repair

Red flags:

- claims to optimize everything
- outputs that aren’t decisions (just discussion)
- decision type unclear or shifting

6) Artifacts

Purpose: enforce inspectability.

Answer:

- What does the system produce every time it runs?
- Where is the artifact stored?
- Who can challenge it?

Common artifact types:

- map
- table
- score
- vocabulary
- contract
- canvas
- rule set
- decision log entry

Red flags:

- artifacts that are never used in later decisions
- artifacts optimized for reporting rather than truth
- artifacts too ambiguous to dispute

7) Vocabulary & boundary rules

Purpose: prevent semantic drift and vague thinking.

Answer:

- What terms must be defined precisely?
- What vague terms are disallowed or must be operationalized?
- What does the system refuse to do?

Red flags:

- key terms left to interpretation
- “alignment,” “value,” “impact” used without qualifiers
- no explicit “no” (no boundary rules)

8) Operating mode

Purpose: ensure the system can be run as intended.

Answer:

- Is it one-off or continuous?
- Is it slow strategic or fast operational?
- Is it facilitation-heavy or solo-usable?
- What triggers a run (cadence, event, threshold)?

Red flags:

- requires constant facilitation to function
- cadence without a decision output
- unclear triggers (“whenever needed”)

9) Failure & misuse model

Purpose: make breakage predictable and manageable.

Answer:

- How does the system degrade when misapplied?
- What anti-patterns does it attract?
- What incentives drive misuse?
- What mitigations exist?

Red flags:

- “people are doing it wrong” as the only explanation
- no stated misuse modes
- system becomes identity (“we are a X company”)

10) Adoption path

Purpose: ensure the system can enter reality.

Answer:

- Who can use it first successfully?
- What is minimum viable use?
- Time to first value?
- What changes are required to scale it?

Red flags:

- requires org-wide buy-in before any value appears
- unclear enforcement authority
- adoption plan is “training”

The One-Sentence System Spec

When decomposing, always write this:

> “This system reduces **__ failure by optimizing _ *decisions through control of , producing artifacts, enforced by constraints, at the* ___** unit of analysis.”

If you can’t write it, you don’t have a system definition.

Quick Diagnostic Prompts (Fast Use)

Use these prompts when you're in a meeting and need immediate clarity.

- "What failure are we seeing that this solves?"
- "Which decision will this make easier?"
- "What does it control directly?"
- "What artifact comes out every time?"
- "What constraint gives it teeth?"
- "How will people avoid or game it?"
- "Who can actually enforce this?"

Misuse Model: How This Reference Is Misused

Misuse 1: Turning dimensions into jargon

People use the words without doing the work.

Correction:

- require an artifact output per decomposition (a filled table, not spoken labels).

Misuse 2: Treating decomposition as judgment

Decomposition is a specification, not a moral evaluation.

Correction:

- evaluate fit against your failure and constraints, not against abstract ideals.

Misuse 3: Over-detailing

Teams fill the dimensions with essays and lose the decision.

Correction:

- timebox decomposition; prefer crisp statements and mark uncertainties.

Exit Condition for This Chapter

You can treat this chapter as "installed" when you can decompose:

- one system you currently use, and
- one system you are considering adopting,

and write a one-sentence system spec for each.

Chapter 14 — Common Failure Patterns

This chapter is a catalog of **system failure patterns**—not as trivia, but as a diagnostic tool.

Each pattern describes:

- the observable symptoms
- the underlying mechanism
- why it's attractive
- what it breaks
- a corrective design move

Use this chapter when you suspect your "system" is turning into ritual, theater, or politics.

Pattern 1: Vocabulary-First Systems

Symptoms

- lots of new terms
- debates about definitions
- "we need alignment" becomes frequent
- artifacts are mostly glossaries and decks
- decisions don't get faster or safer

Mechanism

Vocabulary creates shared labels without enforcing shared commitments.

People mistake naming for control.

Why it's attractive

- low conflict
- feels sophisticated
- easy to teach and spread

What it breaks

- decision accountability
- learning loops
- speed under stress

Corrective move

Add:

- one explicit **decision output**
- one **artifact** that records the decision
- one **constraint** that triggers a default if the decision isn't made

If you can't enforce those, admit it's a vocabulary, not a system.

Pattern 2: Ritual Without Artifact

Symptoms

- recurring meetings that “feel important”
- endless revisiting of the same topics
- outcomes are verbal and evaporate
- decisions are postponed by default

Mechanism

The system is actually a cadence. Cadence is mistaken for governance.

Why it’s attractive

- creates social rhythm
- provides a sense of control
- distributes responsibility through attendance

What it breaks

- inspectability
- commitment
- continuity across time and people

Corrective move

Define the meeting’s purpose as:

- produce a specific artifact
- that contains a specific decision
- that triggers specific next actions

No artifact, no meeting.

Pattern 3: Artifact Without Decision

Symptoms

- many documents and dashboards
- “status” is always available
- nobody can name the decision that changed because of the artifact
- planning produces slides, not commitments

Mechanism

Artifacts are optimized for reporting, not for decision-making.

Why it’s attractive

- legibility is rewarded
- leadership feels informed
- avoids conflict (information can be shared without choosing)

What it breaks

- prioritization
- accountability
- speed to commitment

Corrective move

Require each artifact to include:

- a decision field (“What changed?”)
- an owner
- a default if no decision is made
- a link to the next action

If the artifact can’t change action, delete it.

Pattern 4: Constraintless Systems

Symptoms

- “it’s a guideline”
- exceptions are constant
- under stress, people ignore the system
- enforcement relies on reminders and shame

Mechanism

Without constraints, the system cannot force tradeoffs or defaults.

Why it’s attractive

- reduces resistance
- avoids authority conflict
- feels “empowering”

What it breaks

- reliability under pressure
- learning (no stable process)
- credibility (“process doesn’t matter here”)

Corrective move

Add one enforceable constraint with a default:

- scope cap, WIP limit, timebox, gate, authority boundary

If enforcement is impossible, redesign the unit of analysis or adoption path.

Pattern 5: “Best Practice” Systems

Symptoms

- justification is “industry standard”

- system is adopted with minimal local diagnosis
- dissent is framed as immaturity (“you’ll get it later”)

Mechanism

External legitimacy replaces local problem fit.

Why it’s attractive

- reduces uncertainty
- provides political cover
- accelerates rollout

What it breaks

- fit to context
- trust (people feel coerced)
- learning (feedback is dismissed)

Corrective move

Force a local contract:

- name the local failure
- name the decision optimized
- define the artifact and constraint
- define misuse risks in your environment

If you can’t do this, you’re buying identity, not design.

Pattern 6: Framework Stacking

Symptoms

- multiple overlapping systems
- conflicting artifacts
- teams spend time translating between methods
- governance expands to reconcile contradictions

Mechanism

When one system fails, another is added rather than replacing or subordinating.

Why it’s attractive

- avoids admitting failure
- keeps everyone’s preferences alive
- adds “coverage” without removal

What it breaks

- coherence
- throughput (coordination load rises)

- accountability (no one knows which system wins)

Corrective move

Apply the landscape rule:

- each recurring decision must have one “source” artifact
- adding a system requires removing or subordinating another

Pattern 7: Scale Mismatch (Scale Collapse)

Symptoms

- a team practice mandated org-wide
- heavy governance to enforce simple routines
- widespread superficial compliance
- local adaptations are treated as failures

Mechanism

System assumptions don’t survive the unit-of-analysis change.

Why it’s attractive

- leaders want comparability
- copying seems cheaper than redesign
- success stories are compelling

What it breaks

- autonomy
- adaptability
- credibility of governance

Corrective move

Redesign for the new unit:

- control interfaces rather than rituals
- shift artifacts to match the scale (ownership maps, contracts, portfolio allocations)
- explicitly define enforcement authority

Pattern 8: Metric Capture

Symptoms

- metrics improve while reality worsens
- teams optimize numbers over outcomes
- people hide work to protect metrics
- metrics become performance weapons

Mechanism

Measurement becomes an incentive system, not an information system.

Why it's attractive

- makes leadership feel in control
- simplifies complexity
- supports comparability and accountability narratives

What it breaks

- truth
- psychological safety
- long-term performance

Corrective move

- separate metrics used for learning from metrics used for evaluation
- add "anti-gaming" signals
- tie metrics to decision logs ("what did we change because of this?")

Pattern 9: Legibility Over Truth

Symptoms

- sanitized status updates
- surprises emerge late
- bad news travels slowly
- artifacts look clean but don't predict outcomes

Mechanism

Artifacts are optimized to be understood by power, not to reflect reality.

Why it's attractive

- reduces conflict
- protects reputations
- makes planning look stable

What it breaks

- early detection
- repair speed
- trust

Corrective move

Design artifacts for truth first:

- include uncertainty explicitly
- record assumptions
- require red/yellow flags with owners and next actions
- protect candor from punishment

Pattern 10: Consensus Systems (Veto Everywhere)

Symptoms

- decisions take a long time
- “alignment” becomes endless
- people avoid proposing bold moves
- the loudest or most risk-averse actor controls outcomes

Mechanism

Consensus creates distributed veto power and rewards avoidance.

Why it’s attractive

- feels fair
- reduces overt conflict
- spreads responsibility

What it breaks

- speed
- ownership
- innovation under uncertainty

Corrective move

Introduce authority boundaries and defaults:

- define who decides
- define consult vs inform
- define escalation rules
- use timeboxed decision windows with default outcomes

Pattern 11: Systems That Can’t Die (Fossilization)

Symptoms

- systems remain after context changes
- nobody remembers why the system exists
- the system is defended as “how we do things”
- removing it feels dangerous

Mechanism

No retirement mechanism + identity attachment + fear of chaos.

Why it's attractive

- stability feels safe
- removing systems creates political risk
- "more process" seems responsible

What it breaks

- adaptability
- speed
- morale (people feel trapped in rituals)

Corrective move

Add review and retirement rules:

- sunset clauses
- kill criteria
- cost caps (time budget)
- explicit "replace vs stack" governance

Pattern 12: Hero-Dependent Systems

Symptoms

- works only when one person is present
- unclear rules; "just ask Alex"
- decisions stall when the hero is absent
- knowledge is social, not inspectable

Mechanism

Enforcement and interpretation live in a person, not in artifacts and constraints.

Why it's attractive

- fast initially
- avoids writing things down
- hero becomes a coordination hub

What it breaks

- scalability
- resilience
- onboarding and continuity

Corrective move

Move the system into artifacts:

- decision logs
- explicit rules and defaults
- ownership maps
- documented constraints

If you can't make it inspectable, it's not a system—it's a person.

How to Use This Chapter

When evaluating a system, ask:

- Which two patterns does this system most resemble?
- Which pattern is it drifting toward under stress?
- What single constraint or artifact change would reverse the drift?

This is not about perfection. It's about preventing predictable failure.

Exit Condition for This Chapter

Pick one system you currently use and do two things:

1. Identify the top 2 failure patterns it exhibits today
2. Apply one corrective move that changes either:
3. a constraint, or
4. an artifact, or
5. an authority boundary

If nothing changes in decisions within a week, the correction was cosmetic.

Chapter 15 — Minimal Viable Systems

Most systems fail because they start too large.

They try to fix everything, satisfy everyone, and represent the full complexity of reality. That produces a familiar outcome:

- heavy adoption cost
- weak enforcement
- artifact overload
- inevitable drift into ritual

This chapter defines an alternative: **Minimal Viable Systems (MVS)**.

An MVS is the smallest system that can reliably improve a decision under real constraints and survive misuse.

The Failure This Chapter Prevents

Observable failure: teams design systems that are too complex to adopt, too broad to enforce, and too expensive to maintain.

Symptoms:

- long rollout plans and training programs
- many artifacts created before any decision changes
- stakeholders demand customization before first use
- the system requires facilitation to function
- the system becomes “optional” because it’s too heavy

Root cause:

- the system is designed as a full operating model rather than a decision machine.

What “Minimal Viable” Means Here

Minimal viable does not mean simplistic.

It means:

- the smallest scope that still has **teeth**
- the smallest artifact that is still **inspectable**
- the smallest constraint that still **forces tradeoffs**
- the smallest adoption path that still produces **visible value**

An MVS must still include:

- one observable failure target
- one optimized decision type
- one object of control (maybe two at most)
- one inspectable artifact
- one enforceable constraint with a default
- one misuse warning
- one adoption path

If you remove any of these, the system becomes either:

- a suggestion, or
- a template, or
- a meeting.

The MVS Core Formula

You can build an MVS by committing to this formula:

One Decision + One Artifact + One Constraint + One Default

Everything else is optional until proven necessary.

One Decision

Pick one primary decision type:

- priority
- scope
- ownership
- sequencing
- investment
- diagnosis
- repair

Avoid multi-decision systems at first. They become political immediately.

One Artifact

The artifact is the “shared object” that makes the decision inspectable.

Examples (by decision type):

- Priority → ranked stack with capacity allocation
- Scope → explicit in/out boundary list with change log
- Ownership → ownership map for one domain/interface
- Sequencing → dependency map + chosen order
- Investment → allocation table + rationale + kill criteria
- Diagnosis → causal map + “next change” decision
- Repair → repair backlog with protected capacity rule

One Constraint

A constraint is how the system forces reality to be confronted.

Examples:

- “Max 3 active items; new work displaces old.”
- “No work starts without an owner and exit criteria.”
- “If unresolved dependency > 48h, escalate.”
- “If decision not made by Friday, default ships.”

One Default

Defaults are what prevent “we’ll revisit” from defeating the system.

Examples:

- “If we can’t agree, we pick the option with lower blast radius.”
- “If priority isn’t decided, we continue the current top item.”
- “If ownership unclear, the team closest to the interface owns until reassigned.”

Defaults must be explicit and socially acceptable, or they will be ignored.

The MVS Build Procedure

Step 1: Choose a narrow target failure

Pick a failure you can observe weekly.

Good targets:

- “Work items labeled small take > 10 days”
- “Ownership disputes block releases”
- “Priorities reverse after executive reviews”
- “Incidents repeat without structural fixes”

Bad targets:

- “Improve alignment”
- “Increase innovation”
- “Improve culture”

Step 2: Choose a decision you can change immediately

Ask:

- “What decision do we keep avoiding or re-litigating?”

Choose one.

Step 3: Pick the smallest object of control that touches the failure

Prefer objects you can edit without permissions:

- work item definitions
- WIP limits
- interface contracts
- ownership maps
- decision logs

Avoid incentives and org chart unless you own them.

Step 4: Design the artifact to be hard to fake

Artifacts should not be “nice to have.” They should be hard to produce dishonestly.

Techniques:

- include a “decision changed” field
- include an owner and timestamp
- include assumptions and constraints
- include the next action and deadline
- make it visible to the people affected

Step 5: Add a constraint with an explicit default

Ask:

- “How will we avoid this decision?”
- “What rule blocks avoidance?”
- “What happens automatically if we still avoid it?”

Step 6: Define minimal adoption

An MVS must be adoptable by a small group without institutional permission.

Define:

- who runs it first
- how often it runs
- what it produces on the first run
- what improvement should be visible within 1–2 cycles

If first value requires org-wide rollout, it’s not minimal.

Minimal Viable System Patterns (Reusable Shapes)

These are MVS “shapes” you can deploy quickly.

Pattern A: Decision Log MVS

Target failure:

- repeated reversals
- loss of rationale
- blame cycles

Decision optimized:

- diagnosis / investment / scope

Artifact:

- one-page decision record: context, options, decision, rationale, owner, review date

Constraint + default:

- “No major work starts without a decision record.”
- Default: “If no record, work cannot claim priority.”

Pattern B: WIP Cap MVS

Target failure:

- too much in progress
- slow cycle time
- constant context switching

Decision optimized:

- sequencing / repair

Artifact:

- visible queue with explicit WIP count

Constraint + default:

- “Max N items in progress.”
- Default: “New work displaces lowest priority in-progress item.”

Pattern C: Ownership Boundary MVS

Target failure:

- cross-team blocks
- interface disputes
- “not my problem”

Decision optimized:

- ownership

Artifact:

- ownership map for one interface/domain + escalation path

Constraint + default:

- “Every interface must have an owning team.”
- Default: “Owning team is the one operating it in production until reassigned.”

Pattern D: Capacity Allocation MVS

Target failure:

- repair work always sacrificed
- strategy resets without behavior change

Decision optimized:

- investment / priority

Artifact:

- allocation table (e.g., feature/repair/platform) with explicit percentages

Constraint + default:

- “Allocation is fixed for the cycle.”
- Default: “If requests exceed allocation, they queue, not steal capacity.”

Misuse Model: How MVS Gets Misapplied

Misuse 1: Minimal becomes “optional”

Teams interpret minimal as “lightweight” and remove enforcement.

Correction:

- minimal means few parts, not weak constraints.

Misuse 2: Minimal becomes “pilot forever”

People keep it small to avoid commitment or organizational exposure.

Correction:

- define explicit scale criteria:
- “If X improves for 3 cycles, expand scope.”

Misuse 3: Minimal becomes “local patch”

Teams fix local pain while the real failure is cross-boundary.

Correction:

- if the failure is multi-team, your MVS must control interfaces or decision rights, not just team rituals.

The Non-Negotiable Rule Introduced Here

If you cannot produce a system that changes a real decision within two cycles, you are not designing a system—you are designing compliance.

Start minimal. Prove value. Then expand.

Exit Condition for This Chapter

Design one Minimal Viable System by writing:

1. Target observable failure (3–5 sentences)
2. One decision type to optimize
3. One artifact (name + fields it contains)
4. One constraint + explicit default
5. One misuse warning + mitigation
6. Who can adopt it first and what “first value” looks like

Chapter 16 — Teaching This Logic to Others

Systems fail socially more often than they fail logically.

You can design a clean decision machine—clear failure, clear artifact, enforceable constraint—and still watch it collapse because:

- people don't share the same frame
- authority is ambiguous
- incentives oppose enforcement
- the system threatens identity or status
- artifacts feel like surveillance

This chapter is about transferring the logic of System Design Lens to humans, in real organizations, without turning it into jargon or ideology.

The Failure This Chapter Prevents

Observable failure: a good system design fails adoption because the reasoning discipline isn't shared, and the system becomes either misunderstood, resisted, or turned into ritual.

Symptoms:

- people "agree" but behave as before
- adoption varies wildly by team
- the system gets simplified into slogans
- leadership mandates compliance to compensate
- the system becomes a political symbol

Root cause:

- the system's operating logic is not taught as a decision discipline; it is taught as process.

What You Are Actually Teaching

You are not teaching steps.

You are teaching four mental moves:

1. **Anchor in observable failure**
2. **Name the decision being optimized**
3. **Control something real (object of control)**
4. **Force inspectability and constraints, with misuse awareness**

If learners walk away with those moves, they can evaluate and design systems without you.

If they walk away with terminology, they will cargo-cult.

The Teaching Order (Do Not Change)

Teach in this sequence because it follows how systems become misused:

1. Failure → prevents abstract motivation
2. Decision → prevents “alignment” theater
3. Artifact → prevents meetings-as-systems
4. Constraint → prevents optional systems
5. Misuse model → prevents naive rollout
6. Adoption path → prevents authority fantasies

If you start with artifacts or templates, you will teach compliance.

The Minimal Teaching Kit

To teach this logic, you need only three things:

- a one-page cheat sheet (the dimensions and rules)
- one real example system (good or bad) to decompose
- one local failure to use for a small invention exercise

Everything else is optional.

How to Teach Without Jargon

Jargon is not evil; it’s just a common failure mode. Use plain language first.

Translate the canonical dimensions into everyday questions:

- Failure: “What’s repeatedly going wrong?”
- Decision: “What decision do we keep avoiding or re-litigating?”
- Object of control: “What can we actually change?”
- Artifact: “What proof will we leave behind?”
- Constraint: “What rule forces the tradeoff?”
- Misuse: “How will this get gamed or turned into theater?”
- Adoption: “Who can really enforce this?”

Only introduce canonical terms after the questions are understood.

The Core Classroom Exercise: Decompose a Familiar System

Pick something everyone experiences (for engineers, common choices are: sprint planning, incident response, architecture review, on-call handoff).

Run this exercise:

1. Write the system name on a board.
2. Ask: “What decision is it supposed to optimize?”
3. Ask: “What artifact proves it happened?”
4. Ask: “What happens when it doesn’t happen?”
5. Ask: “How is it commonly misused?”

You will usually discover:

- the decision is unclear
- artifacts are weak or unused
- enforcement is social, not structural
- misuse is common and unspoken

That discovery creates readiness for redesign.

The Adoption Reality Rule: Authority Before Elegance

People will adopt systems they can enforce.

If you teach people to design constraints without teaching authority boundaries, you will train frustration.

Always teach this sentence:

> "If you can't enforce it, it's not a rule, it's a wish."

Then teach adoption design:

- start where authority exists
- prove value quickly
- expand only when constraints can still be enforced

Teaching Through Artifacts (Best Method)

Artifacts are teachable because they are inspectable.

When introducing a system, teach by showing:

- one real artifact produced by the system
- a before/after example (bad vs good artifact)
- how the artifact changes a real decision

This avoids ideological debates and keeps learning concrete.

Artifact coaching pattern

- "Show me the last 3 artifacts this system produced."
- "Point to the decision in each."
- "Show what changed because of it."

If they can't, the system isn't functioning.

Training Misuse Awareness Without Cynicism

Misuse modeling can sound negative. Frame it as engineering:

- "Every system has failure modes."
- "We want predictable failures, not surprising ones."

Teach misuse with three lenses:

- busy-team misuse (time pressure)
- leadership misuse (legibility and control pressure)
- political misuse (status, veto, shielding)

Then teach one mitigation type:

- change the artifact
- change the constraint/default
- change authority boundaries

Avoid moralizing (“people are bad”). Focus on incentives and defaults.

The Two Common Teaching Traps

Trap 1: Teaching as “compliance”

People learn to fill forms and follow rituals.

Result:

- adoption looks high
- decision quality doesn’t improve
- cynicism grows

Avoid by:

- requiring evidence of decision improvement as success criteria.

Trap 2: Teaching as “framework identity”

People start saying “we’re a ____ organization.”

Result:

- system becomes untouchable
- critique becomes heresy

Avoid by:

- teaching systems as tools with expiry and review rules.

The Facilitation Script (Field-Usable)

Use this script in workshops and real meetings.

Opening

- “What failure are we seeing repeatedly?”
- “Which decision keeps breaking because of it?”

Specification

- “What will we control directly?”
- “What artifact will prove the decision happened?”
- “What constraint forces the tradeoff?”
- “What default triggers if we avoid the decision?”

Misuse and adoption

- “How will this get turned into reporting?”
- “How will busy teams weaken it?”

- “Who can adopt and enforce this first?”

Close with a commitment

- “What’s the smallest run we can do this week?”
- “What decision will change if it works?”

How to Measure Teaching Success

Do not measure “understanding.”

Measure behavior change:

- Can people write an observable failure statement?
- Can they name a decision type quickly?
- Can they produce an inspectable artifact without prompting?
- Can they propose a constraint with a default?
- Can they name likely misuse?

If they can do those, they’ve learned the logic.

The Non-Negotiable Rule Introduced Here

You are not done teaching until learners can design a minimal viable system without you:

- one decision
- one artifact
- one constraint + default
- one misuse warning
- one adoption path

If they need you to interpret the system, you taught dependence, not capability.

Exit Condition for This Chapter

Run a 60–90 minute session where a group:

1. Decomposes one existing system using the canonical dimensions
2. Identifies two misuse modes currently occurring
3. Designs one Minimal Viable System improvement (artifact + constraint + default)
4. Commits to running it once within a week

If they can do this, the logic has transferred.

Final Chapter — The Exit Condition

A system is supposed to make itself unnecessary.

If you are still dependent on this book to evaluate or design systems after meaningful practice, the book has failed its purpose.

This final chapter defines what “done” looks like: the capabilities that mean you can leave and operate independently.

The Failure This Chapter Prevents

Observable failure: people become dependent on frameworks, templates, or doctrine to think, and substitute “following the method” for making real decisions.

Symptoms:

- you look for the “right framework” before diagnosing the failure
- you ask for templates before naming the decision
- you treat system language as authority
- you keep adding systems instead of removing collisions
- you confuse artifact completion with outcomes

Root cause:

- the discipline of system design did not internalize as a habit.

What “Exit” Means

Exiting does not mean you stop using systems.

It means:

- you can evaluate systems without mystique
- you can invent minimal systems when needed
- you can recognize misuse early
- you can retire systems without fear
- you can teach the reasoning to others

The goal is autonomy, not loyalty.

The Four Competencies That Replace This Book

1) Failure anchoring

You can reliably do this:

- write an observable failure statement in 3–5 sentences
- locate the dominant frame (strategy, discovery, delivery, cooperation, evolution)
- refuse abstract motivations until evidence appears

You do not reach for “alignment” or “clarity” as problem statements.

2) Decision clarity

You can reliably do this:

- name the decision type being optimized (priority, scope, ownership, sequencing, investment, diagnosis, repair)
- identify when a meeting is producing discussion rather than decisions
- translate vague requests into decision questions

You notice decision avoidance as a design flaw, not a personality issue.

3) Inspectable artifacts with enforceable constraints

You can reliably do this:

- design artifacts that make decisions inspectable
- attach constraints that force tradeoffs
- define defaults that trigger when avoidance occurs
- check authority and unit-of-analysis fit before declaring rules

You do not accept “guidelines” as systems unless they have enforcement or meaningful defaults.

4) Misuse modeling and system governance

You can reliably do this:

- name how a system will be misused by busy teams, leadership pressure, and political actors
- design mitigations (artifact changes, constraint changes, authority boundaries)
- run system reviews that end with keep/modify/subordinate/remove
- retire systems with clear criteria

You treat drift as inevitable and plan for it.

The Graduation Test

You are “done” with this book when you can perform these actions without prompts.

Test A: Decompose any system in 15 minutes

Given any framework or process, you can produce:

- the one-sentence system spec
- the primary decision type
- the object of control
- the artifact and constraint
- three misuse modes
- unit of analysis and causality model

If you can do this, you are no longer vulnerable to framework marketing.

Test B: Design a Minimal Viable System in 30 minutes

Starting from a real local failure, you can produce:

- one decision output
- one artifact with fields
- one constraint with an explicit default
- one misuse warning with mitigation
- a minimal adoption plan

If you can do this, you can build systems instead of importing them blindly.

Test C: Kill a system responsibly

Given an existing system, you can:

- show evidence that its target failure is no longer dominant (or never was)
- identify collisions and costs
- propose a safe removal or replacement plan
- set a sunset timeline and review point

If you can do this, you can evolve an organization without fossilizing it.

The Minimal Oath (Practical, Not Moral)

If you keep one thing from this book, keep these rules:

1. **No failure, no system.**
2. **No decision, no system.**
3. **No artifact, no system.**
4. **No constraint, no system.**
5. **No misuse model, no rollout.**
6. **No review, no permanence.**

These rules are not ideology. They are safeguards against predictable failure.

How This Book Will Betray You If You Let It

Even this doctrine can become cargo cult.

Watch for these signs in yourself:

- using the terms to win arguments
- decomposing systems for intellectual satisfaction without changing decisions
- designing systems as a craft hobby rather than as a response to failure
- insisting on purity (“real systems must…”) instead of problem fit
- refusing to remove systems because they feel like identity

If that happens, apply the lens to the lens:

- What failure is the doctrine now causing?
- What decision are you avoiding by following it?

Then remove what no longer serves.

A Practical Exit Path

If you want to leave the book behind intentionally, do this:

1. Pick one real failure in your environment.
2. Decompose one existing system relevant to it.
3. Modify or remove one constraint or artifact based on the decomposition.
4. Run one review cycle and record what changed.
5. Teach the method to one other person by doing the exercise together.

If you can do this without needing new frameworks, you're out.

The Real Ending

Systems are tools. Tools exist to be replaced.

If you can:

- name failures precisely
- optimize decisions deliberately
- produce inspectable artifacts
- enforce constraints with defaults
- predict misuse
- retire what no longer fits

Then you don't need this book.

You can build your own doctrine, adapted to your environment, and keep it honest through review.

That is the exit condition.

TOOLKIT

Toolkit — Cheat Sheet

> **Purpose** Prevent accidental system design and framework cargo-culting by forcing decision clarity, constraint, and inspectable artifacts.

---

0. ENTRY CONDITION (Do Not Proceed Without This)

Observable Failure (Required)

> What is visibly going wrong *right now* if no system exists?

- ☐ Strategy confusion
- ☐ Discovery churn
- ☐ Delivery bottleneck
- ☐ Coordination breakdown
- ☐ Evolution / scaling failure

Rule: If you cannot point to a concrete failure, **stop**.

---

1. SYSTEM IDENTITY

System Name (Optional but stabilizing)

> What people will call this when they misuse it.

---

2. DECISION THE SYSTEM OPTIMIZES (NON-NEGOTIABLE)

Primary Decision Type (Choose one):

- ☐ Priority
- ☐ Scope
- ☐ Ownership
- ☐ Sequencing
- ☐ Investment
- ☐ Diagnosis
- ☐ Repair

Test:

> Which decision becomes safer, faster, or harder to avoid?

If none → this is not a system.

---

3. PRIMARY OBJECT OF CONTROL

What the system directly manipulates (choose 1–2 max):

- ☐ Goals
- ☐ Work items
- ☐ Interfaces
- ☐ Domains
- ☐ Constraints
- ☐ Incentives
- ☐ Information flow

Rule: Systems do **not** control outcomes — only objects.

---

4. UNIT OF ANALYSIS

Smallest unit the system is valid for:

- ☐ Individual
- ☐ Team
- ☐ Multi-team
- ☐ Organization
- ☐ Ecosystem / Market

Warning: Scaling beyond this without redesign guarantees failure.

---

5. CAUSALITY MODEL (MATCH OR FAIL)

Dominant causality assumption:

- ☐ Linear planning
- ☐ Feedback loops
- ☐ Constraints & flow
- ☐ Evolution / selection
- ☐ Socio-technical dynamics

Mismatch here invalidates the system.

---

6. ARTIFACTS (INSPECTABILITY RULE)

Concrete outputs the system MUST produce:

- ☐ Map
- ☐ Table
- ☐ Score
- ☐ Vocabulary
- ☐ Contract
- ☐ Canvas
- ☐ Rule set

Rule: If nothing can be inspected, challenged, or revised → no system exists.

---

7. VOCABULARY & BOUNDARY RULES

Terms that MUST be defined precisely:

> (List 3–5 max)

Vague thinking explicitly disallowed:

> (e.g., “alignment”, “value”, “impact” without qualifiers)

---

8. NON-NEGOTIABLE CONSTRAINT (POWER SOURCE)

What the system forbids or forces:

- ☐ Timebox
- ☐ Scope limit
- ☐ Participation rule
- ☐ Sequence rule
- ☐ Authority boundary

Rule: A system that allows everything enforces nothing.

---

9. FAILURE & MISUSE MODEL (REQUIRED)

How this system breaks when misapplied:

- ☐ Used as justification
- ☐ Turned into ritual
- ☐ Scaled incorrectly
- ☐ Vocabulary without enforcement
- ☐ Tool-first adoption

If misuse cannot be named, it will dominate usage.

---

10. ADOPTION PATH (REALISTIC)

Who can use this first, successfully:

- ☐ Solo
- ☐ Facilitated team
- ☐ Leadership only
- ☐ Org-wide mandate

Time to first value:

- ☐ < 1 hour
- ☐ 1–2 days
- ☐ 1–2 weeks
- ☐ Structural change required

EXIT TEST (MANDATORY)

Answer **YES** to all:

- ☐ Failure is observable
- ☐ Decision optimized is explicit
- ☐ Object of control is narrow
- ☐ Artifact is inspectable
- ☐ Constraint is enforced
- ☐ Misuse is predictable

If any box is unchecked → **do not deploy**.

Core Law (Memorize This)

> **Every system is a decision machine with a failure model. If you can't name both, you're designing folklore.**

Toolkit — Glossary

Adoption path

A system's realistic entry into the organization: who can run it first, what minimum viable use looks like, how value appears quickly, and what redesign is required to scale.

Artifact

A concrete output a system produces that makes thinking and decisions inspectable (e.g., decision log entry, ownership map, interface contract, allocation table). If there is no artifact, there is no system—only conversation.

Artifact explosion

A scale failure mode where artifacts multiply faster than decisions improve, and teams spend time reconciling representations instead of acting.

Authority boundary

The explicit limit of who can decide, enforce constraints, approve changes, or escalate. Systems without authority boundaries rely on persuasion and drift under stress.

Authority mismatch

A failure mode where a system requires enforcement power that the unit running it does not have, producing superficial compliance and governance inflation.

Boundary rules

The system's explicit "no": what it refuses to do, what vague language it disallows, what behaviors it forbids, and what must be true to proceed.

Busy-team misuse

A predictable misuse mode where time pressure causes teams to weaken constraints, skip artifacts, or convert the system into a checkbox ritual.

Causality model

The system's embedded belief about how actions produce outcomes. Common models: linear planning, feedback loops, constraints & flow, evolution/selection, socio-technical dynamics. Model mismatch invalidates systems.

Collision

When two systems optimize the same decision differently (or control the same object differently), causing conflict, drift, and "which system wins?" ambiguity.

Compliance theater

When a system's visible motions are performed to signal control, while decisions and outcomes remain unchanged.

Constraint

A rule that forbids or forces behavior to create real tradeoffs and prevent decision avoidance. Constraints must be enforceable and should include explicit defaults.

Constraint ladder

Strength levels of constraints, from weak to strong: norm → policy → rule → gate → automatic default.

Constraints & flow

A causality model focused on bottlenecks, queues, WIP, and throughput. Fits delivery/operations problems where work doesn't move.

Cost cap

A governance rule that limits how much time/effort a system may consume before it must justify itself or be redesigned/retired.

Default

The system's automatic behavior when people avoid a decision or violate a constraint. Defaults make avoidance expensive and keep systems functional under pressure.

Decomposition

A method for reconstructing a system's operating logic across the canonical dimensions so it can be evaluated, compared, adapted, or rejected.

Decomposition table

The primary artifact of decomposition: a filled specification of the system across the ten canonical dimensions.

Decision

A commitment that changes what happens next (priority, scope, ownership, sequencing, investment, diagnosis, repair). Discussion is not a decision.

Decision clarity

The property of a system where participants can name the optimized decision and produce consistent decision outputs.

Decision log

An artifact that records a decision with context, options, rationale, owner, and review date. Useful for preventing reversals, blame cycles, and memory loss.

Decision machine

A system understood operationally: it takes ambiguity as input, applies rules/constraints, and outputs a decision that commits action. If no decision improves, the system is ornamental.

Decision type

The primary category of decision a system optimizes: priority, scope, ownership, sequencing, investment, diagnosis, repair.

Drift

The predictable degradation of a system over time due to incentives, convenience, turnover, and political pressure. Systems must be designed with drift in mind.

Enforcement

The mechanism that makes constraints real: who enforces, how, and what happens when rules are violated or decisions are avoided.

Evolution / selection

A causality model where progress emerges through variation, selection, and retention over time. Fits scaling, innovation portfolios, and ecosystem dynamics.

Failure anchoring

The discipline of grounding a system in an observable failure rather than abstract aspirations like “alignment” or “clarity.”

Failure map

A compact description of where a failure appears (strategy/discovery/delivery/cooperation/evolution), how it manifests, and what it costs.

Feedback loops

A causality model where truth is learned through action, observation, and adaptation. Fits discovery, strategy under uncertainty, experimentation.

Framework stacking

A failure mode where new systems are added on top of old systems rather than replacing or subordinating them, creating collisions and artifact overload.

Gap (blind spot)

A recurring failure or decision domain not owned by any system in the landscape, leading to ad hoc behavior and recurring surprises.

Inspectability

The property of a system where its artifacts and outputs can be examined, challenged, compared over time, and used to make subsequent decisions.

Interface

The boundary where two units interact (teams, services, domains). Interface clarity often matters more than internal team rituals at multi-team scale.

Interface contract

An artifact defining how an interface behaves and is governed (ownership, expectations, change protocol, SLAs where relevant).

Kill criteria

Explicit conditions under which an initiative, system, or experiment must stop or be retired. Prevents endless accumulation.

Landscape (system landscape)

The set of systems—formal and informal—that shape decisions in an environment, including processes, governance, tools, and cultural defaults.

Landscape matrix

The primary artifact for system landscape identification: a table listing systems and their decision type, unit of analysis, object of control, artifacts, constraints, and misuse modes.

Legibility

How easily leadership or outsiders can “read” what is happening. Useful, but dangerous when it replaces truth and drives reporting over decisions.

Legibility over truth

A failure mode where artifacts are optimized to look stable and understandable, causing bad news to travel slowly and surprises to appear late.

Linear planning

A causality model assuming sufficient predictability to plan sequences and expect them to hold. Fits stable, repeatable work.

Metric capture

A failure mode where metrics become targets or weapons, driving gaming and hiding reality. Measurement becomes an incentive system rather than a learning system.

Minimal Viable System (MVS)

The smallest system that can reliably improve a real decision under real constraints. Core formula: **one decision + one artifact + one constraint + one default** (plus misuse warning and adoption path).

Misuse model

A prediction of how a system will be gamed, ritualized, captured, or degraded—plus mitigations. Misuse is part of design, not an afterthought.

Non-negotiable rule

The constraint that gives a system power—what it forbids or forces even when inconvenient.

Object of control

What a system directly manipulates (e.g., work items, interfaces, constraints, information flow). Systems do not control outcomes directly; they control objects that influence outcomes.

Observable failure

A repeated breakdown that can be witnessed and has concrete consequences. The required anchor for system design and selection.

Observable Failure Statement

A 3–5 sentence artifact describing: situation, recurring symptom, consequence/cost, who is impacted, and frequency. The entry condition for system work.

Operating mode

How a system runs in practice: one-off vs continuous, slow strategic vs fast operational, facilitation-heavy vs solo-usable, cadence/event triggers.

Ownership

A decision domain about responsibility and authority: who owns what, who decides, who maintains, who approves changes, and how disputes escalate.

Precedence rule

A rule that resolves collisions by declaring which system's artifact/decision wins when systems overlap.

Problem frame

The category of failure the system addresses: strategy, discovery, delivery, cooperation, evolution/scaling. Frames determine appropriate causality assumptions and artifacts.

Repair

A decision domain about structural fixes: what changes remove constraints or recurring failures, and how capacity is protected for that work.

Review & validation

A governance practice that ends with one of four outcomes: keep, modify, subordinate, remove—based on evidence of decision improvement, enforceability, and misuse.

Scale collapse

Failure caused by applying a system at the wrong unit of analysis (e.g., team practice mandated org-wide) without redesign of artifacts, authority, and constraints.

Selection pressure

The force that causes winners to persist and losers to stop (e.g., kill criteria, gates, budget limits). Without selection pressure, “evolution” becomes sprawl.

Socio-technical dynamics

A causality model where outcomes are shaped by incentives, authority, trust, identity, and power interacting with technical reality. Fits cooperation and governance failures.

Source of truth

The artifact (or artifact set) that is authoritative for a decision domain. Multiple sources create collision and translation work.

Subordinate

A review outcome: keep a system but clarify it is secondary to another system (via precedence rules) to prevent collisions.

Sunset clause

A rule that a system (or constraint) expires unless renewed based on evidence. Prevents fossilization.

System

An engineered construct that reduces a specific observable failure by optimizing a specific decision through controllable objects, producing inspectable artifacts, under enforceable constraints, with a misuse model.

System Contract

The required specification for deliberate system invention: target situation, observable failure, root-cause assumption, object of control, decision optimized, artifacts, constraint + default, misuse model, adoption path.

System shape

A common structural pattern of systems (e.g., diagnostic, allocation, boundary, flow-control, selection). Shapes help keep systems coherent and minimal.

Unit of analysis

The smallest boundary where the system’s logic is valid and enforceable: individual, team, multi-team, organization, ecosystem/market.

Vocabulary substitution

A failure mode where words replace decisions (“alignment” replaces priority, “clarity” replaces scope). Creates debate without commitment.

WIP (Work in Progress)

Work started but not finished. Uncontrolled WIP creates queues, context switching, and long cycle times; controlling WIP is a common flow constraint.

“We’ll revisit”

A common decision-avoidance escape hatch. Systems counter it with constraints and explicit defaults.

Toolkit — Worksheets

These worksheets are designed to be copied into markdown, filled in collaboratively, and treated as living artifacts. They are intentionally short. If you need more than one page to fill one out, you are likely over-scoping the system.

Worksheet 1 — Observable Failure Statement

Rule: no system work without this.

Fill-in

Situation (where/when):

Recurring symptom (what happens repeatedly):

Consequence / cost (what it causes):

Who is impacted:

Frequency / duration:

Validation checks

- ☐ An outsider could witness it
- ☐ It is not written as an aspiration (“we need alignment…”)
- ☐ It contains at least one concrete consequence
- ☐ It implies a decision that keeps failing

Worksheet 2 — Problem Frame Selection

Choose the dominant failure location.

- ☐ **Strategy:** we fail to choose what matters (priority/investment/scope drift)
- ☐ **Discovery:** we fail to learn what’s true (assumptions persist untested)
- ☐ **Delivery:** we fail to move work (flow, predictability, quality)
- ☐ **Cooperation:** we fail across boundaries (ownership, interfaces, dependency friction)
- ☐ **Evolution:** we fail to adapt/scale (drift, fragmentation, governance inflation)

Evidence supporting the chosen frame:

Secondary frame (if any):

Worksheet 3 — Decision Type Declaration

Rule: pick one primary decision type.

Primary decision optimized:

- ☐ Priority
- ☐ Scope
- ☐ Ownership
- ☐ Sequencing
- ☐ Investment
- ☐ Diagnosis
- ☐ Repair

Write the decision as a sentence:

“We are trying to make the _____ decision safer/faster/harder to avoid.”

How we currently avoid this decision:

What “better” looks like (observable):

Worksheet 4 — Object of Control Selection

Choose 1–2 max.

Primary object(s) of control:

- ☐ Goals
- ☐ Work items
- ☐ Interfaces
- ☐ Domains
- ☐ Constraints
- ☐ Incentives
- ☐ Information flow

For each chosen object:

Object:

Who can change it (authority):

How often it changes (cadence/event):

Why changing it should reduce the failure (assumption):

Worksheet 5 — System Landscape Matrix (Lightweight)

List the systems actually shaping decisions today.

For each system:

System name:

Primary decision type it optimizes:

Lifecycle location: strategy / discovery / delivery / cooperation / evolution

Unit of analysis: individual / team / multi-team / org / ecosystem

Object(s) of control:

Artifact produced:

Constraint / enforcement mechanism:

Known misuse / failure mode:

Landscape outputs

Collisions (at least 2):

*

Blind spot (at least 1):

What should be removed/subordinated before adding anything new:

Worksheet 6 — System Decomposition Table

Use this to evaluate any existing framework or process.

One-sentence system spec

“System X reduces __ failure by optimizing _ *decisions through control of , producing artifacts, enforced by constraints, at the* ___ unit of analysis.”

Dimension fill-in

1) Problem frame (target failure + location):

2) Primary object of control:

3) Unit of analysis:

4) Causality model: linear / feedback / flow / evolution / socio-technical

5) Decision type optimized (primary):

6) Artifacts produced (inspectable outputs):

7) Vocabulary & boundary rules (what must be precise; what is disallowed):

8) Operating mode (cadence, triggers, facilitation needs):

9) Failure & misuse model (3 predictable misuses):

* *

10) Adoption path (who first; minimal viable use; time to first value):

Fit decision

- ☐ Adopt
- ☐ Adapt
- ☐ Subordinate
- ☐ Reject

Reason (1–3 sentences):

Worksheet 7 — System Contract (Deliberate Invention)

Rule: if any field is missing, the system is invalid.

1) Target situation:

2) Observable failure:

3) Root-cause assumption:

4) Object(s) of control (1–2):

5) Decision optimized (primary):

6) Artifact(s) produced:

7) Non-negotiable constraint + default:

- Constraint:
- Default when violated:

8) Misuse warnings + mitigations (at least 3):

- Misuse:
- Mitigation:
- Misuse:
- Mitigation:
- Misuse:
- Mitigation:

9) Adoption path:

- Who can use it first:
- Minimal viable use (smallest run):
- Time to first value (what changes quickly):
- Scaling note (what must change to scale):

Worksheet 8 — Minimal Viable System Builder

Core formula: one decision + one artifact + one constraint + one default.

Decision (type + sentence):

Artifact (name + required fields):

Constraint (rule):

Default (automatic outcome if avoided):

Misuse warning (most likely):

Mitigation (change artifact/constraint/authority):

First run plan (who/when/where):

Expected visible improvement after 1–2 cycles:

Worksheet 9 — System Review Scorecard

Review outcome must be one of: keep / modify / subordinate / remove.

System under review:

Original target failure (still present?):

- Evidence:

Decision clarity (0–2):

- 0 = unclear; 1 = somewhat; 2 = explicit and consistent
- Score:

Artifact inspectability (0–2):

- 0 = missing/unused; 1 = inconsistent; 2 = inspectable + used
- Score:

Constraint enforcement (0–2):

- 0 = optional; 1 = partial; 2 = enforced + defaults trigger
- Score:

Misuse resistance (0–2):

- 0 = dominated by misuse; 1 = mixed; 2 = misuse managed
- Score:

Landscape compatibility (0–2):

- 0 = colliding; 1 = some overlap; 2 = clear precedence
- Score:

Total (0–10):

Decision: keep / modify / subordinate / remove

Change list (if modify/subordinate):

Retirement plan (if remove):

Worksheet 10 — Replace vs Stack Gate

Use this before introducing any new system.

New system name (proposed):

Owned decision (primary):

Artifact that becomes source of truth:

What existing system will be removed, weakened, or subordinated:

Collision risk if we do nothing:

Sunset/review date for the new system:

Toolkit — Facilitation Script

This is a field-ready script for running a 60–90 minute session to diagnose a failure and either adapt an existing system or design a Minimal Viable System (MVS). It is designed to prevent the most common failure mode: producing conversation and artifacts without decisions.

You can run this with a whiteboard and a shared markdown doc.

Preconditions

Do not run this session unless:

- You have at least one real, recurring failure people can point to.
- Someone present can enforce at least one constraint within the unit of analysis.

Roles (lightweight):

- **Facilitator:** runs prompts and enforces timeboxes
- **Scribe:** writes artifacts live (shared doc)
- **Decider (optional but ideal):** person/team with authority for the target decision
- **Participants:** people closest to the failure and the object of control

Hard rule: if you can't name a decision output by minute 25, stop and re-scope.

Session Outputs (Artifacts)

By the end, you must produce:

1. Observable Failure Statement
2. Dominant problem frame + decision type
3. Object of control selection
4. One of:
5. Decomposition table (if adopting/adapting), or
6. System Contract + MVS builder (if inventing)
7. Constraint + default (non-negotiable)
8. Misuse warning + mitigation
9. Next-run plan (who/when/where)

If any output is missing, the session did not complete.

60–90 Minute Agenda

0) Setup (0–5 minutes)

Facilitator says:

- "We are here to improve a specific decision by designing or adjusting a system. We will not leave without a decision output."
- "If we can't name an observable failure, we stop."
- "If we can't name a decision output, we stop."
- "Artifacts are mandatory. Constraints are mandatory."

Scribe opens a doc with headings:

- Failure statement
- Frame + decision type
- Object of control
- Candidate systems (optional)
- MVS (artifact + constraint + default)
- Misuse + mitigation
- Next run

---

1) Failure anchoring (5–15 minutes)

Prompt sequence (write answers verbatim):

1. “What is repeatedly going wrong?”
2. “Give two recent examples with dates or incidents.”
3. “What does it cost?” (time, risk, money, morale, customer impact)
4. “Who is impacted?”
5. “How often does it happen?”

Scribe writes the Observable Failure Statement (3–5 sentences).

Stop rule: if participants can’t cite concrete examples, stop and assign observation tasks.

---

2) Frame selection (15–20 minutes)

Facilitator:

- “Which frame is dominant: strategy, discovery, delivery, cooperation, or evolution?”

Ask:

- “If we solved only one thing, which frame would matter most?”

Record:

- dominant frame
- secondary frame (optional)
- one supporting piece of evidence

---

3) Decision type declaration (20–30 minutes)

Facilitator:

- “Which decision are we failing to make repeatedly?”

Force one primary decision type:

- priority / scope / ownership / sequencing / investment / diagnosis / repair

Prompts:

- "If this session succeeds, what decision becomes easier next week?"
- "What decision keeps getting avoided or re-litigated?"
- "What decision gets forced only by urgency?"

Write:

- decision sentence: "We are optimizing the _____ decision."

Stop rule: if the group insists the goal is "alignment," translate it into a decision type or stop.

---

4) Object of control selection (30–40 minutes)

Facilitator:

- "What can we actually control that influences this failure?"

Choose 1–2 objects max:

- goals / work items / interfaces / domains / constraints / incentives / information flow

Prompts:

- "What would we edit on Monday to influence this?"
- "Who has authority to change it?"
- "How often does it change?"

Write:

- chosen object(s)
- authority holder(s)

Stop rule: if the object requires changing incentives/org structure and nobody present can do it, choose a different object or redesign the unit of analysis.

---

5) Choose path: adopt/adapt vs invent (40–45 minutes)

Facilitator asks two questions:

1. "Is there an existing system already in play that claims to solve this decision?"
2. "Is the current landscape colliding (two systems owning the same decision) or is this a gap (no owner)?"

Decision:

- **Path A: Adopt/Adapt** (decompose a candidate system)
- **Path B: Invent** (design an MVS using the System Contract)

If uncertain, default to **decomposition first**.

Path A — Adopt or Adapt (45–70 minutes)

A1) One-sentence system spec (45–50)

Prompt:

- "In one sentence: what does this system do?"

Use:

- "System X reduces **__ failure by optimizing _ *decisions through control of , producing artifacts, enforced by constraints, at the* ___** unit of analysis."

If you can't write this, you don't understand it yet.

A2) Decompose quickly (50–65)

Fill the 10 dimensions at high speed:

- problem frame
- object of control
- unit of analysis
- causality model
- decision type
- artifacts
- boundary rules
- operating mode
- misuse model
- adoption path

Focus prompts:

- "What artifact does it *actually* produce?"
- "What happens when people avoid it?"
- "How does it get gamed?"
- "Does its causality model match our failure?"

A3) Fit decision (65–70)

Choose one:

- adopt / adapt / subordinate / reject

If adapt, write the changes as a delta:

- artifact change
- constraint/default change
- authority boundary change
- cadence/trigger change

Then jump to sections: Constraint + Misuse + Next run.

Path B — Invent an MVS (45–80 minutes)

B1) System Contract (45–60)

Fill in:

- target situation
- observable failure (already)
- root-cause assumption
- object of control (already)
- decision optimized (already)
- artifact(s)
- constraint + default
- misuse warnings + mitigations
- adoption path

Facilitator prompts to keep it narrow:

- “What is the smallest artifact that makes the decision inspectable?”
- “What constraint forces the tradeoff?”
- “What default triggers when we avoid it?”

B2) Minimal Viable System builder (60–70)

Write the MVS in one block:

- Decision:
- Artifact (fields):
- Constraint:
- Default:
- Misuse:
- Mitigation:

If you can’t state it in 10 lines, it’s not minimal.

B3) First-run plan (70–80)

Write:

- “We will run this on **_ *(date) with*** _ (people).”
- “The artifact will live at ____.”
- “The decider is ____.”
- “The default triggers when ____ happens.”
- “We expect to see **_ *change within*** _ cycles.”

6) Constraint + default (mandatory checkpoint) (end of either path)

Facilitator asks:

- “How will people avoid this decision?”
- “What rule blocks avoidance?”
- “What happens automatically if they still avoid it?”

Write:

- constraint
- default
- who enforces it

Stop rule: if there is no enforceable constraint, you do not have a system.

---

7) Misuse rehearsal (mandatory checkpoint)

Run three rehearsals (5 minutes total):

1. **Busy team:** “How will teams weaken this to save time?”
2. **Leadership:** “How will this be turned into reporting/control?”
3. **Politics:** “How will someone use this as a shield/veto/weapon?”

For each misuse, write one mitigation by changing:

- artifact, or
- constraint/default, or
- authority boundary

---

8) Close: commit to a real run

End with a decision output:

- “Keep / modify / subordinate / remove” (if reviewing)
- or “Run the MVS once” (if inventing)

Write the commitment:

- Date/time:
- Participants:
- Artifact link/location:
- Success criterion (observable change after 1–2 cycles):

Final rule: if there is no scheduled first run, the session produced theory, not a system.

Optional 90-minute extensions

Use only if needed.

Extension 1 — Landscape collisions (add 10 minutes)

Prompt:

- “Which existing system’s artifact will this override or replace?”
- “What’s the precedence rule?”

Extension 2 — Sunset clause (add 5 minutes)

Prompt:

- “When will we review this system?”
- “What evidence renews it?”
- “What kills it?”

Toolkit — System Fitness Checklist

Use this checklist to evaluate any system (framework, process, governance mechanism, operating routine) on a monthly or quarterly cadence. The goal is not to defend or attack systems—it is to decide **keep / modify / subordinate / remove** based on evidence.

This checklist is intentionally short. If you can't evaluate a system quickly, you probably don't have an inspectable system—you have an ambient ritual.

How to Use

1. Pick one system.
2. Bring the last **3 artifacts** it produced (or admit there are none).
3. Score it across five dimensions (0–2 each).
4. Decide: **keep, modify, subordinate, remove**.
5. If modifying, choose one lever: artifact, constraint/default, authority boundary, cadence/trigger, unit-of-analysis scope.

The Fitness Scorecard (0–10)

1) Problem fit (0–2)

Question: Is the system still aimed at the real, dominant failure?

- **0:** The target failure is unclear, obsolete, or not evidenced.
- **1:** The failure exists but is not clearly dominant, or evidence is mixed.
- **2:** The failure is observable, evidenced, and still dominant.

Evidence to check:

- recurring incidents, missed commitments, escalation logs, queues, reversals

Score: **__ / 2**

2) Decision clarity (0–2)

Question: Can people name the decision this system optimizes, and does it produce decisions?

- **0:** Decision type is unclear; outputs are discussions.
- **1:** Decision type is sometimes clear; outputs inconsistently commit action.
- **2:** Decision type is explicit; outputs reliably produce commitments.

Test prompt:

- "Which decision became safer/faster/harder to avoid because this system ran?"

Score: **__ / 2**

3) Artifact inspectability (0–2)

Question: Does the system produce artifacts that can be challenged and reused?

- **0:** No consistent artifact, or artifacts exist but are unused.
- **1:** Artifacts exist but are inconsistent, ambiguous, or not referenced later.
- **2:** Artifacts are consistent, inspectable, and used in future decisions.

Artifact checks:

- includes owner
- includes decision change
- includes assumptions/constraints where relevant
- lives in a known “source of truth” location

Score: **__ / 2**

---

4) Constraint enforcement (0–2)

Question: Does the system have teeth (constraints + defaults) and are they enforced?

- **0:** Constraints are optional; enforcement is reminders/shame; defaults don’t exist.
- **1:** Some constraints are enforced but exceptions are frequent or invisible.
- **2:** Constraints are enforced; defaults trigger; exceptions are explicit and rare.

Test prompt:

- “What happens when people avoid the decision or skip the artifact?”

Score: **__ / 2**

---

5) Misuse resistance (0–2)

Question: Is misuse recognized and mitigated, or is it dominating usage?

- **0:** The system is dominated by misuse (reporting theater, ritual, gaming, capture).
- **1:** Misuse exists; mitigations are partial or informal.
- **2:** Misuse is anticipated; mitigations exist in artifacts/constraints/authority.

Misuse lenses:

- busy-team weakening
- leadership reporting/control pressure
- political capture (veto/shield/weapon)

Score: **__ / 2**

Total Score

Total: __ / 10

Decision Rule

Choose one outcome (mandatory):

- **8–10 → Keep** System is fit, enforced, and producing decisions.
- **5–7 → Modify** System is partially working; change one lever (see below).
- **3–4 → Subordinate or Redesign** System likely collides with others, lacks enforcement, or targets the wrong failure.
- **0–2 → Remove** System is ritual/theater or obsolete. Retire it; replace only if a real gap remains.

Decision: keep / modify / subordinate / remove

Modification Levers (Pick One)

If you chose **modify**, choose exactly one primary lever first.

Lever A — Artifact upgrade

Use when:

- artifacts exist but don't drive decisions

Moves:

- add "decision changed" field
- add owner + timestamp
- add explicit next action
- reduce artifact scope (make it harder to fake)

Lever B — Constraint + default

Use when:

- decisions are avoided or deferred

Moves:

- add timebox + default outcome
- add scope/WIP cap + displacement rule
- add gate ("cannot proceed unless…")

Lever C — Authority boundary

Use when:

- system relies on consensus or persuasion

Moves:

- define decider
- define consult vs inform
- define escalation path + default resolution

Lever D — Cadence / trigger change

Use when:

- system runs too often (ritual) or too rarely (stale)

Moves:

- shift from time-based to event/threshold-based triggers
- shorten cycles to increase learning
- remove meetings unless artifact is produced

Lever E — Unit-of-analysis correction

Use when:

- system was copied across scales and collapsed

Moves:

- redesign artifacts for the correct scale (interfaces/contracts at multi-team)
- reduce scope to where enforcement exists
- add precedence rules for cross-scale coordination

Collision & Landscape Check (Fast)

Answer these every time:

- **Does this system collide with another system's artifact?**
- ☐ Yes ☐ No If yes: name the other system and define a precedence rule or subordinate one.
- **Is it duplicating a decision already "owned" elsewhere?**
- ☐ Yes ☐ No

- **What would we remove if we keep this?**

Retirement Checklist (If Removing)

If you chose **remove**, do these three steps to avoid chaos:

1. **Name what decision it was supposed to optimize:**
2. **Name what artifact (if any) will remain as historical record:**
3. **Name what replaces it, or explicitly accept the gap for now:**

Add a **sunset date** if you're not removing immediately:

- Sunset date:
- Owner of retirement:

“Healthy System” Quick Tell

A healthy system has:

- a named failure
- a named decision
- an inspectable artifact
- an enforced constraint with a default
- known misuse modes and mitigations

If you can’t say all five in two minutes, the system is not healthy.

Toolkit — Anti-Pattern Playbook

This playbook is a set of **field countermeasures** for common system failures. Each entry is structured for fast use:

- **What you'll see** (symptoms)
- **What's really happening** (mechanism)
- **Fast test** (how to confirm)
- **Countermove** (artifact / constraint / authority / cadence)
- **Default** (what to do if people avoid the fix)

Use it in reviews, retro-style sessions, and whenever "process" starts feeling heavy but outcomes don't improve.

Anti-Pattern 1 — "Alignment" as a Goal

What you'll see

- "We need alignment" becomes the problem statement.
- Meetings increase; decisions don't.
- People leave with different interpretations.

What's really happening

A failing decision type is being disguised to avoid conflict.

Fast test

Ask: "Alignment about **which decision**?" If nobody can answer in one sentence, it's not alignment—it's avoidance.

Countermove

- **Decision:** force a decision type (priority/scope/ownership/…)
- **Artifact:** create a decision log entry with "Decision changed" field
- **Constraint:** timebox the decision + default outcome

Default

If the decision isn't named in 10 minutes, stop the meeting and assign an owner to produce an Observable Failure Statement.

Anti-Pattern 2 — Ritual Without Artifact

What you'll see

- Recurring meetings exist "because we always do them."
- Outcomes are verbal; nothing persists.
- The same topics reappear weekly.

What's really happening

Cadence is substituting for a decision machine.

Fast test

Ask: “Show me the last 3 artifacts this meeting produced.” If there are none, it’s a ritual.

Countermove

- **Artifact:** define a single required output (one page / one table)
- **Constraint:** “No artifact, no meeting”
- **Cadence:** shorten or remove until artifact exists

Default

If artifact isn’t produced, automatically cancel the next occurrence and run async.

Anti-Pattern 3 — Artifact Without Decision

What you’ll see

- Dashboards/slides/docs are produced regularly.
- People feel “informed,” but actions don’t change.
- Artifacts are optimized for polish.

What’s really happening

Reporting is replacing decision-making.

Fast test

Ask: “Which decision changed because of this artifact?” If there’s no answer, it’s theater.

Countermove

- **Artifact change:** add mandatory fields:
- “Decision changed”
- “Owner”
- “Next action + date”
- **Constraint:** artifact must contain a decision or it’s invalid

Default

If no decision is recorded, the system defaults to the previous decision (explicitly) and escalates the missing decision to the decider.

Anti-Pattern 4 — Constraintless “Guidelines”

What you’ll see

- “It’s just guidance.”
- Exceptions are constant.
- Under stress, the system vanishes.

What's really happening

There is no enforcement mechanism; the system is optional.

Fast test

Ask: "What happens when someone ignores this?" If the answer is "nothing," it's not a system.

Countermove

- **Constraint:** add one enforceable rule
- **Default:** define automatic behavior when ignored
- **Authority:** name who enforces it

Default

If enforcement authority can't be named, redesign the system to a smaller unit of analysis where authority exists.

Anti-Pattern 5 — Framework Stacking

What you'll see

- New systems are added; none are removed.
- Multiple sources of truth exist for the same decision.
- Teams spend time translating between systems.

What's really happening

Systems collide or duplicate decision ownership.

Fast test

Ask: "Which artifact is the source of truth for priority/scope/etc.?" If there are two, you have a collision.

Countermove

- **Precedence rule:** declare which system wins for that decision domain
- **Removal/subordination:** retire or subordinate one system
- **Landscape matrix:** map collisions explicitly

Default

No new system may launch until it names what it replaces/subordinates and where the single source artifact will live.

Anti-Pattern 6 — Consensus Veto Everywhere

What you'll see

- Decisions take too long.
- "We need more buy-in" repeats.
- The most risk-averse actor controls outcomes.

What's really happening

Consensus creates distributed veto power and rewards avoidance.

Fast test

Ask: "Who decides if we disagree?" If unclear, you don't have decision rights.

Countermove

- **Authority boundary:** name the decider
- **Participation rule:** consult vs inform
- **Constraint:** timeboxed decision window + default outcome
- **Artifact:** decision log with dissent recorded

Default

If the group can't agree on a decider, escalate that *as the decision* to leadership immediately (ownership is the blocker).

Anti-Pattern 7 — Metric Capture

What you'll see

- Metrics improve; reality worsens.
- People hide work to protect numbers.
- Metrics become performance weapons.

What's really happening

Measurement became an incentive system.

Fast test

Ask: "Who gets punished or rewarded by this metric?" If the answer is "individuals/teams," gaming is predictable.

Countermove

- Separate:
- **learning metrics** (diagnosis)
- **evaluation metrics** (performance)
- Add "decision link":
- metrics must map to an action decision log entry

Default

If a metric is used for evaluation, it must have an explicit anti-gaming review and be paired with at least one qualitative truth check (incidents, audits, samples).

Anti-Pattern 8 — Legibility Over Truth

What you'll see

- Status looks green until failure is unavoidable.
- Bad news travels slowly.
- Artifacts are sanitized.

What's really happening

Artifacts optimize safety/politics over reality.

Fast test

Ask: "What does this artifact make harder to say?" If uncertainty and risk aren't representable, truth is being filtered.

Countermove

- **Artifact:** require fields for:
- uncertainty
- risks with owners
- assumptions
- "what would change our mind"
- **Constraint:** red/yellow flags require next action + date

Default

If uncertainty can't be written, the artifact is invalid and must be replaced by a simpler decision-focused artifact.

Anti-Pattern 9 — "Everything Is Urgent"

What you'll see

- Priorities shift constantly.
- Work starts but doesn't finish.
- Teams are always "busy," outcomes lag.

What's really happening

No enforced prioritization and uncontrolled WIP.

Fast test

Ask: "What are the top 3 priorities and what are we not doing?" If "not doing" is empty, priority is fake.

Countermove

- **Constraint:** WIP limit or initiative cap
- **Default:** new work displaces lowest-priority in-progress work
- **Artifact:** ranked stack with capacity allocation

Default

If leadership introduces urgent work, they must name what it displaces (explicit tradeoff rule).

Anti-Pattern 10 — Ownership Fog (“Not My Problem”)

What you’ll see

- Work gets blocked on dependencies.
- Escalations replace collaboration.
- Interfaces and responsibilities are disputed.

What’s really happening

Ownership and interfaces are the real object of control, but the system is trying to control “communication.”

Fast test

Ask: “Who owns this interface/domain and what authority do they have?” If unclear, you have boundary failure.

Countermove

- **Artifact:** ownership map + interface contract
- **Constraint:** every interface has an owner; changes require owner approval
- **Authority:** escalation path defined

Default

If ownership is disputed, default ownership goes to the team operating it in production until reassigned by a named authority.

Anti-Pattern 11 — Hero-Dependent Systems

What you’ll see

- One person makes the system “work.”
- Everyone says “ask Alex.”
- Progress stalls when the hero is absent.

What’s really happening

Interpretation and enforcement live in a person, not in artifacts and defaults.

Fast test

Ask: “Could a new hire run this system from artifacts alone?” If not, it’s hero-dependent.

Countermove

- Move rules into:
- decision logs
- explicit constraints and defaults
- ownership map

- Reduce reliance on tacit knowledge

Default

If a step requires a person's memory, treat it as a missing artifact and stop the workflow until it's captured.

Anti-Pattern 12 — Fossilized Systems (Can't Die)

What you'll see

- Nobody remembers why the system exists.
- Removing it feels dangerous.
- The system survives via identity.

What's really happening

No review cadence and no retirement mechanism.

Fast test

Ask: "What failure was this designed to prevent?" If unclear or obsolete, it's fossilized.

Countermove

- Add:
- sunset clause
- review cadence
- kill criteria (evidence-based)
- Run the System Fitness Checklist

Default

If the system can't justify itself within one review cycle, schedule retirement and protect a replacement only if a proven gap remains.

Fast "What Should We Change First?" Rule

When a system is failing, change in this order:

1. **Constraint + default** (teeth)
2. **Artifact** (inspectability)
3. **Authority boundary** (enforcement)
4. **Cadence/trigger** (operational fit)
5. **Unit of analysis** (scale validity)

If you start with cadence or templates, you usually get theater.

Mini Index (Anti-Pattern → Countermove)

- Alignment talk → force decision type + timebox + default
- Too many meetings → "no artifact, no meeting"

- Pretty reports → add “decision changed” + next action
- Optional guidelines → add enforceable constraint + default
- Too many frameworks → precedence + remove/subordinate
- Consensus slow → decider + consult/inform + default
- Metrics gamed → split learning vs evaluation + decision link
- Green status lies → uncertainty/risk fields + red requires action
- Everything urgent → WIP/initiative cap + displacement default
- Blocked dependencies → ownership map + interface contract + escalation
- Hero hub → externalize into artifacts + defaults
- Fossilization → sunset clause + kill criteria

Toolkit — Case Studies

These case studies are **illustrative composites**: they're built from common patterns seen across engineering/product organizations, but they are not reports of a single identifiable company. Their purpose is to show how the lens "bites" in practice: turning vague discomfort into decisions, artifacts, constraints, and defaults.

Use them as reference moves, not as prescriptions.

Case Study 1 — "We Need Alignment" That Was Actually an Ownership Failure

Context

- Unit of analysis: multi-team (3 product teams + 1 platform team)
- Domain: shared API used by multiple products

Observable failure statement

Dependency work routinely stalled 1–2 weeks because teams disputed who owned API changes and backward compatibility decisions. Escalations became the default. Releases slipped and platform engineers became a bottleneck for decisions, not implementation.

Lens diagnosis

- Frame: cooperation
- Decision type failing: ownership
- Object of control: interfaces
- Causality model: socio-technical dynamics (authority and boundaries)
- Landscape note: sprint planning existed, but it optimized sequencing *within* teams, not ownership *between* teams

Minimal viable system change

- Artifact: **Interface Ownership Contract** (one page per API)
- Owning team
- Change approval rule
- Compatibility policy
- Escalation path
- SLA for review
- Constraint + default:
- Constraint: "No API change merges without owner approval within 24h."
- Default: "If no response in 24h, change proceeds under 'backward compatible only' rule."

Misuse model + mitigation

- Misuse: owner becomes gatekeeper
- Mitigation: timeboxed SLA + compatibility default limits power capture
- Misuse: contract becomes stale
- Mitigation: quarterly review + sunset clause ("expires unless renewed")

What changed after 2 cycles (expected)

- Fewer escalations, faster API decisions, and disputes moved from “who owns?” to “does this violate compatibility?” (a healthier argument).

Case Study 2 — “We’re Slow” That Was Actually Uncontrolled WIP

Context

- Unit of analysis: team
- Domain: feature delivery with frequent interrupts

Observable failure statement

“Small” tasks routinely took 10–20 days. Engineers had 5–10 in-progress items each. Work piled up in review/QA queues. Releases slipped and quality incidents increased.

Lens diagnosis

- Frame: delivery
- Decision type failing: sequencing (and repair secondary)
- Object of control: constraints (WIP) + work items (size/definition)
- Causality model: constraints & flow

Minimal viable system change

- Artifact: **Single visible queue** with columns and explicit WIP counts
- Constraint + default:
- Constraint: “Max 3 items in progress for the team.”
- Default: “New urgent work displaces the lowest-priority in-progress item (paused, not parallelized).”

Misuse model + mitigation

- Misuse: teams hide work “off-board”
- Mitigation: definition of work includes “anything consuming >30 min/day”
- Misuse: WIP becomes a performance weapon
- Mitigation: treat metrics as learning-only; tie changes to decision logs, not evaluation

What changed after 2 cycles (expected)

- Cycle time dropped; bottlenecks became visible; fewer “busy but not finishing” weeks.

Case Study 3 — OKRs Became Reporting, Not Investment Decisions

Context

- Unit of analysis: organization (portfolio)
- Domain: product + platform investments

Observable failure statement

Quarterly OKRs were written and reviewed, but priorities didn’t change. Teams committed to too many initiatives. Leadership used OKR reviews for status reporting; kill decisions were rare. Work accumulated and strategy felt performative.

Lens diagnosis

- Frame: strategy + evolution
- Decision type failing: investment (and scope secondary)
- Object of control: goals + allocation constraints
- Causality model: feedback loops (learning) + evolution (selection)
- Landscape note: roadmaps and OKRs both claimed priority authority (collision)

Minimal viable system change

- Artifact: **Allocation Table** (portfolio capacity by category)
- Feature / Reliability / Platform / Discovery (percentages)
- Named owners per allocation slice
- “Kill / continue / increase” decision field
- Constraint + default:
- Constraint: “Allocation is fixed for the quarter; initiatives must fit.”
- Default: “If a new initiative enters mid-quarter, it must displace an existing one (named).”

Misuse model + mitigation

- Misuse: OKRs stay as slogans
- Mitigation: goals must link to allocation and a kill/continue decision
- Misuse: leadership adds more goals rather than choices
- Mitigation: initiative cap + forced displacement rule

What changed after 2 cycles (expected)

- Fewer initiatives, clearer tradeoffs, and “kill decisions” became normal rather than taboo.

Case Study 4 — Incident Reviews That Didn’t Produce Repair

Context

- Unit of analysis: team / multi-team (service ownership)
- Domain: reliability and on-call

Observable failure statement

Incidents repeated with similar causes. Postmortems existed, but action items were vague and often not completed. Reliability work lost to feature delivery, and on-call fatigue increased.

Lens diagnosis

- Frame: delivery (operations) + cooperation (handoffs)
- Decision type failing: repair

- Object of control: constraints (capacity protection) + information flow (decision log)
- Causality model: constraints & flow

Minimal viable system change

- Artifact: **Repair Decision Record** (one per incident)
- “What will change?” (single repair decision)
- Owner
- Due date
- Verification method
- Constraint + default:
- Constraint: “Protect 20% capacity for repair until repair backlog < threshold.”
- Default: “If repair capacity is consumed by features, release scope is reduced automatically (predefined cut list).”

Misuse model + mitigation

- Misuse: postmortems become blame narratives
- Mitigation: focus artifact on “next change decision,” not storytelling
- Misuse: repair capacity is silently borrowed
- Mitigation: allocation table visible + displacement rule enforced

What changed after 2 cycles (expected)

- Fewer repeated incidents, and repairs stopped competing invisibly with features.

Case Study 5 — Architecture Review Board as a Bottleneck

Context

- Unit of analysis: multi-team / org
- Domain: design approvals and standards

Observable failure statement

Architecture reviews introduced multi-week delays. Teams prepared large docs for approval, then reworked them after late feedback. The board became a gate without clear criteria, and teams routed around it.

Lens diagnosis

- Frame: cooperation + evolution (standards)
- Decision type failing: sequencing (and ownership of standards)
- Object of control: interfaces + authority constraints
- Causality model: socio-technical dynamics

Minimal viable system change

- Artifact: **Decision Gate Checklist + Interface Contract**
- Only decisions that affect shared interfaces require review
- Standard criteria listed explicitly

- Constraint + default:
- Constraint: “Review SLA is 72h for interface-affecting changes.”
- Default: “If SLA expires, change proceeds if checklist is satisfied; dissent is logged for later audit.”

Misuse model + mitigation

- Misuse: board expands scope to everything
- Mitigation: boundary rule: “No interface impact, no board”
- Misuse: default becomes reckless
- Mitigation: checklist includes blast radius and rollback plan minimums

What changed after 2 cycles (expected)

- Reviews became narrower, faster, and focused on shared interface risk rather than general design style.

Case Study 6 — “Innovation Program” Without Selection Pressure

Context

- Unit of analysis: organization
- Domain: internal innovation bets

Observable failure statement

> Many “pilots” ran indefinitely, consuming capacity with unclear outcomes. Nothing died. Teams accumulated tools and prototypes, creating fragmentation and maintenance burden.

Lens diagnosis

- Frame: evolution / scaling
- Decision type failing: investment (selection/kill)
- Object of control: constraints (kill criteria) + information flow (bet tracker)
- Causality model: evolution / selection

Minimal viable system change

- Artifact: **Bet Tracker**
- Hypothesis
- Success criteria
- Budget/timebox
- Owner
- Review date
- Kill/continue decision field
- Constraint + default:
- Constraint: “Every bet has a 6–8 week timebox and kill criteria.”
- Default: “If success criteria aren’t met at review, the bet is killed and artifacts are archived.”

Misuse model + mitigation

- Misuse: criteria become vague to avoid killing
- Mitigation: criteria must be measurable or observable; otherwise bet cannot start
- Misuse: killing becomes politically unsafe
- Mitigation: normalize killing as success (“saved capacity” tracked and celebrated)

What changed after 2 cycles (expected)

- Fewer pilots, clearer learning, and reduced tool sprawl because selection started happening.

How to Use These Case Studies

When you face a real situation, don’t copy the solution. Copy the move:

1. Write the observable failure statement
2. Choose the dominant frame
3. Name the decision type being optimized
4. Choose 1–2 objects of control
5. Produce one artifact that makes the decision inspectable
6. Add one enforceable constraint + default
7. Predict misuse and mitigate
8. Run one cycle, then review: keep/modify/subordinate/remove

Toolkit — Framework Decompositions

These decompositions are **reference specifications**, not endorsements. Real implementations vary, but the purpose here is stable: make the operating logic inspectable so you can decide **adopt / adapt / subordinate / reject**.

For each framework, you get:

- one-sentence system spec
- the 10 canonical dimensions (compressed)
- common misuse modes (because they dominate real life)

OKRs

One-sentence system spec OKRs reduce strategy drift by optimizing **investment/priority** decisions via control of **goals and information flow**, producing **OKR sets and check-ins**, enforced by **cadence and (sometimes) scope constraints**, at the **org/multi-team** unit.

Canonical dimensions

1. **Problem frame:** strategy (sometimes discovery)
2. **Object of control:** goals; information flow (visibility)
3. **Unit of analysis:** org or multi-team (often misapplied to team-only)
4. **Causality model:** feedback loops (learn/adjust)
5. **Decision type optimized:** investment / priority (primary)
6. **Artifacts:** objective list; key results; check-in notes; (optional) decision log links
7. **Vocabulary & boundary rules:** “objective,” “key result,” “committed vs aspirational” (if used); disallow vague “impact” without measure
8. **Operating mode:** continuous cadence (quarterly + weekly/biweekly check-ins)
9. **Misuse model:** becomes reporting theater; key results become tasks; sandbagging; local optimization
10. **Adoption path:** leadership + 1–2 groups first; value appears when priorities/investments actually change

Common high-leverage adaptation: pair OKRs with an **allocation constraint** (capacity table + forced displacement rule).

Scrum

One-sentence system spec Scrum reduces delivery unpredictability by optimizing **sequencing/scope** decisions via control of **work items**, producing **sprint goals and increments**, enforced by **timeboxed sprints and role/accountability rules**, at the **team** unit.

Canonical dimensions

1. **Problem frame:** delivery
2. **Object of control:** work items (backlog, sprint backlog)
3. **Unit of analysis:** team
4. **Causality model:** feedback loops (inspect/adapt) with mild linear planning inside a sprint
5. **Decision type optimized:** sequencing (primary), scope (secondary)
6. **Artifacts:** sprint goal; backlog; increment; (optional) Definition of Done
7. **Vocabulary & boundary rules:** sprint, backlog, DoD; disallow “done-ish”
8. **Operating mode:** cadence-driven (sprints, reviews, retros)
9. **Misuse model:** sprint becomes a contract; velocity weaponization; ceremonies without decisions; backlog as junk drawer

10. **Adoption path:** single team can start; value appears when commitments stabilize and learning loops work

Common high-leverage adaptation: make **Definition of Done** and **WIP/queue visibility** explicit to prevent “in progress forever.”

Kanban

One-sentence system spec Kanban reduces flow delay by optimizing **sequencing/repair** decisions via control of **constraints (WIP) and work item policies**, producing **workflow visualization and flow metrics**, enforced by **WIP limits and explicit policies**, at the **team or service** unit.

Canonical dimensions

1. **Problem frame:** delivery (operations/service)
2. **Object of control:** constraints; work items
3. **Unit of analysis:** team/service (can extend to multi-team with interface focus)
4. **Causality model:** constraints & flow
5. **Decision type optimized:** sequencing (primary), repair (secondary)
6. **Artifacts:** board/queue; WIP limits; policies; cycle-time metrics
7. **Vocabulary & boundary rules:** WIP, classes of service; disallow hidden work
8. **Operating mode:** continuous pull; event-driven replenishment
9. **Misuse model:** board becomes status wallpaper; WIP limits ignored; metrics used for punishment; work moved off-board
10. **Adoption path:** one team can adopt quickly; value appears when cycle time drops and bottlenecks are visible

Common high-leverage adaptation: add a **default displacement rule** for urgent work (what gets paused).

Domain-Driven Design (DDD)

One-sentence system spec DDD reduces socio-technical coupling and semantic drift by optimizing **ownership** decisions via control of **domains and interfaces**, producing **bounded context maps and ubiquitous language**, enforced by **boundary rules and architectural constraints**, at the **multi-team/org** unit.

Canonical dimensions

1. **Problem frame:** cooperation + evolution/scaling
2. **Object of control:** domains; interfaces; vocabulary (as boundary enforcement)
3. **Unit of analysis:** multi-team / org
4. **Causality model:** socio-technical dynamics + evolution (boundaries evolve)
5. **Decision type optimized:** ownership (primary), repair (secondary via decoupling)
6. **Artifacts:** context map; domain glossary; aggregates boundaries; integration contracts
7. **Vocabulary & boundary rules:** “bounded context,” “ubiquitous language,” “aggregate”; disallow ambiguous shared meanings
8. **Operating mode:** continuous; design-time and change-time
9. **Misuse model:** becomes jargon; over-modeling; “domain” as political territory; ignores operational reality
10. **Adoption path:** start on one painful boundary; value appears when integration friction and rework drop

Common high-leverage adaptation: require **interface contracts + ownership** for each bounded context integration point.

Team Topologies

One-sentence system spec Team Topologies reduces coordination drag at scale by optimizing **ownership/sequencing** decisions via control of **team boundaries and interaction modes**, producing **team maps and interaction contracts**, enforced by **boundary rules and platform constraints**, at the **multi-team/org** unit.

Canonical dimensions

1. **Problem frame:** cooperation + evolution/scaling
2. **Object of control:** interfaces; ownership boundaries; information flow
3. **Unit of analysis:** multi-team/org
4. **Causality model:** socio-technical + evolution
5. **Decision type optimized:** ownership (primary)
6. **Artifacts:** team API/interaction map; responsibility boundaries; service ownership catalog
7. **Vocabulary & boundary rules:** stream-aligned/platform/enabling; disallow “everyone owns everything”
8. **Operating mode:** continuous; org design and change governance
9. **Misuse model:** reorg theater; labels without interface enforcement; platform becomes gatekeeper
10. **Adoption path:** start with one value stream + platform slice; value appears when dependency friction decreases

Common high-leverage adaptation: define **interaction SLAs/defaults** (e.g., response times, escalation).

ITIL Change Enablement (Change Management)

One-sentence system spec Change enablement reduces production risk by optimizing **sequencing/risk** decisions via control of **constraints and approvals**, producing **change records and risk classifications**, enforced by **gates and authority rules**, at the **org/service** unit.

Canonical dimensions

1. **Problem frame:** delivery (operations/risk)
2. **Object of control:** constraints; information flow
3. **Unit of analysis:** org/service
4. **Causality model:** linear planning + socio-technical (governance)
5. **Decision type optimized:** sequencing (primary), repair (secondary)
6. **Artifacts:** change request/record; risk category; approval trail; implementation plan
7. **Vocabulary & boundary rules:** standard/normal/emergency change; disallow unclassified change
8. **Operating mode:** event-driven; gate-based
9. **Misuse model:** bureaucracy; slows safe changes; people route around; approvals become blame shields
10. **Adoption path:** start with high-risk services; value appears when incident rate drops without killing throughput

Common high-leverage adaptation: make **standard changes** truly low-friction and use **defaults** to prevent dead queues.

SRE Incident Management

One-sentence system spec Incident management reduces outage impact by optimizing **diagnosis/repair** decisions via control of **information flow and constraints (roles, timelines)**, producing **incident timelines and action items**, enforced by **roles, severity policies, and post-incident requirements**, at the **team/multi-team** unit.

Canonical dimensions

1. **Problem frame:** delivery (operations) + cooperation
2. **Object of control:** information flow; constraints
3. **Unit of analysis:** team/multi-team (depending on incident scope)
4. **Causality model:** constraints & flow + feedback loops
5. **Decision type optimized:** diagnosis (during), repair (after)
6. **Artifacts:** incident channel/log; timeline; severity classification; postmortem; repair decision record
7. **Vocabulary & boundary rules:** sev levels, incident commander; disallow “no owner” incidents
8. **Operating mode:** event-triggered; high-tempo operational
9. **Misuse model:** blame; postmortems without repair capacity; action items rot; over-classifying as “emergency”
10. **Adoption path:** one service/team can begin; value appears when repeated incidents fall and MTTR improves

Common high-leverage adaptation: protect **repair capacity allocation** with a displacement default.

Shape Up

One-sentence system spec Shape Up reduces delivery thrash by optimizing **scope/sequencing** decisions via control of **work shaping and timeboxes**, producing **pitch documents and shaped bets**, enforced by **fixed cycles and appetite constraints**, at the **team/multi-team** unit.

Canonical dimensions

1. **Problem frame:** delivery (and strategy-lite via bet selection)
2. **Object of control:** work items; constraints (time/appetite)
3. **Unit of analysis:** team/multi-team
4. **Causality model:** constraints & flow + mild evolution/selection (bets)
5. **Decision type optimized:** scope (primary), sequencing (secondary)
6. **Artifacts:** pitches; shaped work boundaries; cycle plan; cool-down notes
7. **Vocabulary & boundary rules:** appetite, shaping, bet; disallow open-ended scope
8. **Operating mode:** cycle-based (bets + cool-down)
9. **Misuse model:** shaping becomes central gate; “appetite” used to underfund; teams skip discovery and learn late
10. **Adoption path:** start with one product group; value appears when scope stays bounded and shipping cadence stabilizes

Common high-leverage adaptation: add explicit **kill/reshape triggers** mid-cycle when assumptions break.

How to Use This Page

1. Pick the framework someone is proposing.
2. Compare its **decision type**, **object of control**, **unit of analysis**, and **causality model** to your observable failure.
3. If there’s mismatch, **adapt** (change artifact/constraint/authority) or **reject**.
4. If it collides with an existing system, define a **precedence rule** or **subordinate** one.

Toolkit — Replace vs Stack Policy

This policy prevents framework stacking: the slow accumulation of overlapping systems that produce conflicting artifacts, unclear precedence, and compliance theater.

It is intentionally strict. If you can't enforce it, don't pretend you have governance—admit you are experimenting.

Purpose

Ensure that every recurring decision domain has:

- one clear **owner** (authority boundary),
- one **source-of-truth artifact**, and
- one system that is **primary** (others must be subordinate or removed).

Definitions

System: a decision machine that produces inspectable artifacts under enforceable constraints.

Stacking: adding a new system without removing or subordinating existing systems that optimize the same decision or control the same object.

Replace: retire a system and move its decision ownership to a new system.

Subordinate: keep a system but explicitly declare it secondary via precedence rules.

Non-Negotiable Rules

Rule 1 — No new system without an observable failure

A proposal must include an Observable Failure Statement.

If the proposal begins with "we need alignment/clarity/visibility," it is rejected until rewritten as an observable failure.

Rule 2 — No new system without a named decision type

A proposal must specify exactly one primary decision type:

- priority / scope / ownership / sequencing / investment / diagnosis / repair

If it claims to optimize multiple, it must be decomposed into smaller systems or rejected.

Rule 3 — No new system without a source-of-truth artifact

A proposal must name the artifact that becomes authoritative for the decision domain and where it lives.

If it doesn't change what artifact wins, it isn't a system—it's a meeting.

Rule 4 — Add implies remove or subordinate

Every new system must identify at least one existing system that will be:

- removed, or
- subordinated (with explicit precedence rules)

If "none," then the proposal must prove it owns a currently unowned decision domain (a true gap).

Rule 5 — Precedence must be explicit

If two systems touch the same domain, the policy requires a precedence rule:

- "When X conflicts with Y, Y wins, because Y owns the decision."

Ambiguity is treated as collision and must be resolved before rollout.

Rule 6 — Enforcement and defaults are mandatory

Every system must include at least one enforceable constraint and an explicit default when violated.

If enforcement relies on reminders or goodwill, the system cannot be mandated.

Rule 7 — Systems must be reviewable and killable

Every new system must include:

- a review date, and
- kill criteria (evidence that triggers modify/subordinate/remove)

If the system cannot be retired, it is an institution and requires a different governance process.

Required Proposal Packet (Minimum)

A proposal is considered valid only if it includes:

1. Observable Failure Statement
2. Dominant frame (strategy/discovery/delivery/cooperation/evolution)
3. Primary decision type
4. Object(s) of control (1–2 max)
5. Artifact(s) produced (source of truth identified)
6. Constraint + default
7. Misuse model (at least 3) + mitigations
8. Adoption path (who first; minimal viable use; time to first value)
9. Replacement/subordination plan (what changes in the landscape)
10. Review date + kill criteria

Replace vs Subordinate Decision Guide

Prefer Replace when:

- the old system optimizes the same decision and causes collision
- the old system is ritual/theater (no enforceable constraints)
- the new system provides a stronger artifact + enforcement model

Replacement requirement:

- migrate or archive the old system's artifacts
- explicitly communicate end date and what changes

Prefer Subordinate when:

- the old system is still useful locally but conflicts at a higher level
- you need it as an input artifact, not a decision artifact
- the new system owns the decision but consumes outputs from the old

Subordination requirement:

- write precedence rules in the documentation and artifacts
- clarify what the subordinate system is *not allowed* to decide

---

Precedence Rules Template

Use this exact format:

- **Decision domain:** _____
- **Primary system:** _____
- **Source-of-truth artifact:** _____
- **Subordinate systems:** _____
- **Precedence rule:** "When artifacts conflict, _____ wins."
- **Default when conflict is unresolved by deadline:** _____
- **Decider for precedence disputes:** _____

Governance Workflow (Lightweight)

Step 1 — Decompose the proposed system

Use the Decomposition Table to confirm decision type, object of control, unit of analysis, and enforcement.

Step 2 — Map the collision risk

Update the System Landscape Matrix:

- show where it overlaps
- show what it replaces/subordinates

Step 3 — Require a minimal viable run

Before any mandate:

- run one cycle in a small scope
- produce real artifacts
- demonstrate enforcement once (constraint + default triggered or proven)

Step 4 — Decide rollout level

Only after evidence:

- keep local
- expand to adjacent teams
- mandate (rare; requires strong enforcement and clear authority)

Step 5 — Review and kill if needed

At the review date, choose:

- keep / modify / subordinate / remove

"No decision" is treated as "remove" by default.

Default Rules (When People Avoid Decisions)

These defaults prevent indefinite limbo:

- If precedence disputes are unresolved by the deadline, the **primary system's artifact wins**.
- If a new system cannot name what it replaces/subordinates, it is **not introduced**.
- If a system cannot show decision improvement by the review date, it is **removed or reduced**.

Common Policy Failure Modes (and Fixes)

Failure mode: "We'll keep both"

Result: collisions, translation work, slow drift into theater.

Fix:

- enforce Rule 4 (add implies remove/subordinate) with no exceptions.

Failure mode: "We'll pilot forever"

Result: no commitment, no selection pressure, sprawl.

Fix:

- require kill criteria and a hard review date.

Failure mode: "This is just guidance"

Result: no enforcement, no decision improvement.

Fix:

- refuse mandate; redesign constraint/default or abandon.

Policy Summary (One Screen)

- No failure → no system
- No decision type → no system
- No source artifact → no system
- Add implies remove/subordinate
- Precedence rules are mandatory
- Constraints + defaults are mandatory
- Review + kill criteria are mandatory

Toolkit — Decision-Type Field Guide

This guide helps you identify *which decision you are actually failing to make*, and what kinds of systems tend to work for that decision. Each decision type includes:

- failure signals (what you'll observe)
- best-fit objects of control
- artifact options (inspectability)
- constraint + default patterns (teeth)
- common misuses (predictable breakage)

Use it to translate vague problems ("alignment", "clarity", "speed") into a concrete decision machine.

Priority

When priority is failing (signals)

- Everything is "top priority"
- Work starts faster than it finishes
- Priorities reverse after reviews
- Teams optimize local queues, global goals drift

Best-fit objects of control

- Work items
- Constraints (initiative/WIP caps)
- Information flow (single priority artifact)

Artifact options

- Ranked stack with explicit capacity allocation
- "Now / Next / Later" with explicit "Not doing"
- Decision log of priority changes (why, when, who)

Constraint + default patterns

- Initiative cap (e.g., max 3) + **displacement default** ("new displaces lowest")
- WIP cap + **pause default** ("pause, don't parallelize")
- Deadline to decide + **continue-current default** ("if not decided, current top remains")

Common misuses

- Priority artifact becomes status report
- Leadership injects "urgent" without displacement
- Priority is declared per team, not per shared constraint (collision)

Scope

When scope is failing (signals)

- Work expands mid-flight (“just one more thing”)
- Commitments are missed due to scope creep
- Requirements debates never end
- “Done” is negotiable

Best-fit objects of control

- Work items (definition, slicing, acceptance)
- Constraints (timeboxes, gates)
- Information flow (scope boundary visibility)

Artifact options

- In/Out list (explicit boundary)
- Acceptance criteria / Definition of Done
- Shaped pitch document (bounded work)

Constraint + default patterns

- Timebox (“appetite”) + **ship-something default** (reduce scope, keep deadline)
- “No change without tradeoff” + **remove-something default**
- Gate: “No start without exit criteria” + **block-start default**

Common misuses

- Scope becomes contract rigidity (no learning allowed)
- Scope is negotiated via politics, not artifacts
- “Scope control” becomes bureaucracy (heavy approvals)

Ownership

When ownership is failing (signals)

- “Not my problem” or “ask X” is common
- Dependencies stall, escalations dominate
- Interfaces change without coordination
- Responsibility is assigned without authority

Best-fit objects of control

- Interfaces
- Domains
- Authority boundaries (decision rights)

Artifact options

- Ownership map (system/interface/domain → owning team)

- Interface contract (change protocol, compatibility rules, SLAs)
- Decision rights table (decide/consult/inform)

Constraint + default patterns

- “Every interface has an owner” + **ops-default** (“team running it owns until reassigned”)
- Review SLA + **safe-default** (“if no response, proceed under backward-compatible-only rule”)
- Escalation timer + **auto-escalate default**

Common misuses

- Ownership becomes gatekeeping power capture
- “Shared ownership” becomes “nobody owns”
- Ownership maps rot (no review cadence)

Sequencing

When sequencing is failing (signals)

- Work is constantly blocked by dependencies
- Too many parallel efforts, low completion rate
- “Next” changes daily; throughput is volatile
- Bottlenecks are known but ignored

Best-fit objects of control

- Work items (ordering, dependency clarity)
- Constraints (WIP/queue policies)
- Interfaces (dependency boundaries at multi-team)

Artifact options

- Dependency map + chosen order
- Queue policy (classes of service)
- Bottleneck map (where work waits)

Constraint + default patterns

- WIP limit + **displacement default**
- Dependency resolution timebox + **escalate default**
- “Stop-the-line” rule + **stabilize-first default**

Common misuses

- Sequencing becomes “plans” disconnected from capacity
- People route around the sequence with side channels
- Dependencies are tracked but not resolved (artifact without enforcement)

Investment

When investment is failing (signals)

- Too many initiatives, few outcomes
- Nothing dies; pilots run forever
- Strategy resets don't change allocation
- Important but non-urgent work never gets funded

Best-fit objects of control

- Goals (as selection criteria)
- Constraints (allocation caps, kill criteria)
- Information flow (portfolio visibility)

Artifact options

- Allocation table (capacity by category)
- Initiative/bet tracker with owners and budgets
- Kill/continue/increase decision log

Constraint + default patterns

- Fixed allocation per cycle + **displacement default** for mid-cycle adds
- Initiative cap + **must-kill default** (new requires killing one)
- Kill criteria + **auto-kill default** at review if criteria unmet

Common misuses

- Investment artifacts become reporting decks
- Kill decisions become politically unsafe (no selection pressure)
- Teams sandbag to protect funding (metric gaming)

Diagnosis

When diagnosis is failing (signals)

- Same failures repeat with different blame stories
- Postmortems exist but don't change behavior
- "Root cause" is vague or moral ("communication")
- Data exists but doesn't change decisions

Best-fit objects of control

- Information flow (what evidence is captured)
- Work items (problem definitions)
- Constraints (required "next change" output)

Artifact options

- Incident timeline + causal map

- Decision log entry (“what we now believe and will change”)
- Hypothesis + evidence table (what would change our mind)

Constraint + default patterns

- “No postmortem without next-change decision” **+ block-close default**
- Evidence requirement + **unknown-explicit default** (“mark as unknown, assign investigation”)
- Timeboxed diagnosis + **smallest-safe-fix default**

Common misuses

- Diagnosis becomes storytelling, not decision-making
- Blame capture (politics)
- Endless analysis avoids repair commitments

Repair

When repair is failing (signals)

- Reliability and tech debt always lose to features
- Incidents recur; “we’ll fix later” repeats
- Teams normalize toil and firefighting
- Known constraints remain for months

Best-fit objects of control

- Constraints (protected capacity, gates)
- Work items (repair backlog definition)
- Interfaces (where failures originate across boundaries)

Artifact options

- Repair backlog with owners and deadlines
- Reliability budget / error budget policy (if applicable)
- Repair decision record per incident (single “next change”)

Constraint + default patterns

- Protected repair allocation + **scope-cut default** (feature scope reduces if repair violated)
- Stop-the-line threshold + **stabilize-default** (no new releases until fixed)
- SLA for repair items + **escalate-default**

Common misuses

- Repair becomes “nice to have” with no enforcement
- Repair capacity silently borrowed
- Repair backlog becomes a graveyard (no selection/pruning)

Fast Translation Table

Use this to convert vague requests into decision types:

- “We need alignment” → usually **priority, scope, ownership**, or **investment**
- “We need to move faster” → often **sequencing** (flow) or **repair** (constraints)
- “We keep redoing work” → often **diagnosis** (learning loop) or **ownership** (boundaries)
- “Too many initiatives” → **investment** + scope constraints
- “Cross-team friction” → **ownership** (interfaces/domains)

Choosing a Decision Type Under Ambiguity

If multiple seem true, ask:

1. “Which decision, if made well, would dissolve the others?”
2. “Which decision are we most consistently avoiding?”
3. “Which decision is currently being made by accident (defaults, politics, urgency)?”

Pick one primary decision type and treat others as secondary effects.

Exit Condition for This Guide

You can treat this guide as “installed” when you can:

- take a vague complaint in a meeting,
- name the likely decision type within 60 seconds,
- propose a candidate artifact and a constraint+default that would force the decision to become inspectable.

ABOUT

Version and Licensing

This section documents the current version, licensing terms, and attribution principles for the **System Design Lens (SDL)**. Its purpose is to ensure clarity, traceability, and responsible reuse across derivative works, teaching materials, internal playbooks, and organizational systems built using this doctrine.

Version Information

Attribute	Description
System Name	**System Design Lens (SDL)**
Version	**V1.0**
Status	Stable — doctrinal release
Release Date	December 2025
Canonical Reference	*The Discipline of Decision Design*
Maintained by	3in3.dev
Latest Online Documentation	Always find the latest version and web source here.

Version 1.0 Summary (Current)

Version 1.0 establishes **System Design Lens** as a **decision-design doctrine**, not a framework or operating model.

It formally defines:

- Systems as **decision machines**, not practices or rituals
- A fixed set of **canonical dimensions** for system analysis and design
- **Observable failure** as the non-negotiable entry condition for system work
- **Artifacts and constraints** as mandatory enforcement mechanisms
- **Misuse modeling** as a first-class design requirement
- **Minimal Viable Systems (MVS)** as the default construction strategy
- Explicit rules for **system replacement, subordination, and retirement**

Version 1.0 is intentionally complete. Future versions are expected to extend applicability or clarify edge cases, not to alter the core logic.

Versioning Policy

- **Major versions (V2, V3, …)** may introduce new decision categories, canonical dimensions, or system shapes that expand the doctrine.
- **Minor revisions (V1.1, V1.2, …)** provide clarifications, refinements, examples, or misuse guidance without changing core rules.
- **Patch revisions** correct errors, language precision, or internal consistency.
- All published versions remain **permanently valid for reference and citation**.
- Backward compatibility is prioritized at the **conceptual level**: new material must not invalidate the original decision logic.

License

System Design Lens (SDL) is released as **open intellectual infrastructure** for decision and system design.

> **License Type:** Creative Commons Attribution 4.0 International (CC BY 4.0)

You are free to:

- **Share** — copy and redistribute the material in any medium or format.
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Attribution & Citation

To attribute this work, please use the following reference:

> *System Design Lens (SDL), from* **The Discipline of Decision Design** by 3in3.dev — licensed under CC BY 4.0 via 3in3.dev.

For derivative works, adaptations, or internal frameworks:

- Clearly state that the work is **based on or adapted from System Design Lens**
- Identify what has been modified, extended, or omitted
- Preserve original terminology where possible to avoid semantic drift

Attribution Guidelines for Derivative Works

If you reuse or adapt SDL:

1. Retain core definitions (decision types, canonical dimensions, non-negotiable rules).
2. Do not present SDL-derived material as proprietary or closed.
3. Clearly separate **original SDL doctrine** from **local adaptations or extensions**.
4. If teaching or embedding SDL inside another framework, ensure SDL remains a **lens**, not a rebranded method.

Recommended phrasing for adaptations:

> "Adapted from System Design Lens (SDL), licensed under CC BY 4.0."

Scope Boundary

System Design Lens deliberately **does not prescribe**:

- specific tools or templates
- organizational structures
- process cadences
- technology stacks
- cultural or motivational techniques

SDL governs **how systems are reasoned about**, not how organizations are run day to day.

Misrepresenting SDL as an operating model or methodology violates the intent of the doctrine, even if license terms are met.

Copyright Notice

About the Author

Viktor Jevdokimov, Vilnius, Lithuania — Creator of 3in3.dev, HCS, and 3SF

Viktor Jevdokimov is a software engineering leader, systems thinker, and framework designer with over 30 years of experience in software product delivery, modernization, and team alignment.

He is the creator of the **Human Cooperation System (HCS)** and the **3-in-3 SDLC Framework (3SF)**, and founder of the **3in3.dev** initiative — an independent platform dedicated to advancing collaboration and alignment between **Client**, **Vendor**, and **Product** ecosystems.

Professional Background

- Began career supporting distributed banking software on DOS and Windows, developing a deep appreciation for troubleshooting and system design.
- Progressed through roles of **developer**, **architect**, **delivery lead**, and **practice lead**, working with international clients on modernization and cloud migration initiatives.
- Specializes in **Client–Vendor relationship design**, **project leadership**, and **delivery system diagnostics**.
- Advocates for *"Context before Method"* and *"Trust before Control"* as guiding principles of effective collaboration.

Creative and Personal Work

Beyond software, Viktor is an **active musician and live sound engineer**, performing and mixing with the *Great Things* cover band.
He approaches both sound and systems with the same mindset: striving for **clarity, balance, and authenticity**.

About 3in3.dev

3in3.dev is an independent research and publishing initiative founded by Viktor Jevdokimov.
It consolidates his experience and experimentation into open frameworks that help organizations improve how they **engage, deliver, and measure value** across collaborative ecosystems.

3in3.dev publishes:

- The **Human Cooperation System (HCS)** — theoretical foundation for cooperative system design.
- The **3-in-3 SDLC Framework (3SF)** — practical application of HCS principles in software delivery.
- Supporting tools, templates, and learning materials under an open license.

> "These systems aren't about control — they're about clarity, trust, and the shared intent that makes collaboration work."
> — Viktor J., Creator of 3in3.dev

Connect and follow on **LinkedIn** for updates and professional discussions.
For contact, collaboration, or speaking requests, visit **https://3in3.dev**.