3-in-3 SDLC Framework (3SF)

A relationship-aware software delivery framework that unites client, vendor, and product/service into one coherent system of collaboration.

Viktor Jevdokimov, Vilnius, Lithuania, 2025-11-04

Table of contents

3-in-3 SDLC Framework (3SF) 8

3-in-3 SDLC Framework (3SF) 9

- Introduction 9
- Purpose 9
- Core Idea 9
- Framework Overview 10
- Evolution 10
- Resilience by Design 10
- When to Use 3SF 11
- Benefits 11
- Reading Structure 11
- Summary 11

Vision, Principles & Beliefs 12

- Vision 12
- Principles 12
- Beliefs 13
- Systemic Balance 14

3-in-3 Model 15

- The 3-in-3 Model 15
- The Three Relationship Lines 15
- The 3-in-3 Dynamic 16
- When the Triangle Collapses 16
- Summary 16

3-in-3 Model Maturity 17

- Shared Responsibility & Relationship Maturity 17
- Assessing Relationship Maturity 17
- Why Relationship Maturity Matters 17
- When Maturity Stalls 18
- Summary 18

SDLC Stages 19

- Overview 19
- The Seven SDLC Stages 19
- SDLC as a System of Flow and Learning 19
- Relationship-Aware SDLC 20
- Flow Characteristics Across the SDLC 20

- Run vs Evolve: Stability vs Learning 20
- Using the SDLC Stages 21
- Summary 21

SDLC Stages Maturity 22

- Purpose 22
- Maturity Levels Across the SDLC 22
- SDLC Stage Maturity Map 23
- Structural, Operational, and Relational Dimensions 24
- Assessing SDLC Stage Maturity 24
- When Maturity Plateaus 24
- Summary 24

SDLC Practices 25

- Overview 25
- The Six SDLC Practices 25
- Practices as the Delivery Backbone 25
- Practices Across SDLC Stages 26
- Practice Interdependence 26
- When Practices Compete 27
- Practices as a Foundation for Maturity 27
- Summary 27

SDLC Practices Maturity 28

- Purpose 28
- Practice Maturity Levels 28
- Maturity by Practice 28
- Cross-Practice Synergy 29
- When Maturity Inverts 29
- Assessing Practice Maturity 29
- Summary 29

SDLC Stage Dimentions 30

- Purpose 30
- What Stage Dimensions Are 30
- The Role of Dimensions in 3SF 30
- Dimension Categories 31
- Example: How Dimensions Interconnect 31
- Why Dimensions Matter 31
- Relationship to Maturity 31
- Applying Stage Dimensions 32
- When Dimensions Become Bureaucracy 32

- Structure of Stage Dimension Files 32
- Summary 32

SDLC Stage Dimensions – Discover 33

- Purpose 33
- Core Outcomes 33
- Discovery Dimensions 33
- Common Failure Modes 35
- Measuring Discovery Health 35
- Discovery and Relationship Maturity 36
- Summary 36

SDLC Stage Dimensions – Shape 37

- Purpose 37
- Core Outcomes 37
- Shape Dimensions 37
- Common Failure Modes 39
- Measuring Shape Health 40
- Shape and Relationship Maturity 40
- Summary 40

SDLC Stage Dimensions – Build 41

- Purpose 41
- Core Outcomes 41
- Build Dimensions 41
- Common Failure Modes 43
- Measuring Build Health 43
- Build and Relationship Maturity 44
- Summary 44

SDLC Stage Dimensions – Validate 45

- Purpose 45
- Core Outcomes 45
- Validate Dimensions 45
- Common Failure Modes 47
- Measuring Validation Health 48
- Validate and Relationship Maturity 48
- Summary 48

SDLC Stage Dimensions – Release 49

- Purpose 49
- Core Outcomes 49
- Release Dimensions 49

- Common Failure Modes 51
- Measuring Release Health 51
- Release and Relationship Maturity 52
- Summary 52

SDLC Stage Dimensions – Run & Evolve 53

- Purpose 53
- Core Outcomes 53
- Run & Evolve Dimensions 53
- Common Failure Modes 55
- Measuring Run & Evolve Health 55
- Run & Evolve and Relationship Maturity 56
- Summary 56

Contextual Drivers Layer (CDL) 57

- Purpose 57
- CDL in the 3SF Structure 57
- CDL Categories 57
- CDL as a Diagnostic Lens 58
- CDL → SDLC Connection Map 58
- CDL Assessment 58
- CDL in Practice 59
- CDL and Relationship Maturity 59
- Principles Failure: Contextual Contradictions 59
- Summary 59

Stable Rules Layer (SRL) 60

- Purpose 60
- SRL in the 3SF Structure 60
- Rule Design Philosophy 60
- The Twelve Stable Rules of 3SF 61
- The Why Behind the Rules 62
- Rule Relationships and Hierarchy 62
- When Rules Collide 62
- SRL → SDLC Mapping 63
- Violations and Signals 63
- Using the Stable Rules Layer 63
- SRL and Maturity 64
- Summary 64

Rule Audit Checklist (RAC) 65

- Purpose 65

- RAC in the 3SF Structure 65
- Using the RAC 65
- RAC Template 66
- Interpreting RAC Results 68
- RAC Example Summary Report 69
- RAC Application in Practice 69
- RAC Facilitation Tips 69
- RAC and Relationship Maturity 69
- Systemic Failure: The Maturity Mirage 69
- Summary 69

Contextual Rules Catalog (CRC) 71

- Purpose 71
- CRC in the 3SF Structure 71
- CRC Design Philosophy 71
- CRC Categories (Delivery Context Archetypes) 72
- CRC Example Mappings 72
- CRC Mapping Template 74
- CRC and Maturity 74
- CRC Governance 75
- Adapting Without Excusing 75
- CRC and Relationship Maturity 75
- Summary 75

Relationship and System Maturity Integration 76

- Purpose 76
- The Integrated Maturity Curve 76
- The Hidden Cost of Transactional Trust 76
- The Six Capability Gap Categories 77
- Capability Maturity Mapping 77
- Relationship Progression Goals (Practices That Move Rightward) 77
- Systemic Integration Across 3SF Layers 78
- Model Failure: The Collapsed Triangle 79
- Example Progression Scenario 79
- Measuring Relationship Maturity in Practice 79
- Behavioral Failure: The Scapegoat Culture 79
- Summary 80

3SF Practice Architecture 81

- Purpose 81
- Objectives 81

Structure of the Practice Part 81
Dual-Perspective Application 82
Application Modes 82
Lifecycle Navigation 83
3SF Layer Mapping 83
3SF Functional Role Model 83
Practice Template 85
Using Practice Architecture in Training 85
Summary 85

3-in-3 SDLC Framework (3SF)

3-in-3 SDLC Framework (3SF)

Introduction

The **3-in-3 SDLC Framework** — or **3SF** — is a relationship-aware software delivery framework that unites **client**, **vendor**, and **product/service** into one coherent system of collaboration.
It was created to help organizations design, execute, and evolve delivery systems that are **aligned, predictable, and continuously learning**, regardless of project type or commercial model.

Where most delivery frameworks describe *what happens inside a vendor organization*, 3SF explains **how success depends on the relationship between all three system elements** — *Client ↔ Vendor ↔ Product* — and how alignment across them determines value realization.

Purpose

The 3SF was designed to:

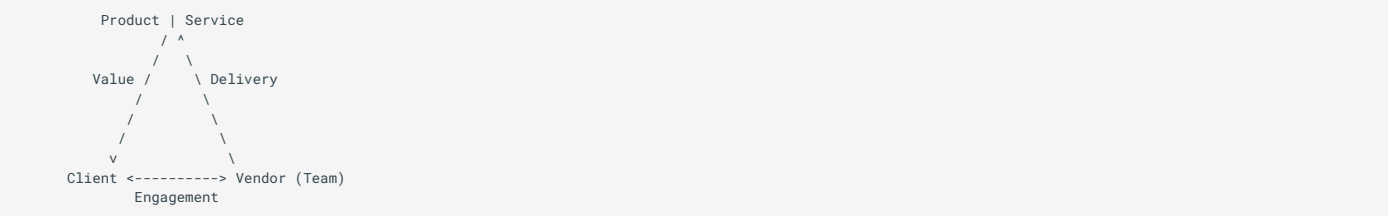
- **Connect strategy and delivery** by translating goals, constraints, and context into practical delivery configurations.
- **Expose and close maturity gaps** between client and vendor through shared understanding of responsibilities and decision-making.
- **Balance structure and adaptability** — combining predictable delivery with continuous learning and feedback.
- **Enable contextual decision-making** — showing how commercial, organizational, and technical drivers shape the right delivery approach.
- **Build partnership maturity** over time, turning transactional engagements into strategic collaborations.

It serves both as:

1. **A delivery design framework** — guiding how to configure projects before execution.
2. **A diagnostic tool** — helping leaders assess delivery health and identify systemic improvements.
3. **A learning model** — supporting continuous improvement and leadership development.

Core Idea

Every delivery system operates within a triangle of relationships:



Each side of this triangle represents a **relationship line**:

- **Engagement (Client ↔ Vendor):** defines collaboration and trust.
- **Delivery (Vendor ↔ Product):** defines execution and quality.
- **Value (Product ↔ Client):** defines outcomes and impact.

Healthy delivery emerges only when all three sides evolve together.

Framework Overview

The **3-in-3 SDLC Framework (3SF)** organizes delivery into **eight systemic layers**, each serving a distinct purpose within the client–vendor–product ecosystem.
Together, these layers form a complete system of alignment, flow, and learning — connecting strategy to execution and collaboration to maturity.

#	Layer	Purpose
1	**3-in-3 Model**	Defines the delivery ecosystem — the relationships between Client, Vendor (Team), and Product / Service through Engagement, Delivery, and Value lines.
2	**SDLC Stages**	Describes how value flows through seven delivery stages — Discover, Shape, Build, Validate, Release, Run, and Evolve.
3	**SDLC Practices**	Provide technical, quality, and governance disciplines that sustain flow and reliability across all stages.
4	**SDLC Stage Dimensions**	Define configurable decision spaces and conditions that determine how each stage operates in context.
5	**Contextual Drivers Layer (CDL)**	Identify external and internal forces — commercial, organizational, or technical — that shape delivery decisions.
6	**Stable Rules Layer (SRL)**	Encode universal system rules (R1–R12) that maintain coherence and balance under changing conditions.
7	**Rule Audit Checklist (RAC)**	Provide a structured method for evaluating delivery health, rule adherence, and improvement priorities.
8	**Contextual Rules Catalog (CRC)**	Translate the framework into actionable playbooks for specific delivery contexts (e.g., Fixed-Bid, Co-Delivery, Continuous Evolution).

The **3-in-3 Maturity Layer** overlays the entire framework — describing how **relationship trust and shared responsibility evolve** as the delivery system matures from **Transactional Trust** to **Strategic Partnership** through aligned autonomy, feedback, and learning.

Evolution

3SF evolved from repeated real-world observations in client-vendor engagements where:

- Projects struggled not because of technical complexity, but due to *mismatched expectations and responsibilities.*
- Vendors assumed clients handled early discovery or sustainment functions, leaving unowned gaps.
- Delivery methodologies (Agile, Waterfall, Hybrid) failed when context and relationships weren't considered.

The framework emerged as a **unifying model** — connecting strategic, operational, and adaptive layers into one ecosystem of delivery understanding.

Resilience by Design

3SF acknowledges that **delivery systems fail not only because of bad execution but because of systemic resistance** — competing incentives, unclear accountability, or organizational inertia.

To remain realistic and resilient, the framework was designed with *anti-fragility* in mind:

- Each layer contains **diagnostic and corrective mechanisms** (CDL, SRL, RAC) to detect and respond to friction before it becomes failure.
- The **relationship maturity model** exposes illusionary progress — when processes appear mature but trust and ownership remain shallow.
- The **Stable Rules Layer** acts as a safety net — ensuring that adaptation never breaks coherence.
- And through **feedback-driven learning**, every mistake becomes a data point for system evolution, not evidence of personal failure.

In practice, 3SF doesn't eliminate risk — it **makes risk visible, discussable, and manageable** across both client and vendor organizations.

When to Use 3SF

Use Case	Typical Goal
Before a project starts	Design the right delivery system based on context and commercial model.
During delivery	Diagnose bottlenecks, misalignments, and maturity gaps using RAC and CRC.
After release	Guide retrospectives, learning, and relationship improvement across projects.
For leadership development	Train Delivery Leaders to think systemically and lead through context, not process.

Benefits

- **Clarity:** Shared language across client, vendor, and leadership levels.
- **Alignment:** Connected view of strategy, delivery, and value realization.
- **Predictability:** Systematic handling of uncertainty and governance.
- **Learning:** Built-in reflection loops to prevent repeated failures.
- **Scalability:** Applicable to individual projects or portfolios.

Reading Structure

The theory section introduces 3SF from **philosophy to application**:

1. **Vision, Principles, Beliefs** → Why the framework exists.
2. **3-in-3 Model & Maturity** → Core relationship foundation.
3. **SDLC Model** → Phases and practices.
4. **Stage Dimensions** → How each stage is configured.
5. **Contextual Drivers Layer (CDL)** → What shapes delivery decisions.
6. **Stable Rules Layer (SRL)** → Universal rules of systemic integrity.
7. **Rule Audit Checklist (RAC)** → Diagnostic and assessment method.
8. **Contextual Rules Catalog (CRC)** → Practical playbooks for different contexts.
9. **Relationship and System Maturity Integration** → Integration of partnership growth into SDLC.

Summary

> **3SF connects technical delivery with relationship maturity — aligning structure, behavior, and learning into a single, living system.**

It is not a new methodology; it is a **meta-framework** that ensures any methodology fits its context and grows trust between organizations.

Vision, Principles & Beliefs

Vision

We envision a world where **client, vendor, and product** teams collaborate as one delivery system — driven by shared purpose, transparent flow, and continuous learning.
Software delivery should not be a negotiation between parties but an **ecosystem of mutual value creation**.

3SF exists to make this possible: to replace process compliance with contextual alignment, and to transform transactional engagements into partnerships built on trust and adaptability.

However, **not every organization that uses 3SF language achieves its spirit**.
Without the underlying mindset shift, structures can imitate maturity while behaviors remain transactional — creating the illusion of transformation.
3SF treats this *"maturity mirage"* as a signal, not a success, guiding leaders to inspect how intent, trust, and accountability truly manifest in action.

Principles

3SF is grounded in a set of **delivery principles** — systemic behaviors that connect context, structure, and trust.
They are not rules to enforce but **lenses for decision-making** that preserve alignment and flow under change.

Core Principles

Principle	Meaning
Context before Method	Every process, tool, or practice must serve the context, not the other way around.
Flow before Speed	Predictable progress matters more than motion without direction.
Outcome before Output	Value is defined by impact, not activity.
Trust before Control	Alignment and autonomy are stronger than oversight; trust is earned through transparent flow and proven competence.
Learning before Blame	Every issue is a feedback signal, not a failure of character.
Shared Accountability	Responsibility for value and quality extends across client and vendor boundaries.
Transparency Enables Adaptation	Information symmetry is the foundation of trust and continuous improvement.
Balance over Rigidity	Sustainable delivery emerges from equilibrium between structure and flexibility.

Principles in Action

Each principle represents a **systemic antidote** to common failure patterns:

Failure Pattern	Counteracting Principle
Fragmented delivery and unclear ownership	Context before Method
Reactive decision-making	Flow before Speed
Feature-driven planning without impact assessment	Outcome before Output
Micromanagement and escalation culture	Trust before Control
Fear of retrospectives or feedback	Learning before Blame
Siloed goals between client and vendor	Shared Accountability
Information asymmetry or selective reporting	Transparency Enables Adaptation
Over-reliance on rigid processes	Balance over Rigidity

Principles Under Pressure

Principles do not operate in isolation — real delivery systems expose **tensions between them**.
For example, *"Outcome before Output"* may conflict with *"Trust before Control"* in regulated environments that demand auditable processes.
In such cases, 3SF prioritizes principles hierarchically:

> **Context → Alignment → Flow → Learning**

When conflicts arise, decisions should return to higher-order principles:

- **Context** defines boundaries.
- **Alignment** ensures coherence.
- **Flow** maintains momentum.
- **Learning** preserves adaptability.

By making these trade-offs explicit, teams prevent principle collision from devolving into politics or paralysis.

Beliefs

3SF operates from a set of shared beliefs — the **cultural DNA** that supports system maturity.

Belief	Description
People build systems, not processes.	No framework compensates for missing trust or unclear purpose.
Clarity enables autonomy.	When everyone understands why, they can decide how.
Collaboration scales faster than control.	Distributed trust accelerates adaptation and decision speed.
Transparency multiplies learning.	Open systems self-correct faster than closed ones.
Every failure is data.	Problems are feedback — essential to evolution, not obstacles to avoid.

Beliefs are also the **first layer to erode under pressure** — often replaced by performance metrics or process compliance.
3SF uses reflective rituals and maturity assessments to **surface this erosion early**, turning it into learning rather than cultural decay.

Systemic Balance

Vision, Principles, and Beliefs form the **human foundation** of 3SF.
They sustain coherence when context shifts and prevent the framework from becoming a set of mechanical practices.

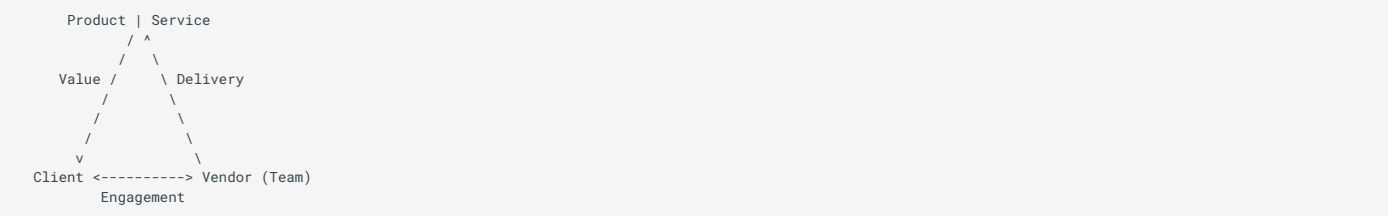
When one element dominates — **vision without principles**, or **principles without belief** — coherence breaks, and the system reverts to control.

Together, they define the **why**, **how**, and **who** behind every 3SF decision.

3-in-3 Model

The 3-in-3 Model

The **3-in-3 Model** defines the entire delivery ecosystem as a system of three entities — **Client**, **Vendor (Team)**, and **Product / Service** — connected by three relationships: **Engagement**, **Delivery**, and **Value**.



Each side of the triangle represents a vital relationship that must evolve for sustained delivery success.
When any side becomes weak or unbalanced, the system loses integrity — resulting in friction, rework, or loss of trust.

The Three Relationship Lines

1. Engagement — *Client ↔ Vendor*

Defines the **working relationship and trust** between those who need the solution and those who build it.
It determines how decisions are made, how transparency is maintained, and how collaboration scales.

Healthy Engagement enables:

- Shared understanding of goals and constraints.
- Clear governance, communication cadence, and risk ownership.
- Trust that empowers autonomy and reduces friction.

When weak:

- Misaligned expectations or conflicting priorities.
- Reactive communication, scope disputes, and politics.
- “Us vs Them” behavior that blocks collaboration.

2. Delivery — *Vendor ↔ Product / Service*

Defines the **execution system** that transforms ideas into working outcomes.
It represents the vendor team’s capability, flow, and technical integrity.

Healthy Delivery enables:

- Predictable progress and measurable quality.
- Architecture and engineering aligned with product intent.
- Continuous feedback between design, build, and validation.

When weak:

- Over- or under-engineering due to unclear direction.
- Hand-offs that break accountability.
- Quality or automation gaps that erode trust.

3. Value — *Product / Service ↔ Client*

Defines the **outcome connection** — how the work contributes to business goals and user needs.

Healthy Value enables:

- Clear problem–solution alignment.
- Defined success metrics and value validation.
- Regular learning loops between users, client, and team.

When weak:

- Delivered features without measurable outcomes.
- Lack of user validation or impact tracking.
- Value assumptions never revisited post-release.

The 3-in-3 Dynamic

The triangle functions as **a system of balance and tension**:

- **Engagement** builds alignment and trust.
- **Delivery** ensures capability and flow.
- **Value** validates outcomes and relevance.

When one side strengthens, the others benefit:

- Strong Engagement improves Delivery coordination.
- Reliable Delivery builds confidence and strengthens Engagement.
- Proven Value reinforces both partnership and autonomy.

3SF ensures that every decision across the SDLC protects this systemic equilibrium.

When the Triangle Collapses

If any relationship line weakens, the 3-in-3 system contracts — often reverting to a transactional pattern.
For example, when the **Value** connection breaks (no client product ownership or user insight), Delivery continues but meaning is lost.
When **Engagement** fails, trust erodes and coordination turns contractual.
These imbalances signal a system collapse, not a team failure — recovery begins by restoring the missing relationship, not by adding more process.

Summary

The **3-in-3 Model** represents *what the system is*:
a triangle of **Client**, **Vendor**, and **Product** connected through **Engagement**, **Delivery**, and **Value** lines.

3-in-3 Model Maturity

Shared Responsibility & Relationship Maturity

Client and Vendor maturity grow **together** through progressive collaboration and shared ownership of decisions.
This maturity expresses how responsibility, trust, and autonomy evolve between both sides.

Maturity Level	Client Focus	Team Focus	Shared Responsibility
Transactional Trust	Provides requirements and approvals.	Acts as **Trusted Executor** — follows plan, ensures quality within scope.	Defined by contracts and control; limited feedback.
Aligned Autonomy	Frames outcomes, clarifies success measures.	Becomes **Autonomous in Execution** — self-manages within agreed direction.	Alignment grows through transparency and predictable delivery.
Shared Ownership	Owns vision and intent, participates in decision shaping.	**Owns Shaping and Execution** — proposes options and trade-offs.	Decisions become co-created; priorities discussed openly.
Strategic Partnership	Entrusts business risk and direction.	**Co-defines and Co-navigates Path** — contributes to vision and strategy.	Full partnership; joint accountability for outcomes and value.

Progression along this curve is gradual and reversible — it must be *earned* through reliability, openness, and continuous feedback.

Assessing Relationship Maturity

In 3SF, relationship maturity is measured not by duration but by **quality of collaboration** across three relationship lines:

1. **Engagement (Client + Vendor):** Trust, transparency, and clarity of collaboration.
2. **Delivery (Team + Delivery):** Capability, agility, and self-management in achieving goals.
3. **Value (Product + Client):** Alignment of product outcomes with real business impact.

Each can be evaluated as:

- **Reactive → Managed → Proactive**, or
- **Transactional → Aligned Autonomy → Strategic Partnership**, depending on project depth.

Tools such as the *3-in-3 Maturity Self-Assessment* help teams reflect on their current and target levels in each dimension.

Why Relationship Maturity Matters

Traditional SDLC frameworks assume delivery success depends solely on process quality.
In client–vendor environments, **relationship quality is equally structural.**

A technically correct system can still fail when:

- Governance is one-sided or unclear.
- Communication is filtered or delayed.
- Outcomes are validated separately by each side.

3SF closes this gap by embedding **dual accountability** and **shared responsibility** — ensuring that both client and vendor evolve together toward aligned autonomy and partnership.

When Maturity Stalls

Relationship maturity can regress when reliability or transparency erode.
Common signals include defensive communication, parallel decision-making, or focus on protecting scope instead of outcomes.
Such reversals do not mean failure — they reveal where trust and accountability need reinforcement.
3SF treats these moments as diagnostic opportunities to restore balance rather than assign blame.

Summary

The **Relationship Maturity Model** represents *how the system grows* —
a trajectory from **Transactional** to **Strategic Partnership**, embedded in how collaboration, trust, and shared ownership evolve over time.

Together, they make the 3-in-3 ecosystem a **living model of partnership evolution**, ensuring that as the product matures, so does the relationship that builds it.

SDLC Stages

Overview

The **3SF SDLC Stages** define the full flow of software product delivery as a **continuous system of understanding, creation, validation, and improvement**.
They describe *how value moves through the system* — from the first discovery of a problem to the ongoing evolution of the solution.

Each stage represents a distinct purpose but connects through feedback loops, ensuring learning never stops and every delivery cycle strengthens both product and partnership maturity.

The Seven SDLC Stages

Stage	Purpose	Outcome
Discover	Understand the problem, align on goals, and explore solution directions with stakeholders.	A validated understanding of the problem space and desired outcomes.
Shape	Define the product scope, architecture vision, and delivery feasibility based on real constraints.	A feasible and outcome-aligned delivery plan with known assumptions and trade-offs.
Build	Design, develop, and integrate software components with automation and quality at the core.	A working, testable product increment that demonstrates tangible progress and reliability.
Validate	Test functionality, performance, usability, and security to ensure the solution meets expectations.	Verified confidence that the product delivers the intended experience and quality.
Release	Package, approve, and deploy the solution for users through reliable and repeatable pipelines.	A stable release available to users with traceable approvals and predictable deployment.
Run	Monitor and support the live system, ensuring reliability, scalability, and fast issue resolution.	Reliable operations, user satisfaction, and data-driven visibility of system health.
Evolve	Learn from outcomes and feedback to continuously improve product, process, and team effectiveness.	Actionable insights leading to better products, stronger relationships, and organizational learning.

SDLC as a System of Flow and Learning

The stages form an *adaptive flow* of intent, delivery, and feedback:

`Discover → Shape → Build → Validate → Release → Run → Evolve ↺`

- **Discover & Shape** establish clarity and feasibility before investment.
- **Build, Validate, and Release** create and deliver tangible value.
- **Run & Evolve** close the learning loop, informing future discovery.

Each iteration through the loop increases **delivery maturity** and **relationship trust**, evolving both the system and the collaboration that sustains it.

Relationship-Aware SDLC

Each SDLC stage also represents a **relationship checkpoint** between Client and Vendor — defining how shared responsibility matures as delivery progresses.

Stage	Relationship Focus	Shared Ownership Example
Discover	Transparency and alignment on problem space.	Joint definition of goals, assumptions, and success criteria.
Shape	Trust and outcome-based planning.	Co-design of roadmap, architecture, and estimation.
Build	Autonomy and visibility in execution.	Shared sprint reviews and delivery metrics.
Validate	Confidence and mutual accountability.	Joint acceptance and performance evaluation.
Release	Predictable and transparent delivery.	Shared approval and communication to users.
Run	Stability and operational trust.	Shared monitoring dashboards and escalation paths.
Evolve	Partnership and learning continuity.	Joint retrospectives and improvement backlogs.

Relationship maturity increases as both organizations learn to rely on one another for clarity, delivery, and learning — turning the SDLC into a **partnership growth mechanism**, not just a technical lifecycle.

Flow Characteristics Across the SDLC

Characteristic	Early Phases (Discover–Shape)	Middle Phases (Build–Validate–Release)	Late Phases (Run–Evolve)
Focus	Alignment and feasibility	Execution and verification	Continuity and learning
Feedback Loops	Qualitative (people, intent)	Quantitative (metrics, performance)	Systemic (value, outcome)
Risk Profile	High uncertainty	Managed risk	Controlled adaptation
Leadership Mode	Facilitative and consultative	Operational and technical	Strategic and reflective
Ownership Distribution	Client-driven collaboration	Vendor-led autonomy	Shared strategic renewal

Run vs Evolve: Stability vs Learning

In many organizations, the **Run** phase is mistaken for the end of delivery — focused purely on uptime, cost control, and SLA adherence.
While stability is vital, stopping there creates a **false plateau of success** where no new learning enters the system.

The **Evolve** phase exists to prevent that stagnation.
Its metrics differ from Run:

- *Run* measures reliability (uptime, MTTR, operational cost).
- *Evolve* measures adaptability (validated learning, feature adoption, customer feedback integration).

When evolution is skipped, operational excellence turns brittle — systems stay available but stop improving.
A mature delivery culture measures both: **Run for stability, Evolve for growth**.

Using the SDLC Stages

The SDLC Stages act as a **map, not a prescription**.
Each project can emphasize different stages depending on context — for example:

- In R&D or early product discovery, *Discover* and *Shape* dominate.
- In modernization or migration work, *Build*, *Validate*, and *Release* take the lead.
- In product sustainment, *Run* and *Evolve* become the main engines of improvement.

3SF's value lies in enabling **conscious adaptation** — choosing how deep each stage goes based on commercial, organizational, and technical realities.

Summary

- The SDLC Stages define **how value flows** through seven interconnected phases.
- Each stage contributes both to **delivery outcomes** and **relationship maturity**.
- 3SF turns the SDLC into a **system of alignment, flow, and feedback** — ensuring that what we build is valuable, feasible, and continuously improving.

SDLC Stages Maturity

Purpose

The **SDLC Stages Maturity** model describes how delivery systems evolve in capability, alignment, and learning across the seven SDLC stages. It reflects how a project grows from *controlled execution* to *adaptive partnership* — connecting delivery performance to relationship trust.

Each SDLC stage not only defines what to do but also how **maturely** it can be done:

- **Structurally**, by having clarity and consistency.
- **Operationally**, by enabling flow and feedback.
- **Relationally**, by building trust and shared responsibility.

Maturity Levels Across the SDLC

Level	Name	System Characteristics	Collaboration Behavior
1. Controlled Execution	Reactive and rule-bound. Processes exist but are applied rigidly; success depends on individuals.	Client controls; vendor executes. Communication transactional and status-based.	
2. Coordinated Flow	Processes repeatable and partly adaptive. Feedback loops appear, but learning not yet systematic.	Client and vendor coordinate. Transparency increases, autonomy conditional.	
3. Empowered Delivery	Delivery system self-regulating; quality and risk controlled by teams.	Teams own execution; governance focuses on outcomes. Collaboration proactive.	
4. Adaptive Partnership	System continuously learns and adapts; governance and delivery fully integrated.	Client and vendor co-own outcomes, plan jointly, and innovate together.	

Each stage can operate at a different maturity level — for example, *Build* may perform at Level 3 while *Evolve* remains at Level 2. The goal is not uniform maturity, but **balance** between flow, governance, and relationship depth.

SDLC Stage Maturity Map

SDLC Stage	Stage Purpose	Minimum → Target Maturity	Client Role	Vendor / Team Role	Shared Focus
Discover	Understand the problem, align on goals, explore solution directions.	Transactional → Aligned Autonomy	Provides context, goals, and stakeholder access.	Facilitates discovery workshops, defines problem space, validates insights.	Shared understanding of goals, assumptions, and constraints.
Shape	Define product scope, architecture vision, and delivery feasibility.	Aligned Autonomy → Shared Ownership	Frames outcomes and priorities.	Co-creates solution architecture, estimates feasibility, defines delivery model.	Outcome-based planning and governance clarity.
Build	Design, develop, and integrate components with automation and quality at the core.	Shared Ownership → Strategic Partnership	Owns vision and validates increments.	Delivers autonomously with transparency and technical excellence.	Continuous feedback, visibility, and trust in progress.
Validate	Test functionality, performance, usability, and security.	Shared Ownership → Strategic Partnership	Participates in acceptance testing and user validation.	Integrates quality and measurement practices.	Mutual accountability for outcomes and experience.
Release	Package, approve, and deploy through reliable pipelines.	Strategic Partnership (sustained)	Approves and communicates business impact.	Operates reliable CI/CD and rollout procedures.	Shared decision on readiness and release timing.
Run	Monitor and support the live system.	Strategic Partnership → Aligned Renewal	Defines SLAs and improvement goals.	Manages operations and transparency.	Joint visibility of system health and user experience.
Evolve	Learn from outcomes and feedback to improve product, process, and team effectiveness.	Aligned Renewal → Strategic Partnership	Sponsors roadmap and innovation.	Leads improvement cycles, proposes experiments.	Continuous learning, data-driven value refinement.

Maturity across the SDLC reflects **how well flow, accountability, and learning are synchronized**.
Gaps between stages reveal where governance or communication must adapt.

Structural, Operational, and Relational Dimensions

Dimension	Definition	Observable Indicators
Structural Maturity	Clarity of process, ownership, and decision boundaries.	Roles and responsibilities are defined, governance cadence is predictable.
Operational Maturity	Effectiveness of practices and flow in maintaining stability and quality.	Stable velocity, working automation, consistent delivery predictability.
Relational Maturity	Quality of collaboration and transparency between Client and Vendor.	Trust indicators, open communication, joint problem-solving behaviors.

The **Structural** dimension establishes order,
the **Operational** dimension sustains flow,
and the **Relational** dimension drives long-term value.

Assessing SDLC Stage Maturity

Maturity can be assessed per stage or holistically through:

- **RAC (Rule Audit Checklist):** Structural and operational rule compliance.
- **Feedback Indicators:** Lead time, quality metrics, risk signals.
- **Relationship Health:** Trust and alignment feedback between Client and Vendor.

Each stage's maturity score feeds into a *delivery health profile* that highlights which capabilities enable or block overall system flow.

When Maturity Plateaus

Some stages evolve faster than others — for example, Build may reach high autonomy while Shape or Run remains reactive.
Such asymmetry often indicates systemic bottlenecks, not team incompetence.
3SF views these plateaus as signals to inspect **interfaces between stages** — where ownership, feedback, or governance may still operate transactionally.

Summary

- SDLC Stages Maturity connects **delivery quality** with **relationship growth**.
- Each stage advances from controlled execution to adaptive partnership.
- Balance across structural, operational, and relational dimensions ensures the system remains stable and learning-driven.
- Maturity is not about perfection — it's about **adaptive coherence**: the ability to stay aligned while continuously improving.

SDLC Practices

Overview

The **SDLC Practices** represent six cross-cutting disciplines that sustain quality, predictability, and learning throughout all stages of delivery.
Each practice complements the others — forming the behavioral and technical foundation of a reliable delivery system.

These practices act as the *horizontal layer* of the 3SF — spanning all SDLC Stages (Discover → Evolve) to ensure alignment, quality, and continuous improvement.

The Six SDLC Practices

Practice	Definition	Core Focus
Product Thinking	We co-create clarity and direction with clients by defining goals, value, and outcomes before committing to delivery.	Shared understanding of *why* we build.
Architecture & Design	We shape scalable, secure, and adaptable solutions by aligning technical design with product intent and constraints.	Fit-for-purpose design and feasibility.
Engineering & Quality	We build maintainable, high-quality software through craftsmanship, automation, and shared responsibility.	Consistency, reliability, and craftsmanship.
DevOps & Delivery	We enable fast, safe, and repeatable delivery by integrating automation, environments, and monitoring from day one.	Speed and stability through automation and visibility.
Governance & Risk	We manage complexity and protect value by embedding compliance, security, and accountability into every step.	Transparency, predictability, and responsibility.
Feedback & Learning	We continuously improve by listening to signals from users, systems, and teams — turning insight into better outcomes.	Growth, reflection, and adaptability.

Practices as the Delivery Backbone

These practices work across all SDLC Stages — ensuring continuity from discovery to sustainment.
They **stabilize delivery flow** by balancing speed, quality, and adaptability.

- **Product Thinking** brings clarity of purpose.
- **Architecture & Design** ensures technical feasibility.
- **Engineering & Quality** secures craftsmanship and maintainability.
- **DevOps & Delivery** provides automation and reliability.
- **Governance & Risk** maintains control without bureaucracy.
- **Feedback & Learning** closes loops between product, people, and outcomes.

Together, they form the **delivery backbone** of 3SF — connecting intent (why), structure (how), and learning (what next).

Practices Across SDLC Stages

Each SDLC Practice manifests differently depending on the stage of delivery.
The table below shows how every practice supports the system throughout its flow.

Stage →	Discover	Shape	Build	Validate	Release	Run
Product Thinking	Define goals, outcomes, and user needs.	Align scope and priorities with value.	Keep user outcomes visible during development.	Validate value and user satisfaction.	Communicate impact to stakeholders.	Monito and ad
Architecture & Design	Explore high-level approaches and constraints.	Define architecture vision and feasibility.	Implement patterns and integrations.	Validate non-functional requirements.	Harden deployment architecture.	Ensure and rel
Engineering & Quality	Evaluate feasibility and risks.	Prepare automation and QA strategy.	Build with test coverage and code quality.	Run functional and security tests.	Stabilize and fix regressions.	Handle and perfor
DevOps & Delivery	Define automation vision and environments.	Configure pipelines and CI/CD baselines.	Automate builds and integration.	Integrate testing and staging.	Execute deployments and rollback.	Monito infrast and up
Governance & Risk	Define governance and decision cadence.	Validate feasibility vs. budget and compliance.	Track progress and scope change.	Ensure acceptance and compliance sign-offs.	Control approvals and audits.	Manag operati
Feedback & Learning	Capture assumptions and unknowns.	Validate feasibility and stakeholder feedback.	Collect delivery metrics for improvement.	Measure outcomes and quality.	Gather user feedback post-release.	Analyz and us

This table visualizes how practices **connect horizontally** across the lifecycle, while SDLC Stages define the **vertical flow** of value.
Mature systems maintain consistency of these practices even when the project emphasis shifts between stages.

Practice Interdependence

No single practice guarantees success; the system works when all are balanced:

When practice X is weak...	Resulting system effect
Product Thinking	Misaligned priorities, wasted effort.
Architecture & Design	Fragile or over-engineered solutions.
Engineering & Quality	Unpredictable releases, rework cycles.
DevOps & Delivery	Manual bottlenecks, low feedback speed.
Governance & Risk	Unclear accountability, unmanaged debt.
Feedback & Learning	Stagnation and repeated mistakes.

A mature team develops *healthy tension* among practices — trading off intentionally instead of neglecting one dimension.

When Practices Compete

In practice, organizations often over-optimize a single discipline — seeking efficiency, control, or speed at the expense of balance.
Common patterns include:

- **DevOps & Delivery** dominating Engineering & Quality, resulting in automated deployment of unstable code.
- **Governance & Risk** overshadowing Product Thinking, leading to compliance without innovation.
- **Architecture & Design** overemphasized early, creating rigidity that slows later adaptation.

Such competition signals **systemic imbalance**, not bad intent.
The corrective action is to revisit the shared outcomes and ensure every practice still serves the product's purpose, not its own metrics.

Practices as a Foundation for Maturity

As projects evolve, each practice demonstrates maturity through visible behaviors:

Practice	Early Maturity Behavior	Advanced Maturity Behavior
Product Thinking	Feature focus, limited validation.	Outcome focus, evidence-based decisions.
Architecture & Design	Ad-hoc patterns, reactive fixes.	Intentional evolution, transparent trade-offs.
Engineering & Quality	Manual testing, quality owned by QA.	Shared quality ownership, automation first.
DevOps & Delivery	Manual deployments, unstable environments.	Full CI/CD automation, continuous observability.
Governance & Risk	Escalation-based control, unclear roles.	Embedded governance, proactive risk sharing.
Feedback & Learning	Occasional retrospectives.	Continuous data-driven learning loops.

These behaviors become inputs to the **RAC** and **CRC**, ensuring maturity assessments reflect *system reality* — not just process compliance.

Summary

- SDLC Practices are the **operating disciplines** that make each stage reliable and adaptive.
- The **Stages–Practices Matrix** shows how these disciplines interact through the lifecycle, ensuring flow and stability.
- Balanced practices ensure that progress is **fast, visible, and sustainable**.
- Together, they form the continuous backbone of the **3SF delivery system**, enabling predictable execution and continuous learning.

SDLC Practices Maturity

Purpose

The **SDLC Practices Maturity** model describes how each of the six practices develops from isolated effort to an integrated discipline that sustains the entire SDLC system.
It provides a lens for evaluating delivery integrity and learning capability across projects, portfolios, or teams.

Practice Maturity Levels

Level	Description	Behavioral Characteristics
1. Foundational Awareness	Practice exists but is inconsistent or reactive.	Knowledge varies by person; success depends on heroics; quality unstable.
2. Defined and Repeatable	Practice has structure and shared understanding.	Common templates, predictable routines, partial feedback usage.
3. Embedded and Measurable	Practice embedded in day-to-day delivery; outcomes tracked.	Teams self-correct using data; governance observes trends, not incidents.
4. Adaptive and Evolving	Practice continuously improves through feedback and innovation.	Cross-practice learning loops; teams experiment and adjust intentionally.

A project or team can mature unevenly — for example, strong in **Engineering & Quality (L3)** but weak in **Governance & Risk (L1)**.
The goal is not uniformity but **functional balance**.

Maturity by Practice

Practice	Early Maturity Behavior	Advanced Maturity Behavior	Indicators of Growth
Product Thinking	Focus on features and deadlines; limited validation of user value.	Focus on measurable outcomes; hypotheses validated through feedback.	Documented goals, value metrics, user feedback cycles.
Architecture & Design	Decisions implicit or reactive; tech debt grows unnoticed.	Design evolves intentionally; decisions made transparently with trade-offs visible.	Architecture review cadence, traceability of technical rationale.
Engineering & Quality	Manual testing; fragmented ownership of quality.	Quality automated and shared; engineering metrics drive improvement.	Test coverage, defect trends, build stability.
DevOps & Delivery	Manual deployments, unreliable environments.	Automated pipelines, stable environments, real-time observability.	Deployment frequency, MTTR, lead time for changes.
Governance & Risk	Bureaucratic or absent governance; unclear accountability.	Embedded governance with lightweight controls and shared responsibility.	Decision logs, compliance checks, escalation lead time.
Feedback & Learning	Retrospectives rare; insights not tracked.	Continuous feedback loops inform roadmap and process design.	Improvement actions tracked and completed, system metrics evolve.

Cross-Practice Synergy

Maturity in one practice reinforces others:

If this practice matures...	It strengthens...	By...
Product Thinking	Architecture & Design	Aligning solution design with user value.
Engineering & Quality	DevOps & Delivery	Creating reliable automation and faster feedback.
Governance & Risk	All others	Ensuring decisions remain transparent and aligned.
Feedback & Learning	The entire SDLC System	Turning insights into next-cycle improvements.

High-performing teams reach **cross-practice coherence**, where decisions made in one discipline positively impact all others.

When Maturity Inverts

Sometimes, a practice advances faster than its supporting ecosystem — for example, sophisticated DevOps automation in a project lacking Product Thinking or Governance clarity.
This **maturity inversion** can produce friction: faster delivery of unclear value, or automation that amplifies misaligned priorities.
3SF treats these cases as alignment gaps, not regressions — the goal is to synchronize maturity across disciplines rather than maximize any one in isolation.

Assessing Practice Maturity

Assessment combines **qualitative feedback** (behaviors, attitudes) and **quantitative indicators** (metrics, cycle data):

- **RAC linkage:** Stable Rules (R1–R12) mapped to practice maturity signals.
- **CRC linkage:** Contextual archetypes define target maturity profiles for each practice under different project types.
- **Self-assessment cadence:** Quarterly reflection on practice evolution per project or team.

Outputs form a **Practice Maturity Radar**, helping visualize strengths, gaps, and next improvement actions.

Summary

- SDLC Practices Maturity defines *how deeply* each discipline shapes delivery success.
- Practices evolve from awareness to adaptability, reinforcing one another.
- Balanced maturity across practices ensures the SDLC system is **self-correcting, transparent, and learning-oriented**.
- Continuous improvement of practices equals sustained organizational resilience.

SDLC Stage Dimentions

Purpose

The **SDLC Stage Dimensions** define the configurable decision spaces within each stage of delivery.
They help Project Leads, Product Managers, and Architects **diagnose, design, and adjust** how each SDLC Stage operates in context — aligning delivery mechanics with project realities.

Each dimension represents a *set of variables* that influence how work flows through the system.
By understanding and tuning these dimensions, teams can balance speed, quality, and adaptability while maintaining system integrity.

What Stage Dimensions Are

A **Stage Dimension** is a lens for examining *how a stage is executed* — not *what activities occur* within it.
Each dimension captures a set of interdependent conditions such as scope definition, estimation strategy, feedback loops, or risk handling.

Dimensions can be viewed as **levers**:

- When pulled in one direction, they improve stability or predictability.
- When pulled in another, they enable adaptability or speed.

The right balance depends on context, constraints, and maturity.

The Role of Dimensions in 3SF

Within the 3SF structure:

Layer	Purpose
SDLC Stages	Define what happens across the lifecycle.
SDLC Practices	Define how quality and flow are sustained across all stages.
SDLC Stage Dimensions	Define *how each stage behaves* — the adjustable parameters and decision patterns.

Dimensions translate abstract principles (like “governance clarity” or “learning loops”) into **practical, observable variables** that can be inspected or tuned.
They also serve as the **bridge between Contextual Drivers (CDL)** and **Stable Rules (SRL)** — the place where external conditions meet internal system design.

Dimension Categories

Each stage has its own unique decision areas, but all Stage Dimensions fall under **five universal categories** that describe different aspects of delivery configuration:

Category	Definition	Typical Examples
Strategic Alignment	Ensures that stage activities connect to the project's purpose, value, and goals.	Vision clarity, outcome definition, success metrics.
Planning & Flow	Defines how work is scoped, estimated, sequenced, and tracked.	Backlog depth, WIP limits, cadence of planning.
Collaboration & Communication	Governs information flow, stakeholder involvement, and decision-making transparency.	RACI, ceremony design, escalation paths.
Quality & Risk Management	Defines how uncertainty, testing, and verification are handled within the stage.	Definition of done, testing scope, approval criteria.
Learning & Adaptation	Enables reflection, improvement, and responsiveness to change.	Feedback cadence, review structure, continuous improvement loops.

These categories apply to **every stage**, but the specific decisions and trade-offs within each depend on stage purpose and context.

Example: How Dimensions Interconnect

For instance, in the **Shape** stage:

- *Strategic Alignment* defines outcomes and prioritization.
- *Planning & Flow* translates them into epics and estimates.
- *Collaboration* defines how client and vendor make trade-offs.
- *Quality & Risk* determines acceptance conditions and architectural feasibility.
- *Learning & Adaptation* captures feedback from early discovery or prototypes.

When these dimensions align, Shape provides a stable foundation for Build.
When they misalign, Build inherits ambiguity, rework, and unvalidated assumptions.

Why Dimensions Matter

Without dimensions, delivery tuning becomes guesswork.
Projects often fail not because teams lack competence, but because their stages are configured incorrectly for the environment.

Common failure modes:

- Discovery too narrow → Shape inherits false assumptions.
- Build too rigid → Validation discovers issues too late.
- Run disconnected from Learn → No systemic improvement.

Stage Dimensions bring **diagnostic clarity** — enabling teams to identify not only *what's broken*, but *why* it behaves that way.

Relationship to Maturity

Maturity across the SDLC is visible through the **health of Stage Dimensions**:

- *Immature stages* show fragmented, reactive, or unowned dimensions.
- *Mature stages* show integrated, stable, and continuously improving dimensions.

When assessing maturity through the **RAC** or **CRC**, each rule and contextual archetype ultimately maps to one or more Stage Dimensions.
For example:

- **Stable Rule R3 – "Decisions are transparent and reversible"** → affects Collaboration and Quality dimensions.
- **Contextual Driver – "Fixed bid contract"** → constrains Planning & Flow, influencing estimation and governance.

Applying Stage Dimensions

Use Stage Dimensions for:

1. **Design:** Before execution, define the configuration per stage (e.g., estimation depth, backlog readiness, governance model).
2. **Assessment:** During delivery, inspect dimension health through metrics, signals, and team reflection.
3. **Adaptation:** After retrospectives or milestones, adjust dimension design to improve flow and maturity.

Teams can document stage dimensions in simple templates or dashboards — linking them to risks, metrics, and improvement actions.

When Dimensions Become Bureaucracy

Stage Dimensions are diagnostic lenses — not checklists to be filled or enforced.
A common failure pattern is **over-engineering**: documenting every parameter without actually improving flow or clarity.
When dimensions turn into reporting artifacts, they lose their adaptive power.
The goal is not to manage the framework — it is to manage the system it describes.

Structure of Stage Dimension Files

Each stage has a dedicated section detailing:

- The stage's **core purpose and outcomes**.
- Its **dimension breakdown** under the five categories.
- Typical **configuration options**, **failure patterns**, and **improvement strategies**.

Stage	Purpose
Discover	How to frame problems and align understanding before shaping.
Shape	How to define scope, feasibility, and delivery models.
Build	How to structure flow, engineering quality, and visibility.
Validate	How to manage verification, acceptance, and joint confidence.
Release	How to manage approvals, risk, and deployment readiness.
Run & Evolve	How to sustain and improve systems post-launch.

Each file builds upon this overview, creating a **consistent pattern** that allows readers to trace the evolution of alignment, flow, and learning across the SDLC.

Summary

- **SDLC Stage Dimensions** describe how each stage functions in context — the levers and trade-offs teams must balance.
- They provide **diagnostic visibility** and **design flexibility**, connecting strategy, delivery, and governance.
- Dimensions bridge contextual forces (CDL) and systemic stability (SRL), serving as the *operational DNA* of the 3SF delivery system.
- Mastering dimension design is the key to transforming projects from reactive execution into adaptive, learning systems.

SDLC Stage Dimensions – Discover

Purpose

The **Discover Stage** establishes the foundation for the entire delivery system.
Its goal is to **understand the problem**, align stakeholders on goals, and explore possible solution directions before committing to execution.

Discovery defines *what success means*, *for whom*, and *why*.
It transforms assumptions into shared understanding — reducing uncertainty, building trust, and shaping realistic next-stage decisions (Shape, Build).

When done well, Discover prevents most downstream delivery issues: unclear scope, missing validation, misaligned expectations, and low confidence in estimates.

Core Outcomes

Outcome	Description
Clarity of Purpose	A shared understanding of the problem, goals, and expected value.
Context Map	Documented business, user, and technical context (constraints, systems, dependencies).
Validated Assumptions	Key unknowns identified and tested via interviews, prototypes, or data.
Discovery Artifacts	Early deliverables such as user journeys, high-level architecture options, and estimation boundaries.
Stakeholder Alignment	Agreement on next steps, ownership, and engagement model for the Shape stage.

Discovery Dimensions

1. Strategic Alignment

Defines how well the problem and goals are understood and connected to business outcomes.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Problem Definition	Vague, solution-driven (“we need an app”).	Clearly defined, measurable business problem.
Goal Alignment	Conflicting stakeholder objectives.	Shared outcomes defined in measurable terms.
Value Hypothesis	Assumed benefits, no validation.	Explicit value model with metrics and dependencies.
User Understanding	Minimal or second-hand knowledge.	Direct discovery with users, empathy-based insights.

Improvement Strategies

- Use *Product Thinking* workshops to define outcomes and KPIs.
- Apply *Value Mapping* and *Wardley Mapping* to connect goals and capabilities.
- Ensure goals are validated with both business and delivery stakeholders.

2. Planning & Flow

Defines how discovery activities are structured, timeboxed, and prioritized.

Aspect	Low Maturity	High Maturity
Discovery Approach	Unstructured discussions or scattered workshops.	Timeboxed discovery plan with clear goals and outputs.
Work Sequencing	Parallel ideation without validation.	Incremental exploration: learn → converge → define.
Estimation Readiness	Estimation done without validated scope.	Feasibility and complexity assessed through validated artifacts.
Decision Cadence	Decisions deferred or made ad hoc.	Defined checkpoints with documented assumptions.

Improvement Strategies

- Apply *Double Diamond* or *Dual Track* discovery frameworks.
- Visualize discovery backlog and outcomes (e.g., Notion, Miro, Jira).
- Include estimation boundaries early — treat estimates as hypotheses.

3. Collaboration & Communication

Defines how teams, clients, and stakeholders interact to co-create understanding.

Aspect	Low Maturity	High Maturity
Stakeholder Engagement	Sporadic meetings, information silos.	Inclusive workshops, active participation from both sides.
Roles & Responsibility	Undefined — discovery “owned” by one party.	Shared accountability between Client and Vendor teams.
Communication Cadence	Irregular, reactive.	Scheduled sessions with outcomes shared transparently.
Decision Logging	Verbal agreements only.	Structured documentation (decisions, rationale, next actions).

Improvement Strategies

- Establish *Engagement Agreement* — define discovery ownership, cadence, tools.
- Use a shared *Decision Log* or *Discovery Journal*.
- Align technical and product language — reduce translation gaps.

4. Quality & Risk Management

Defines how uncertainty and validation are handled during exploration.

Aspect	Low Maturity	High Maturity
Assumption Management	Risks and unknowns undocumented.	Key assumptions identified, tracked, and tested.
Feasibility Validation	Technical feasibility assumed or skipped.	Feasibility validated via quick prototypes or proof-of-concept.
Scope Boundaries	Discovery expands uncontrollably.	Clear timebox and prioritization of discovery goals.
Risk Sharing	Risks pushed to the next stage.	Risks explicitly shared and mitigation planned jointly.

Improvement Strategies

- Maintain an *Assumptions Board* (risks, evidence, actions).
- Introduce *Rapid Feasibility Tests* (technical spikes, data validation).
- Use a fixed discovery budget with a prioritized scope to maintain focus.

5. Learning & Adaptation

Defines how feedback is captured and turned into better decisions.

Aspect	Low Maturity	High Maturity
Feedback Loops	None or informal ("we'll figure it out later").	Frequent validation with stakeholders or users.
Knowledge Capture	Insights lost in meeting notes.	Structured summary: what we learned, what changed.
Retrospective Practice	Post-discovery review skipped.	Retrospective held to evaluate discovery effectiveness.
Transition Readiness	Shape starts with unvalidated backlog.	Clear handover including context, validated artifacts, and assumptions.

Improvement Strategies

- Document *Discovery Learnings* with impact level and next-step recommendation.
- Run a short *Discovery Retrospective* before Shape.
- Ensure all findings are summarized in a *Discovery Report* template for reuse.

Common Failure Modes

Failure Mode	Root Cause	Correction
"We started building too early."	Discovery cut short or skipped.	Enforce discovery exit criteria before Shape begins.
"We didn't know this dependency existed."	Missing system or stakeholder mapping.	Include dependency identification as a required discovery artifact.
"The client expected X, but we delivered Y."	Goals misaligned or undocumented.	Introduce measurable success metrics and stakeholder review.
"Estimates were way off."	Estimation based on unvalidated scope.	Validate assumptions and establish uncertainty range in estimates.

Over-extending discovery can be as harmful as skipping it — excessive exploration delays learning and erodes stakeholder confidence.

Measuring Discovery Health

Indicators of a healthy Discovery stage:

Signal	Description
Stakeholders can articulate the same goal in one sentence.	Alignment and clarity achieved.
All key assumptions are documented and validated or prioritized for testing.	Systemic thinking established.
Feasibility validated with quick technical or design prototypes.	Confidence increased for Shape.
Discovery report reviewed and accepted by both Client and Vendor.	Mutual commitment achieved.

Quantitative indicators may include:

- % of validated assumptions vs. total identified
- Time spent in Discovery vs. number of unresolved questions
- Confidence score in estimate readiness (self-assessed 1–5 scale)

Discovery and Relationship Maturity

Discovery is where **Transactional Trust** evolves into **Aligned Autonomy**.
Transparency, open communication, and early co-ownership set the tone for the entire engagement.

High-maturity discovery:

- Establishes **shared understanding** instead of deliverables only.
- Builds **trust through visibility**, not control.
- Creates **momentum for partnership** before contractual delivery begins.

Summary

- The **Discover Stage** is about learning, not committing — converting uncertainty into clarity.
- Its five dimensions (Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation) ensure balanced exploration.
- Discovery maturity is measured by *how well understanding, alignment, and validation are achieved together*.
- A disciplined, collaborative discovery is the single most powerful predictor of downstream delivery success.

SDLC Stage Dimensions – Shape

Purpose

The **Shape Stage** defines *how the vision becomes feasible* — transforming ideas and validated insights from Discovery into a **delivery-ready plan**.
It establishes **scope, architecture vision, delivery approach, and feasibility**, aligning all parties on what will be built, how, and under which constraints.

Shape is where **strategy meets execution**.
It converts the "why" and "what" discovered earlier into "how" — preparing teams, stakeholders, and systems for sustainable delivery flow.
When done well, Shape minimizes delivery risk, sets realistic expectations, and becomes the contract of shared understanding between Client and Vendor.

Core Outcomes

Outcome	Description
Delivery Vision	A coherent, outcome-aligned vision that translates business goals into tangible delivery objectives.
Scope Definition	A prioritized backlog or roadmap describing what will be built, when, and why.
Architecture Vision	A documented target architecture and rationale aligned with scalability, security, and maintainability.
Feasibility & Estimation	Validated assumptions, complexity estimates, and resource forecasts.
Delivery Model & Governance	Clear engagement model, communication cadence, and decision-making process.

Shape Dimensions

1. Strategic Alignment

Defines how well Shape connects the validated Discovery findings to actionable scope and measurable outcomes.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Goal Translation	Discovery outputs not reflected in scope.	Goals translated into product outcomes and acceptance metrics.
Prioritization Logic	Everything is "must-have."	Prioritization driven by value, risk, and effort balance.
Outcome Traceability	Delivery tasks disconnected from strategic goals.	Each backlog item traces to a defined business outcome.
Vision Communication	Technical or business bias dominates.	Shared, human-readable vision understood by all stakeholders.

Improvement Strategies

- Use *VMOSA* (Vision, Mission, Objectives, Strategies, Actions) or *Impact Mapping*.
- Apply *Outcome-based Roadmapping* — link objectives to measurable impacts.
- Create a *single-page delivery vision* reviewed jointly with all key stakeholders.

2. Planning & Flow

Defines how scope, estimation, and delivery model are structured to balance feasibility with adaptability.

Aspect	Low Maturity	High Maturity
Scope Definition	Overly detailed or incomplete.	Balanced between high-level themes and well-shaped backlog.
Estimation Method	Based on assumptions, not validated data.	Multi-method estimation (PERT, T-shirt sizing, story points) with uncertainty range.
Delivery Model Choice	Model dictated by contract or inertia.	Model adapted to context (Fixed-Bid, Time & Material, Co-Delivery).
Roadmap Structure	Dates arbitrary, dependencies unclear.	Milestones based on value increments and dependency analysis.

Improvement Strategies

- Introduce *progressive estimation* — refine scope iteratively.
- Use *Scenario Planning* for budget and timeline alignment.
- Align delivery model choice with Contextual Drivers (commercial, organizational, technical).

3. Collaboration & Communication

Defines how decisions are co-created and communicated between client and vendor roles.

Aspect	Low Maturity	High Maturity
Decision-Making	Centralized, delayed, or ambiguous.	Distributed decisions within defined RACI and escalation paths.
Stakeholder Involvement	Limited to approvals.	Stakeholders actively contribute to trade-offs and prioritization.
Communication Flow	Channel chaos, undocumented agreements.	Clear communication map, meeting cadences, and shared documentation.
Cross-Discipline Integration	Product, design, and engineering operate separately.	Joint shaping sessions, shared artifacts, and synchronized priorities.

Improvement Strategies

- Define *Governance Cadence* — e.g., weekly sync, bi-weekly steering.
- Maintain a *Shape Log* capturing decisions, rationales, and risks.
- Use *Collaboration Maps* to visualize stakeholder engagement and information flow.

4. Quality & Risk Management

Defines how feasibility, architecture, and delivery risks are identified and controlled before Build starts.

Aspect	Low Maturity	High Maturity
Architecture Vision	Implicit or over-engineered; lacks trade-offs.	Architecture designed collaboratively with rationale and constraints.
Feasibility Testing	Feasibility assumed, not validated.	Risk spikes and POCs used to confirm critical assumptions.
Technical Debt Awareness	None — "we'll fix it later."	Debt potential logged with mitigation options.
Definition of Ready	Missing or vague.	Agreed entry criteria for Build with clear acceptance boundaries.

Improvement Strategies

- Apply *Architecture Decision Records (ADR)* and *Architecture Runway*.
- Create *Feasibility Register* — track technical risk, cost, and decisions.
- Align readiness checklist with Build expectations (DoR, environments, design assets).

5. Learning & Adaptation

Defines how discovery insights, decisions, and risks evolve into an adaptive delivery mindset.

Aspect	Low Maturity	High Maturity
Feedback Integration	Discovery learnings ignored.	Discovery insights explicitly embedded into scope and architecture.
Reflection on Trade-offs	Decisions made without review.	Retrospective held to assess shaping quality and stakeholder alignment.
Adaptability to Change	Contract or timeline blocks adjustments.	Scope change process defined and integrated into governance.
Knowledge Continuity	Handovers inconsistent or undocumented.	Shape artifacts consolidated in accessible, shared repositories.

Improvement Strategies

- Conduct a *Shape Retrospective* after final sign-off.
- Maintain a *Decision Register* — track what changed, why, and impact.
- Use *Knowledge Handover Packages* (vision, architecture, roadmap) for Build readiness.

Common Failure Modes

Failure Mode	Root Cause	Correction
"We're shaping while building."	Discovery incomplete, Shape rushed.	Separate timeboxes and exit criteria for each stage.
"The backlog is too vague to start."	Missing DoR and architecture clarity.	Enforce readiness validation and backlog refinement.
"Estimates keep changing."	Scope or assumptions not validated.	Document estimation assumptions and uncertainty ranges.
"Client expects more than planned."	Value and scope poorly communicated.	Present scope through outcomes, not features; revalidate alignment.

The illusion of precision — detailed plans built on unvalidated inputs — is a recurring trap. Shape aims for confidence, not certainty.

Measuring Shape Health

Signal	Description
Shared delivery vision and roadmap approved by both Client and Vendor.	Alignment achieved.
Architecture and feasibility validated via ADRs and POCs.	Technical confidence established.
Prioritization matrix links outcomes, effort, and risk.	Informed decisions made.
Definition of Ready checklist accepted by Build team.	Clear transition into Build.

Quantitative indicators:

- % of backlog items meeting DoR criteria.
- Estimation confidence range (± deviation vs. actual).
- Stakeholder alignment score (via feedback survey).

Shape and Relationship Maturity

The Shape stage moves the relationship from **Aligned Autonomy** toward **Shared Ownership**.
Here, trust grows through *joint decision-making* and *transparent trade-offs*.

High-maturity shaping:

- Builds shared accountability for both outcomes and constraints.
- Replaces handovers with collaboration.
- Aligns product ambition with delivery feasibility.

Summary

- The **Shape Stage** translates understanding into a feasible, outcome-driven delivery plan.
- Its five dimensions — Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation — ensure scope, architecture, and delivery model coherence.
- Mature shaping ensures Build starts with confidence, autonomy, and transparency.
- When Shape fails, all later stages pay the cost — when it succeeds, the system flows.

SDLC Stage Dimensions – Build

Purpose

The **Build Stage** is where ideas, designs, and plans turn into a working product.
Its purpose is to **design, develop, and integrate** software components with automation, quality, and shared ownership at the core.

Build is not only about coding — it's about maintaining **flow, visibility, and alignment** across teams, ensuring each increment delivers measurable progress and validated value.
When executed maturely, Build becomes the engine of both **technical excellence** and **relationship trust** between Client and Vendor.

Core Outcomes

Outcome	Description
Working Product Increments	Functional, testable software aligned with scope and goals.
Predictable Delivery Flow	Stable sprint or iteration cadence with transparent progress tracking.
Quality Built In	Testing, automation, and validation integrated into daily development.
Visible Progress	Delivery transparency through demos, metrics, and shared dashboards.
Engineering Feedback Loops	Continuous feedback on quality, velocity, and technical debt.

Build Dimensions

1. Strategic Alignment

Ensures that Build execution remains connected to business outcomes and product intent.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Goal Visibility	Teams focus on tickets, not outcomes.	Developers understand why features matter to users.
Scope Discipline	Frequent scope creep and unclear priorities.	Stable, prioritized backlog tied to objectives.
Value Focus	Progress measured by volume of work.	Progress measured by value delivered or validated.
Decision Context	Engineers disconnected from product strategy.	Engineers involved in trade-off and design discussions.

Improvement Strategies

- Share *Product Vision Boards* and outcome maps with delivery teams.
- Use *Story Mapping* to link user flows to business goals.
- Include engineers in scope and backlog refinement sessions.

2. Planning & Flow

Defines how work is decomposed, planned, and executed to maintain a sustainable pace and predictable outcomes.

Aspect	Low Maturity	High Maturity
Backlog Readiness	Incomplete or unclear stories.	Clear DoR (Definition of Ready) and consistent backlog grooming.
Iteration Planning	Overcommitted or ad hoc scope.	Predictable velocity and adaptive capacity planning.
Work in Progress (WIP)	Too many parallel tasks, context switching.	Limited WIP and visible flow management (Kanban boards, flow metrics).
Dependencies	Managed reactively.	Dependencies identified, visualized, and mitigated early.

Improvement Strategies

- Apply *Kanban* or *Scrum with Flow Metrics*.
- Use *Cumulative Flow Diagrams* to spot bottlenecks.
- Introduce *Sprint Capacity Planning* and WIP limits by team role.

3. Collaboration & Communication

Defines how the team synchronizes, collaborates, and manages feedback during execution.

Aspect	Low Maturity	High Maturity
Team Communication	Silos between roles; issues discussed too late.	Daily visibility, open problem-solving culture.
Stakeholder Updates	Irregular or one-way reporting.	Regular demos, transparent dashboards, two-way communication.
Cross-Functional Collaboration	Developers, designers, testers operate separately.	Integrated squads collaborating around features or epics.
Decision Transparency	Decisions undocumented or repeated.	Decisions visible and logged in shared tools (ADR, issue trackers).

Improvement Strategies

- Encourage *pair programming*, *mob testing*, and *cross-role reviews*.
- Establish a *shared status dashboard* visible to Client and Vendor.
- Maintain a *Team Working Agreement* covering communication, review, and escalation norms.

4. Quality & Risk Management

Ensures that the system remains stable and maintainable while progress accelerates.

Aspect	Low Maturity	High Maturity
Testing Approach	Manual, end-stage QA only.	Continuous testing: unit, integration, end-to-end automated pipelines.
Code Quality	Inconsistent standards, unreviewed code.	Shared coding standards, code review culture, linters and static analysis.
Defect Management	Bugs found late or ignored.	Early detection, visible defect metrics, root cause analysis.
Technical Debt Management	Reactive firefighting, no refactoring time.	Debt logged, prioritized, and managed as part of delivery.

Improvement Strategies

- Adopt *Test Automation Pyramid* and *Shift-Left Testing*.
- Use *SonarQube* or similar tools for code health tracking.
- Schedule *Tech Debt Sprints* or allocate engineering capacity for refactoring.

5. Learning & Adaptation

Defines how the team inspects performance, learns, and improves during the Build cycle.

Aspect	Low Maturity	High Maturity
Retrospectives	Irregular or unfocused.	Regular, data-informed retrospectives with actionable outcomes.
Metrics Utilization	Metrics collected but not used.	Metrics analyzed and drive measurable process adjustments.
Knowledge Sharing	Isolated expertise.	Knowledge documented and spread through pairing, wikis, and reviews.
Continuous Improvement	Process changes reactive to crises.	Improvement backlog prioritized and tracked continuously.

Improvement Strategies

- Use *Team Health Checks* and *Agile Metrics Dashboards* (lead time, defects, predictability).
- Rotate responsibilities to spread system knowledge.
- Introduce *Learning Hours* or short technical demos during sprints.

Common Failure Modes

Failure Mode	Root Cause	Correction
"Velocity fluctuates wildly."	Unclear backlog, overcommitment, or interruptions.	Limit WIP, improve backlog readiness, stabilize team focus.
"Quality issues pile up near release."	Testing deferred or manual only.	Shift-left automation, enforce DoD with test coverage.
"Developers disengaged from product goals."	Poor communication or siloed decision-making.	Include engineers in planning and stakeholder reviews.
"Integration issues late in cycle."	Lack of CI/CD or shared environments.	Adopt trunk-based development and continuous integration.

When flow metrics become performance targets, teams start optimizing numbers instead of outcomes. 3SF treats metrics as feedback, not judgment.

Measuring Build Health

Signal	Description
Stable velocity and flow metrics across iterations.	Predictable delivery pattern achieved.
High automation coverage and quick CI/CD feedback cycles.	Engineering efficiency increasing.
Low defect escape rate between environments.	Quality maturity improving.
Transparent backlog and demo cadence maintained.	Alignment and visibility ensured.

Quantitative metrics may include:

- Lead time and cycle time stability.
- % of automated tests and deployment success rate.
- Ratio of refactoring vs. new feature work.
- Sprint predictability (committed vs. delivered).

Build and Relationship Maturity

The Build stage matures the relationship from **Shared Ownership** to **Strategic Partnership**.
It's where autonomy is proven through reliable delivery, and trust deepens through transparency and quality.

High-maturity Build means:

- Teams operate with **clarity and independence**, not isolation.
- Delivery decisions are **coordinated, not dictated**.
- Transparency builds confidence — enabling Client and Vendor to plan forward together.

Summary

- The **Build Stage** is where structure meets execution — converting vision into reality with precision and accountability.
- Its five dimensions — Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation — define the maturity and stability of delivery.
- Build maturity determines system reliability and organizational trust.
- Strong Build practices create a foundation for fast validation, smooth releases, and long-term partnership success.

SDLC Stage Dimensions – Validate

Purpose

The **Validate Stage** ensures that the product or increment delivered during Build truly meets its intended goals, quality standards, and user expectations.
It verifies that **functionality, performance, usability, and security** align with the outcomes defined in the earlier stages — and that the system is ready for release and adoption.

Validation provides the critical link between **engineering output and business value**.
It transforms delivery confidence from internal assumptions to **evidence-based assurance** — proving that what has been built actually works, delivers value, and is safe to deploy.

When mature, the Validate stage enables short feedback loops, measurable confidence, and shared accountability between Client and Vendor.

Core Outcomes

Outcome	Description
Functional Assurance	All intended features verified and behaving as expected.
Quality Confidence	Reliability, performance, and usability validated against standards.
Risk Mitigation Evidence	Known risks tested, mitigated, or consciously accepted.
User Acceptance	Product validated with users or proxies for real-world readiness.
Release Readiness	Delivery team and stakeholders jointly confirm deployment confidence.

Validate Dimensions

1. Strategic Alignment

Ensures that testing and validation activities map back to business outcomes and user intent — not just requirements coverage.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Validation Purpose	Focused on finding bugs.	Focused on confirming business value and usability.
Traceability	Test cases disconnected from goals.	Requirements, tests, and metrics linked to outcomes.
User Perspective	Validation done by developers or QA only.	Validation includes real users, personas, or user proxies.
Acceptance Criteria	Defined vaguely or retroactively.	Established early and verified continuously.

Improvement Strategies

- Create *Test-to-Value Mapping* linking acceptance criteria to user outcomes.
- Use *Example Mapping* and *BDD (Behavior-Driven Development)* to bridge business and technical validation.
- Include *Product Owners and Clients* in validation sessions.

2. Planning & Flow

Defines how validation activities are integrated into the overall delivery flow — balancing thoroughness with speed.

Aspect	Low Maturity	High Maturity
Validation Timing	Done at the end of the iteration or project.	Continuous and iterative throughout development.
Test Environment Management	Unstable or inconsistent.	Dedicated, reliable, and automatically provisioned environments.
Validation Cadence	Ad hoc, untracked.	Defined test cycle integrated into CI/CD pipeline.
Regression Handling	Manual or skipped due to time pressure.	Automated regression suite maintained and monitored.

Improvement Strategies

- Introduce *Continuous Testing* within CI/CD pipelines.
- Automate environment provisioning using *Infrastructure as Code (IaC)*.
- Establish *Validation Gates* for build quality before release.

3. Collaboration & Communication

Defines how validation activities are shared, reviewed, and acted upon across disciplines and organizations.

Aspect	Low Maturity	High Maturity
Client-Vendor Involvement	Validation owned by one side only.	Validation co-owned with shared metrics and sign-offs.
Defect Communication	Blame culture, unclear priorities.	Collaborative triage, root cause analysis, and transparency.
Feedback Channels	Issues discussed late or inconsistently.	Real-time communication via dashboards, alerts, or shared tools.
Validation Reporting	Reports hidden in tools or manual spreadsheets.	Automated, visual dashboards integrated into governance cadence.

Improvement Strategies

- Host *Joint Quality Reviews* after major validation cycles.
- Share *QA Dashboards* with live status for Client and Vendor.
- Define *Defect Escalation Path* in governance structure.

4. Quality & Risk Management

Defines how quality is measured, controlled, and used to inform decisions before release.

Aspect	Low Maturity	High Maturity
Test Coverage	Limited or unknown.	Comprehensive across unit, integration, end-to-end, and non-functional levels.
Risk-Based Testing	Testing all equally or at random.	Focused testing based on impact and likelihood of failure.
Non-Functional Testing	Skipped or manual.	Automated and benchmarked (performance, security, accessibility).
Release Confidence	Based on opinions.	Data-driven confidence supported by evidence and metrics.

Improvement Strategies

- Apply *Risk-Based Testing* for prioritization.
- Use *Performance Baselines* and *Security Scanning* early.
- Track *Defect Trends* and *Defect Escape Rate* across environments.

5. Learning & Adaptation

Defines how validation results contribute to learning and continuous improvement across teams and stages.

Aspect	Low Maturity	High Maturity
Defect Learning	Same issues reoccur across sprints.	Root cause analysis feeds into backlog and process improvement.
Metrics Utilization	QA metrics collected but ignored.	Quality metrics discussed in retrospectives and governance.
Validation Retrospectives	Skipped or informal.	Dedicated reflection on test coverage, efficiency, and confidence.
Cross-Stage Feedback	Lessons lost post-release.	Findings from Validate feed directly into Evolve and future discovery.

Improvement Strategies

- Introduce *Quality Retrospectives* after major releases.
- Maintain a *Defect Knowledge Base* with lessons and examples.
- Link validation learnings to *Continuous Improvement Roadmaps*.

Common Failure Modes

Failure Mode	Root Cause	Correction
“We found critical bugs right before release.”	Late or skipped validation cycle.	Shift validation left, integrate with CI/CD.
“Client says quality is poor despite passing tests.”	Validation disconnected from user value.	Reconnect tests to user outcomes and business scenarios.
“Test environments never match production.”	Manual setup or outdated configurations.	Automate provisioning and synchronization.
“Validation team overloaded.”	Testing not integrated into delivery flow.	Embed testing ownership across development teams.

Measuring Validation Health

Signal	Description
High regression automation coverage.	Quality controlled continuously, not reactively.
Stable defect escape rate across environments.	Fewer production surprises, mature QA process.
Joint validation sign-offs by Client and Vendor.	Shared accountability achieved.
Quality metrics used in retrospectives and decisions.	Continuous improvement mindset in place.

Quantitative metrics may include:

- Defect density and escape rate.
- Regression automation coverage (%).
- Test cycle duration and stability.
- User acceptance satisfaction rating.

Validate and Relationship Maturity

Validate reinforces **Shared Ownership** and advances toward **Strategic Partnership**.
It transforms quality from a contractual deliverable into a **shared mission of excellence**.

High-maturity validation means:

- The Client and Vendor assess readiness together, not separately.
- Confidence is built through transparency, not persuasion.
- Feedback is used for continuous learning, not blame assignment.

Summary

- The **Validate Stage** ensures confidence and alignment between product functionality and intended value.
- Its five dimensions — Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation — provide a systemic approach to testing and assurance.
- Mature validation replaces inspection with prevention and shared accountability.
- Validation done right creates trust: *evidence replaces opinion, collaboration replaces control*.

SDLC Stage Dimensions – Release

Purpose

The **Release Stage** is where validated increments are prepared, approved, and delivered to users.
Its purpose is to ensure that deployment is **reliable, transparent, and reversible** — maintaining confidence across all stakeholders.

Release represents the intersection of **technical readiness**, **business timing**, and **operational control**.
It is not just a one-time action but a repeatable system of governance, automation, and communication that turns delivery into value.

When mature, Release enables safe, frequent, and visible deployments that sustain trust between Client and Vendor.

Core Outcomes

Outcome	Description
Release Readiness	Product increments meet Definition of Done (DoD) and pass all validation gates.
Deployment Reliability	Automated, monitored, and reversible deployment pipelines in place.
Governance Alignment	Approvals, communication, and documentation aligned between Client and Vendor.
Change Transparency	What, when, and why changes happen is clearly visible.
Operational Continuity	Release executed without disruption or surprise for users or teams.

Release Dimensions

1. Strategic Alignment

Ensures that the release strategy supports business objectives, user expectations, and risk appetite.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Release Strategy	Treated as a technical event.	Planned as a business milestone with measurable outcomes.
Timing & Cadence	Ad hoc or delayed until “everything is ready.”	Predictable cadence aligned with value delivery (e.g., continuous or monthly).
Stakeholder Readiness	Client notified late or post-deployment.	Stakeholders aligned with communication plan and expected impact.
Business Impact Awareness	Impact assumptions unverified.	Business readiness checklist validated before go-live.

Improvement Strategies

- Use *Release Calendar* integrated into governance and roadmap planning.
- Define *Release Goals* that link technical changes to measurable value.
- Align *Release Cadence* with feedback and validation cycles.

2. Planning & Flow

Defines how release preparation, approval, and deployment are coordinated and automated.

Aspect	Low Maturity	High Maturity
Pipeline Automation	Manual steps and human dependencies.	Fully automated CI/CD with pre-deployment validation.
Approval Flow	Bureaucratic or unclear ownership.	Streamlined, traceable approvals in tools and dashboards.
Change Management	Reactive to issues, unplanned changes.	Structured change process with risk classification.
Rollback Procedures	Non-existent or untested.	Reversible releases validated regularly.

Improvement Strategies

- Automate deployment steps via *CI/CD pipelines*.
- Integrate *Change Management Automation* (ServiceNow, Azure DevOps, Jira).
- Test *Rollback Plans* as part of regression testing.
- Maintain a *Release Checklist* for every environment.

3. Collaboration & Communication

Defines how coordination between Client, Vendor, and Operations ensures smooth release flow and visibility.

Aspect	Low Maturity	High Maturity
Client-Vendor Coordination	Information shared post-release.	Joint release planning and readiness reviews.
Operations Involvement	Brought in only during incidents.	Operations co-own release readiness and monitoring.
Communication Plan	Email notifications or chat pings.	Structured release communication with who/what/when/impact.
Transparency	Changes poorly visible.	Single release dashboard showing release scope, status, and outcomes.

Improvement Strategies

- Establish *Joint Release Reviews* with Client and Vendor stakeholders.
- Maintain *Release Notes* in standardized, automated format.
- Use *Live Dashboards* or ChatOps to broadcast release status in real time.

4. Quality & Risk Management

Ensures release confidence through proactive quality control, security, and risk mitigation.

Aspect	Low Maturity	High Maturity
Release Testing	Regression and smoke tests skipped or rushed.	Automated validation in staging or pre-prod environments.
Security Checks	Manual or post-release.	Integrated security scans in the release pipeline.
Risk Assessment	Done informally or retroactively.	Risk classified and approved through governance process.
Release Criteria	Ambiguous or undocumented.	Formal Definition of Done and Definition of Ready for release.

Improvement Strategies

- Implement *Release Gates* in pipelines for test and security validation.
- Conduct *Go/No-Go Reviews* with risk-based criteria.
- Maintain a *Release Risk Register* and monitor risk trends.

5. Learning & Adaptation

Defines how the organization learns from each release to improve stability, cadence, and confidence.

Aspect	Low Maturity	High Maturity
Post-Release Review	Only held after major incidents.	Conducted after every release — focus on wins and learnings.
Metrics Utilization	Success judged by "no issues."	Success measured by stability, feedback, and recovery time.
Continuous Improvement	Repeated release problems tolerated.	Improvement backlog maintained and prioritized after every release.
Knowledge Sharing	Release learnings lost in chat threads.	Centralized release documentation and lessons learned.

Improvement Strategies

- Schedule *Release Retrospectives* as part of governance cadence.
- Track *Deployment Frequency*, *Change Failure Rate*, *MTTR*, and *Lead Time for Changes*.
- Feed learnings into *Run* and *Evolve* stages for system improvement.

Common Failure Modes

Failure Mode	Root Cause	Correction
"Releases are unpredictable and stressful."	No cadence, manual steps, unclear approvals.	Automate pipeline, define release rhythm, clarify roles.
"Client wasn't ready for changes."	Poor communication and no business readiness check.	Align release planning with stakeholder communication.
"Rollback failed."	Plan untested or absent.	Simulate rollback in lower environments and automate recovery.
"Post-release issues damage trust."	Lack of validation and incident transparency.	Create shared incident process and visible metrics.

A common hidden risk is release heroics — dependency on a few experts to 'save' the deployment. Sustainable maturity eliminates heroes by institutionalizing reliability.

Measuring Release Health

Signal	Description
Releases are regular, safe, and predictable.	Cadence aligned with business needs and system capacity.
Automated deployments and rollback tested successfully.	Reliability established.
No "surprise" changes for users or stakeholders.	Communication and readiness mature.
Incident recovery within agreed SLA.	Operational resilience proven.

Quantitative indicators may include:

- Deployment frequency (per environment).
- Change failure rate (% of releases causing incidents).
- Mean Time to Recover (MTTR).
- Lead time from commit to production.
- Number of manual release steps remaining.

Release and Relationship Maturity

The Release stage embodies **Strategic Partnership in action** — where Client and Vendor coordinate as one delivery organism.
Release is not a technical event but a **moment of shared accountability** for value, risk, and experience.

High-maturity release culture:

- Operates with **mutual trust and transparency**.
- Treats incidents as **shared learning opportunities**, not blame moments.
- Builds **confidence in continuity**, enabling faster cycles and innovation.

Summary

- The **Release Stage** is the bridge between delivery and real-world value.
- Its five dimensions — Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation — ensure releases are reliable, visible, and reversible.
- Mature release management replaces heroics with automation, chaos with cadence, and anxiety with confidence.
- Successful releases make trust tangible — every deployment reinforces partnership.

SDLC Stage Dimensions – Run & Evolve

Purpose

The **Run & Evolve Stage** closes the delivery loop by ensuring the system operates reliably in production and continues to improve over time.
It combines two complementary mindsets: **Run** — maintaining stability, performance, and user satisfaction; and **Evolve** — learning from outcomes and feeding insights back into future discovery and shaping.

Where Build delivers features, **Run & Evolve delivers continuity and growth**.
It's the stage where operations, feedback, and innovation merge into a sustainable improvement cycle.
When mature, this stage transforms delivery into a **living system** — capable of adapting, learning, and evolving together with users and business needs.

Core Outcomes

Outcome	Description
Operational Stability	Systems perform reliably under expected loads with minimal incidents.
Continuous Monitoring & Feedback	Real-time insights collected from usage, performance, and user satisfaction.
Incident Response & Learning	Clear processes for issue detection, triage, and recovery with lessons captured.
Value Realization Tracking	Measurement of business outcomes achieved post-release.
Continuous Improvement	Product and process evolve based on validated data and feedback.

Run & Evolve Dimensions

1. Strategic Alignment

Ensures that operational decisions and improvement priorities remain tied to strategic goals and user value.

Aspect	Low Maturity (Reactive)	High Maturity (Proactive)
Operational Goals	Focused only on uptime and SLA.	Balance between stability, performance, and outcome value.
Feedback Integration	Post-release feedback ignored or delayed.	Feedback regularly reviewed and prioritized into roadmap.
Improvement Ownership	No clear accountability for evolution.	Client and Vendor share ownership of continuous improvement.
Business Impact Visibility	Success not measured post-release.	Measurable KPIs tracked and reported in governance cadence.

Improvement Strategies

- Link *Run metrics* (uptime, error rate) to *business KPIs* (conversion, satisfaction).
- Use *Outcome Review Sessions* with Client and Vendor quarterly.
- Define *Continuous Improvement Objectives (CIOs)* tied to roadmap updates.

2. Planning & Flow

Defines how operations and evolution work are balanced, prioritized, and executed without disrupting flow.

Aspect	Low Maturity	High Maturity
Workload Balance	Operations dominate; no time for improvements.	Planned capacity split between Run (stability) and Evolve (improvement).
Incident Management	Reactive firefighting.	Proactive prevention through trend monitoring and capacity planning.
Change Planning	Uncoordinated or emergency patches.	Continuous release cycles with clear change windows.
Improvement Flow	Ideas logged but never executed.	Continuous improvement backlog integrated into delivery system.

Improvement Strategies

- Apply *SRE-inspired capacity planning* (e.g., 70/20/10 rule: 70% operations, 20% improvements, 10% innovation).
- Automate *Change Request* workflows integrated with CI/CD.
- Maintain a *Continuous Improvement Board* linking issues to systemic actions.

3. Collaboration & Communication

Defines how support, development, and business teams communicate and learn from production realities.

Aspect	Low Maturity	High Maturity
Ops-Dev Relationship	“Throw over the wall” mentality.	Shared accountability for uptime, quality, and performance.
User Communication	Reactive support tickets only.	Active feedback collection and user engagement loops.
Transparency	Incidents handled in isolation.	Public incident reporting, shared dashboards, and post-mortems.
Stakeholder Visibility	Only technical teams see system health.	Business and product stakeholders informed via clear metrics.

Improvement Strategies

- Create *Joint Operations Reviews* including engineering and client representatives.
- Implement *ChatOps* for transparency of operations and incidents.
- Publish *Service Health Dashboards* with uptime, incidents, and satisfaction metrics.

4. Quality & Risk Management

Defines how reliability, security, and technical debt are managed over time to ensure resilience and sustainability.

Aspect	Low Maturity	High Maturity
Monitoring & Alerting	Reactive, basic logs only.	Proactive, automated observability with alert thresholds.
Security & Compliance	Checked only after incidents.	Continuous scanning and compliance integrated into pipelines.
Technical Debt	Ignored until it causes issues.	Managed systematically via backlog and roadmap.
Resilience Testing	None or infrequent.	Regular chaos or load testing validating fault tolerance.

Improvement Strategies

- Implement *Observability Stack* (metrics, tracing, logs).
- Schedule *Resilience Tests* quarterly using chaos engineering tools.
- Track *Operational Risk Index* across environments.
- Automate *Security & Compliance Audits* in pipelines.

5. Learning & Adaptation

Defines how operational data and user insights drive iterative improvement across product and process.

Aspect	Low Maturity	High Maturity
Post-Incident Learning	Focus on blame or immediate fixes.	Root cause analysis and systemic improvements logged.
Continuous Improvement Culture	Improvements optional or reactive.	Teams actively suggest and experiment with new ideas.
Data Utilization	Metrics collected but not analyzed.	Metrics reviewed in governance and retrospectives.
Knowledge Continuity	Lessons forgotten between releases.	Knowledge bases maintained, reused in discovery and shaping.

Improvement Strategies

- Conduct *Blameless Post-Mortems* for all incidents.
- Maintain a *Learning Backlog* shared across teams.
- Include *Run & Evolve metrics* in quarterly business reviews.
- Apply *Kaizen* principles for incremental improvement cycles.

Common Failure Modes

Failure Mode	Root Cause	Correction
“We’re always firefighting.”	Lack of preventive monitoring and improvement time.	Dedicate capacity to continuous improvement and automation.
“Users report problems before we notice them.”	Missing observability or alerting.	Implement real-time monitoring and alerting.
“We never find time for evolution.”	No improvement planning or prioritization.	Treat evolution work as roadmap items with time allocation.
“Incidents keep repeating.”	No root cause learning or follow-up.	Introduce structured post-incident learning process.

Measuring Run & Evolve Health

Signal	Description
Uptime and performance metrics stable over time.	Operational discipline and reliability proven.
Reduced incident recurrence.	Effective root cause learning and system improvement.
Regular improvement items delivered each cycle.	Evolution embedded in delivery flow.
Business KPIs tracked post-release.	Outcomes continuously validated.

Quantitative indicators may include:

- Mean Time Between Failures (MTBF).
- Mean Time to Detect (MTTD) and Mean Time to Recover (MTTR).
- % of automated monitoring coverage.
- Number of improvement stories delivered per quarter.
- Customer satisfaction or NPS trend after releases.

Run & Evolve and Relationship Maturity

Run & Evolve strengthens **Strategic Partnership** by proving reliability, transparency, and shared learning over time.
While early stages of SDLC build trust through delivery success, Run & Evolve **sustains** it through dependability and adaptability.

High-maturity operation means:

- Issues are managed collaboratively, not defensively.
- Improvement is a shared habit, not an exception.
- Trust is renewed with every resolved incident and every measurable improvement.

Summary

- The **Run & Evolve Stage** sustains value delivery beyond release — ensuring reliability and driving continuous improvement.
- Its five dimensions — Strategic Alignment, Planning & Flow, Collaboration, Quality & Risk, Learning & Adaptation — close the feedback loop of the SDLC.
- Mature operations turn incident data into insight, insight into innovation, and innovation into renewed value.
- When Run & Evolve thrive, delivery transforms from project management into an adaptive system of partnership and progress.

Contextual Drivers Layer (CDL)

Purpose

The **Contextual Drivers Layer (CDL)** defines the external and internal forces that shape how the SDLC system should operate in a given environment. It recognizes that **no project exists in a vacuum** — every delivery model is influenced by commercial agreements, organizational structures, product maturity, technical landscape, and relationship context.

By identifying these drivers early, teams can **adapt the SDLC configuration consciously**, not reactively — selecting practices, rules, and decision patterns that best fit the project's constraints and opportunities.

CDL is therefore the **adaptive intelligence layer** of the 3SF framework: it provides the situational awareness necessary to design, diagnose, and evolve any project context.

CDL in the 3SF Structure

Layer	Purpose
SDLC Stages	Define the value flow (Discover → Evolve).
SDLC Practices	Ensure quality, automation, and governance throughout.
SDLC Stage Dimensions	Describe how each stage behaves in context.
Contextual Drivers Layer (CDL)	Explain *why* the system must behave that way — the conditions shaping delivery design.
Stable Rules Layer (SRL)	Define universal rules that keep the system coherent.
Rule Audit Checklist (RAC)	Measure adherence to those rules.
Contextual Rules Catalog (CRC)	Translate CDL + SRL into specific delivery playbooks.

CDL thus acts as the **input layer** for delivery system configuration and as the **diagnostic layer** for understanding delivery friction.

CDL Categories

The CDL organizes all contextual drivers into **six categories**, each representing a domain of influence.
These drivers can be viewed as *boundary conditions* or *design constraints* that determine how maturity, practices, and decisions should be applied.

Category	Definition	Examples of Drivers
1. Commercial Context	Financial and contractual models defining risk, ownership, and incentives.	Fixed-Bid vs. Time & Material, shared risk, cost of delay, billing structure, vendor competition.
2. Organizational Context	Structure, processes, and decision hierarchy of both Client and Vendor organizations.	Centralized vs. distributed governance, number of stakeholders, matrix structures, delivery dependencies.
3. Product Context	Nature, maturity, and complexity of the product or service being delivered.	New build vs. modernization, MVP vs. enterprise system, technical debt level, release cadence expectations.
4. Technical Context	Technology stack, platform constraints, and delivery environment.	Cloud migration vs. on-prem, CI/CD maturity, legacy integrations, DevOps readiness.
5. Relationship Context	Trust, collaboration, and alignment level between Client and Vendor.	Transactional vs. strategic partnership, communication openness, feedback culture, autonomy.
6. Environmental Context	External or systemic forces beyond direct control.	Regulatory requirements, data privacy laws, time zone overlap, market dynamics, cultural or language barriers.

Each project operates within a unique combination of these drivers.
Understanding them early allows the SDLC configuration (stages, practices, and rules) to adapt for stability and effectiveness.

CDL as a Diagnostic Lens

Contextual Drivers can be analyzed from two perspectives:

1. **Influence:** How a driver *shapes or constrains* SDLC decisions.
   Example: A *Fixed-Bid* contract limits flexibility in Discovery and Shape — requiring stronger scope definition and risk buffers.
2. **Signal:** How a driver *manifests symptoms* of stress or misalignment.
   Example: Persistent *scope creep* under a *Fixed-Bid* contract indicates mismatch between commercial and delivery context.

By mapping drivers to affected SDLC Stages and Practices, teams can uncover **root causes** of delivery dysfunction rather than symptoms.

CDL → SDLC Connection Map

CDL Category	Primary SDLC Stages Impacted	Key Practices Affected	Typical Constraints or Levers
Commercial Context	Discover, Shape, Validate	Governance & Risk, Product Thinking	Budget model, risk allocation, scope flexibility.
Organizational Context	Shape, Build, Release	Collaboration, Architecture & Design	Decision speed, approvals, dependency management.
Product Context	Discover, Build, Run	Product Thinking, Engineering & Quality	Product maturity, architecture flexibility, tech debt.
Technical Context	Build, Validate, Release	DevOps & Delivery, Architecture & Design	Automation readiness, tooling, environment stability.
Relationship Context	Discover, Shape, Evolve	Feedback & Learning, Governance & Risk	Trust, autonomy, alignment maturity, transparency.
Environmental Context	Release, Run, Evolve	Governance & Risk, Feedback & Learning	Regulatory controls, operational risk, localization.

This mapping becomes the foundation for contextual rule generation (CRC) and rule validation (RAC).

CDL Assessment

A **Contextual Driver Assessment** can be done during Discovery or periodically during long-running projects.
Its purpose is to identify **which drivers are stable, volatile, or misaligned**.

Example checklist (simplified):

Category	Driver	Status	Impact	Action
Commercial	Fixed-bid contract	Active	High constraint on flexibility	Define strict Shape governance and risk buffers.
Relationship	Low client trust	High risk	Blocks autonomy	Introduce transparency rituals and co-created planning.
Technical	Legacy system dependency	Persistent	Medium	Schedule feasibility spikes and rollback plans.

Drivers with high impact should trigger configuration adjustments (via SRL or CRC).

CDL in Practice

1. During Discovery:

- Identify contextual drivers and classify by category.
- Highlight those that constrain design or delivery models.
- Inform Shape decisions (architecture, governance, estimation).

2. During Delivery:

- Monitor changes in contextual drivers (e.g., contract renewal, new stakeholders, tool migrations).
- Adjust Stage Dimensions and Practices to maintain coherence.

3. During Retrospectives or Audits:

- Review contextual shifts to explain performance or relationship changes.
- Update RAC and CRC mappings to reflect new constraints or opportunities.

CDL and Relationship Maturity

Contextual Drivers determine how much maturity a project can realistically achieve.
For example:

- A highly regulated environment may *limit autonomy* but *increase clarity* — maturity can grow in governance, not speed.
- A flexible, trusted relationship may *allow experimentation* — maturity can grow in feedback and learning loops.

Mature organizations don't fight constraints; they **design within them**.
The CDL enables this by making constraints explicit and manageable.

Principles Failure: Contextual Contradictions

At times, contextual forces directly conflict with system principles — for example, a *regulatory environment* that enforces heavy control contradicts the principle of *Trust before Control*, or a *fixed-bid contract* that prioritizes scope over learning undermines *Outcome before Output*.
3SF treats these contradictions not as violations but as **trade-off boundaries**: situations where local compliance may temporarily override ideal behavior.

When contradictions persist unacknowledged, they lead to silent erosion of trust — teams perform rituals without intent, and principles turn into slogans.
The resolution is transparency: make constraints visible, discuss which principles are constrained, and consciously rebalance the system once conditions change.
Context cannot always be changed — but how consciously we adapt within it defines maturity.

Summary

- The **Contextual Drivers Layer (CDL)** captures the forces that shape delivery — commercial, organizational, product, technical, relationship, and environmental.
- It acts as the **input lens** for tuning the SDLC system and explaining why projects behave as they do.
- CDL connects context (why) with system design (how) — forming the foundation for adaptive governance and contextual playbooks (CRC).
- Mastering CDL awareness allows leaders to make informed trade-offs and build delivery systems that thrive under real-world constraints.

Stable Rules Layer (SRL)

Purpose

The **Stable Rules Layer (SRL)** defines the set of **universal principles and systemic constraints** that keep the SDLC delivery system coherent, regardless of context.
While the **Contextual Drivers Layer (CDL)** explains *why* a project behaves a certain way, SRL defines *how it must behave* to remain stable, predictable, and trustworthy.

Stable Rules are **non-negotiable system properties** — guardrails that ensure value flow, quality, and learning even when conditions change.
They act as the *delivery system's immune system* — maintaining balance between autonomy, governance, and feedback.

SRL in the 3SF Structure

Layer	Purpose
CDL	Identifies external and internal forces shaping the SDLC configuration.
SRL	Defines the universal laws that maintain system balance under those forces.
RAC	Audits adherence to SRL rules in specific projects.
CRC	Translates SRL rules into actionable configurations under contextual conditions.

Together, CDL and SRL form the **adaptive core** of the 3SF:

- CDL provides *situational awareness* (context).
- SRL ensures *systemic integrity* (principles).

Rule Design Philosophy

Every Stable Rule represents a **cause-and-effect relationship** critical to delivery stability.
A violation of a rule may not cause immediate failure, but it will always degrade system coherence or trust over time.

Each rule follows the pattern:

> **If** the system condition changes (due to context, scale, or relationship),
> **then** this principle must remain true,
> **otherwise** the delivery system will lose alignment, predictability, or learning capability.

Rules can therefore be applied to **design, audit, or improve** delivery — providing both *governance clarity* and *diagnostic power*.

The Twelve Stable Rules of 3SF

Rule	Principle	Purpose / Effect
R1 – Clarity before Commitment	Work must not start until purpose, outcomes, and assumptions are understood and validated.	Prevents waste and misalignment; anchors flow in value, not motion.
R2 – One System of Truth	Information about scope, status, and risks must be transparent and shared between Client and Vendor.	Avoids dual realities; enables trust and decision coherence.
R3 – Decisions are Transparent and Reversible	All key decisions must be visible, logged, and reversible based on new evidence.	Reduces risk of bias, promotes learning, and supports adaptive governance.
R4 – Progress is Measured by Outcomes, not Output	Delivery performance must be assessed through validated results, not activity volume.	Keeps teams focused on value rather than busyness.
R5 – Quality is Built In, not Inspected In	Testing, validation, and feedback loops must be integrated from the start.	Ensures reliability, prevents late surprises, and enables continuous delivery.
R6 – Flow Requires Limits	Work in Progress (WIP), context switching, and dependencies must be actively managed.	Stabilizes throughput, reduces friction, and improves predictability.
R7 – Risk Shared is Risk Reduced	Risks and uncertainties must be visible, jointly assessed, and owned across Client and Vendor.	Promotes psychological safety, fairness, and faster mitigation.
R8 – Change is a Continuous Signal, not an Exception	Change should trigger adaptation, not escalation.	Builds resilience and prevents process rigidity.
R9 – Feedback Completes the Flow	Each stage must include measurable feedback to inform the next cycle.	Ensures continuous learning and value reinforcement.
R10 – Governance Enables, not Controls	Governance should clarify ownership and empower action, not slow it down.	Balances autonomy and alignment; accelerates decision velocity.
R11 – Transparency Scales Trust	The more visible the system, the faster it learns and the deeper the trust.	Drives partnership maturity and system self-correction.
R12 – Learning is the Only Sustainable Advantage	Every failure or success must result in actionable learning and process evolution.	Institutionalizes adaptability and long-term growth.

The Why Behind the Rules

Each Stable Rule exists to prevent a specific type of systemic friction.
Understanding the “why” behind each rule helps leaders detect early warning signs and coach teams before dysfunction escalates.

Rule	Designed to Prevent
R1 – Clarity before Commitment	Prevents strategic drift and scope confusion.
R2 – One System of Truth	Prevents information asymmetry and parallel realities.
R3 – Decisions are Transparent and Reversible	Prevents bias and irreversible errors under uncertainty.
R4 – Progress is Measured by Outcomes, not Output	Prevents false productivity and local optimization.
R5 – Quality is Built In, not Inspected In	Prevents late discovery of defects and technical debt accumulation.
R6 – Flow Requires Limits	Prevents overcommitment, multitasking, and delivery chaos.
R7 – Risk Shared is Risk Reduced	Prevents blame culture and hidden escalation loops.
R8 – Change is a Continuous Signal, not an Exception	Prevents resistance to adaptation and process rigidity.
R9 – Feedback Completes the Flow	Prevents learning delays and repetitive mistakes.
R10 – Governance Enables, not Controls	Prevents bureaucracy and decision paralysis.
R11 – Transparency Scales Trust	Prevents mistrust and hidden dependencies.
R12 – Learning is the Only Sustainable Advantage	Prevents stagnation and organizational amnesia.

Rule Relationships and Hierarchy

The twelve rules form **three meta-groups**, reflecting how 3SF integrates *structure, flow, and learning*:

Meta-Group	Included Rules	Purpose
Alignment Rules	R1, R2, R3, R4	Keep system purpose clear and decisions coherent.
Flow Rules	R5, R6, R7, R8	Maintain stable, adaptive value flow under changing conditions.
Learning Rules	R9, R10, R11, R12	Ensure system feedback, trust, and improvement scale sustainably.

This grouping enables easier auditing and rule mapping to maturity models and practices.

When Rules Collide

In complex delivery environments, Stable Rules can appear to conflict — for example, **R6 (Flow Requires Limits)** may constrain **R8 (Change as a Continuous Signal)** when too many concurrent adaptations overload the system.
Such collisions are not failures but **tensions to be balanced**.

When conflicts arise:

- Return to **Alignment Rules (R1–R4)** — they define purpose and coherence.
- Reassess which rule serves system stability under current context.
- Revisit decisions once conditions change — rule prioritization is dynamic, not static.

SRL → SDLC Mapping

Rule	Primary SDLC Stages Impacted	Key SDLC Practices Affected
R1 Clarity before Commitment	Discover, Shape	Product Thinking, Governance & Risk
R2 One System of Truth	Shape, Build, Validate	Collaboration, DevOps & Delivery
R3 Decisions are Transparent and Reversible	Shape, Build	Governance & Risk, Feedback & Learning
R4 Progress is Measured by Outcomes, not Output	Build, Validate, Evolve	Product Thinking, Feedback & Learning
R5 Quality is Built In, not Inspected In	Build, Validate	Engineering & Quality, DevOps & Delivery
R6 Flow Requires Limits	Build, Validate, Release	DevOps & Delivery, Governance & Risk
R7 Risk Shared is Risk Reduced	Discover, Shape, Release	Governance & Risk, Collaboration
R8 Change is a Continuous Signal, not an Exception	Build, Evolve	Feedback & Learning, DevOps & Delivery
R9 Feedback Completes the Flow	Validate, Evolve	Feedback & Learning
R10 Governance Enables, not Controls	Shape, Release, Run	Governance & Risk, Collaboration
R11 Transparency Scales Trust	All stages	All practices
R12 Learning is the Only Sustainable Advantage	Run, Evolve	Feedback & Learning, Governance & Risk

This mapping provides traceability from abstract system rules to operational levers — ensuring that every rule has observable effects in delivery behavior.

Violations and Signals

Each rule violation manifests as a **systemic signal** — a pattern of dysfunction visible through metrics or team behavior.

Rule Violation	Typical Signals	Possible Root Cause
R1	Teams start work without validated goals.	Missing or ineffective Discovery.
R2	Client and Vendor track different statuses.	Poor transparency, separate tooling.
R5	QA overloaded near release.	Lack of automation and shift-left testing.
R6	Developers multitasking across projects.	No WIP limits, poor prioritization.
R9	Feedback delayed or ignored.	Weak retrospectives, missing monitoring.
R11	Mistrust between Client and Vendor.	Hidden data, lack of open dashboards.

Recognizing such signals early allows intervention before performance or trust degrade.

Using the Stable Rules Layer

As a design tool:

- Apply SRL when configuring delivery systems to ensure systemic balance regardless of project context.
- Use rules as *design constraints* when defining governance, practices, or automation.

As an audit tool:

- Use SRL rules as checkpoints in the **Rule Audit Checklist (RAC)**.
- Each RAC question should trace back to one or more SRL rules.

As a coaching tool:

- Use rules to guide leadership conversations around responsibility, autonomy, and improvement focus.
- When teams ask *"what should we fix first?"*, use SRL to identify foundational gaps.

SRL and Maturity

Stable Rules evolve from compliance to instinct:

Maturity Level	Rule Behavior
Reactive (Level 1)	Rules followed inconsistently; dependent on individuals.
Structured (Level 2)	Rules known and referenced but not yet internalized.
Integrated (Level 3)	Rules embedded in processes and automation.
Adaptive (Level 4)	Rules lived instinctively; used for self-correction and innovation.

The goal is not to enforce rules mechanically, but to embed them in system design and organizational culture so that *stability emerges naturally*.

Summary

- The **Stable Rules Layer (SRL)** defines universal laws that preserve system coherence and delivery trust under any context.
- The **12 Stable Rules** form the backbone of predictable, learning-oriented delivery.
- SRL ensures that flexibility never compromises integrity — enabling adaptation without chaos.
- When applied consistently, SRL transforms best practices into **governing principles of maturity**, ensuring every project remains stable, transparent, and continuously improving.

Rule Audit Checklist (RAC)

Purpose

The **Rule Audit Checklist (RAC)** provides a structured way to evaluate how well a project or delivery system adheres to the **Stable Rules Layer (SRL)**.
It serves as a **diagnostic and learning instrument**, helping Project Leads, Product Managers, Architects, and Delivery Directors identify where the SDLC system is coherent — and where it may be drifting.

RAC turns the abstract SRL principles into **practical, observable checks** that can be applied across any delivery model, whether fixed-bid, time-and-material, or co-delivery.
It can be used for **self-assessment**, **peer review**, or **formal audit**, depending on project maturity and organizational needs.

RAC in the 3SF Structure

Layer	Purpose
SRL (Stable Rules Layer)	Defines the universal principles that keep delivery coherent.
RAC (Rule Audit Checklist)	Translates those principles into assessable questions.
CRC (Contextual Rules Catalog)	Suggests contextual adaptations when rules need to flex.

The RAC's job is not compliance policing — it is **delivery sense-making**.
It helps leaders interpret project behavior and choose improvement focus areas based on systemic signals.

Using the RAC

1. **Select a project or program** — ideally after one full delivery iteration (e.g., quarter or milestone).
2. **Interview or survey** participants from both Client and Vendor sides (Product Manager, Project Lead, Tech Lead, QA, Architect, Stakeholder).
3. **Score each rule** from 1–4 based on observed behavior.
4. **Discuss evidence and next actions** — RAC is most effective as a conversation, not a checklist.
5. **Summarize results** in a visual radar or heatmap (rules vs. maturity) for transparency.

Scoring Scale

Score	Meaning	Interpretation
1 – Reactive	Rule often violated or ignored.	Delivery unstable, reactive management.
2 – Defined	Rule recognized but inconsistently applied.	Awareness exists, needs reinforcement.
3 – Embedded	Rule consistently applied and measured.	Predictable delivery, learning in motion.
4 – Adaptive	Rule instinctively followed and improved.	Self-correcting system; maturity embedded.

RAC Template

Rule ID	Rule Name	Audit Question(s)	Observed Evidence / Notes	Score (1–4)	Next Action / Owner
R1	Clarity before Commitment	Do all team members understand project goals, value, and success criteria before work begins?			
R2	One System of Truth	Is there a single shared source of information (scope, status, risks) between Client and Vendor?			
R3	Decisions are Transparent and Reversible	Are key decisions visible, documented, and revisited when assumptions change?			
R4	Progress is Measured by Outcomes, not Output	Are outcomes and KPIs used to measure progress instead of just deliverables?			
R5	Quality is Built In, not Inspected In	Is testing and validation integrated into development rather than deferred?			
R6	Flow Requires Limits	Are WIP, dependencies, and task switching actively managed to maintain throughput?			
R7	Risk Shared is Risk Reduced	Are risks logged, owned jointly, and addressed collaboratively?			
R8	Change is a Continuous Signal, not an Exception	Are scope or plan changes processed through adaptive governance, not escalation?			
R9					

Rule ID	Rule Name	Audit Question(s)	Observed Evidence / Notes	Score (1–4)	Next Action / Owner
	Feedback Completes the Flow	Are feedback loops between users, systems, and teams continuous and acted upon?			
R10	Governance Enables, not Controls	Does governance accelerate decision-making and empower teams rather than slow them down?			
R11	Transparency Scales Trust	Are delivery data, metrics, and plans visible to all stakeholders?			
R12	Learning is the Only Sustainable Advantage	Are lessons learned documented and transformed into actions or system changes?			

(Recommended: keep this table as a shared document or dashboard updated quarterly.)

Interpreting RAC Results

1. Identify Systemic Weaknesses

- Low scores across several rules in the same meta-group (e.g., Flow or Learning) indicate systemic imbalance.
- For example, strong **Alignment** but weak **Flow** may suggest bureaucracy or slow adaptation.

2. Analyze Cross-Role Perspectives

- Compare Client vs. Vendor perceptions — maturity gaps often reveal trust or communication issues.
- Different scores for the same rule are **valuable**, not errors — they highlight where transparency or expectations differ.

3. Track Maturity Over Time

- Conduct RAC every 3–6 months.
- Visualize progress using radar charts grouped by Alignment / Flow / Learning rule families.
- Celebrate upward trends; investigate stagnation or regression.

4. Link RAC to Improvement Planning

- Each low-scoring rule becomes an **improvement opportunity**.
- Select 2–3 rules per cycle for focused improvement, linking them to measurable actions in the delivery plan.

RAC Example Summary Report

Meta-Group	Rules	Average Score	Interpretation	Suggested Focus
Alignment	R1–R4	3.0	Clear purpose and measurable outcomes.	Maintain stakeholder alignment.
Flow	R5–R8	2.3	Quality and risk processes under stress.	Strengthen automation and WIP control.
Learning	R9–R12	1.8	Feedback loops and improvement culture weak.	Prioritize retrospectives and shared metrics.

RAC Application in Practice

Use the RAC to:

- Facilitate *quarterly health checks* for ongoing projects.
- Support *retrospectives* at delivery milestones.
- Align *Client–Vendor steering discussions* around facts, not perceptions.
- Provide *input to the Contextual Rules Catalog (CRC)* for tailored playbooks.
- Serve as *baseline measurement* before initiating major process or relationship changes.

RAC Facilitation Tips

- Conduct as a **facilitated workshop**, not a survey — dialogue matters more than scores.
- Keep each rule discussion time-boxed (~10–15 minutes).
- Encourage evidence sharing (artifacts, dashboards, meeting notes).
- End with **3 actionable insights per meta-group** (Alignment, Flow, Learning).
- Visualize results — transparency itself reinforces trust (R11).

RAC and Relationship Maturity

Rule adherence mirrors relationship maturity:

- **Transactional (1–2):** Rules applied inconsistently; focus on compliance and reporting.
- **Aligned Autonomy (2–3):** Rules accepted and measured; governance collaborative.
- **Strategic Partnership (3–4):** Rules instinctive; system self-corrects via feedback and trust.

Thus, the RAC not only measures system health — it measures **partnership evolution**.

Systemic Failure: The Maturity Mirage

A common risk in mature organizations is the **illusion of progress** — performing RAC reviews, retrospectives, and rule discussions as rituals while the underlying mindset remains unchanged.
This *maturity mirage* creates dashboards that look healthy but conceal disengagement, blame avoidance, or shallow trust.
True maturity is evidenced by visible change in behavior and decision quality, not by consistent high scores.
When results stay static despite visible activity, it signals **measurement without learning** — the system has optimized for appearance, not evolution.

Summary

- The **Rule Audit Checklist (RAC)** operationalizes the 12 Stable Rules into measurable behaviors.

• It enables reflective, evidence-based dialogue about delivery health and maturity.

• RAC results guide improvement focus, governance calibration, and contextual rule selection (CRC).

• Used regularly, RAC turns principles into practice — ensuring the delivery system remains *transparent, adaptive, and trusted*.

Contextual Rules Catalog (CRC)

Purpose

The **Contextual Rules Catalog (CRC)** translates the universal **Stable Rules (SRL)** into **context-specific playbooks**, enabling adaptive delivery under real-world conditions.
Where SRL defines *what must always remain true*, the CRC defines *how it can best be achieved* given the unique **Contextual Drivers (CDL)** of each project.

CRC is therefore the **pragmatic layer** of 3SF — connecting principles to practice, helping leaders adjust their delivery approach without compromising coherence or trust.
It is used when system rules need to flex within stable boundaries — for example, when working under fixed-bid contracts, co-delivery models, or continuous product operations.

CRC in the 3SF Structure

Layer	Purpose
CDL (Contextual Drivers)	Describe external/internal forces shaping delivery.
SRL (Stable Rules)	Define non-negotiable systemic principles.
RAC (Rule Audit Checklist)	Measure adherence to rules in practice.
CRC (Contextual Rules Catalog)	Adapt and operationalize rules for specific delivery contexts.

CRC allows organizations to move beyond “one-size-fits-all” process design — by **anchoring adaptability in principle, not preference**.

CRC Design Philosophy

Every contextual rule is expressed as a **conditional mapping**:

> **If** (contextual driver X) and **if** (affected SDLC stage/dimension Y)
> **then** (apply strategy or configuration Z to maintain stability and maturity).

These mappings ensure that project-level adaptations stay consistent with system integrity —
never violating SRL, only expressing it differently.

CRC Categories (Delivery Context Archetypes)

The CRC defines a small number of **delivery context archetypes** that represent typical real-world configurations of Client–Vendor collaboration. Each archetype modifies how the 12 Stable Rules are implemented across the SDLC stages.

Archetype	Description	Typical CDL Signature
A1. Fixed-Bid / Contractual Delivery	Defined scope, fixed timeline and cost; low tolerance for change.	High commercial constraint, low flexibility, often transactional trust.
A2. Co-Delivery Partnership	Shared ownership between client and vendor; teams integrated across functions.	Balanced commercial model, aligned autonomy, medium-high trust.
A3. Continuous Product Evolution	Ongoing development and optimization of a live product.	Time & material, strong feedback loops, strategic partnership.
A4. Rapid Discovery / Innovation Pilot	Short, exploratory engagement with uncertain outcomes.	High uncertainty, high flexibility, fast iteration, trust critical.
A5. Managed Service / Sustainment	Vendor responsible for operations, monitoring, and continuous improvement.	Operational stability, predictable cadence, long-term accountability.

Each archetype inherits all 12 Stable Rules — but expresses them differently depending on constraints and priorities.

CRC Example Mappings

A1. Fixed-Bid / Contractual Delivery

Rule (SRL)	Implementation in Fixed-Bid Context	Reason / Effect
R1 – Clarity before Commitment	Discovery and Shape must be separate contracts or phases with exit criteria.	Ensures estimates and commitments are evidence-based.
R3 – Decisions are Transparent and Reversible	Changes logged formally via controlled governance (CRs), with impact tracking.	Reduces escalation risk while maintaining traceability.
R5 – Quality is Built In	Enforce automated regression testing from sprint one; no manual-only QA.	Compensates for limited change tolerance.
R7 – Risk Shared is Risk Reduced	Define shared risk buffer in the SOW for rework or unforeseen effort.	Protects both parties under rigid commercial terms.
R10 – Governance Enables, not Controls	Steering cadence fixed but focused on evidence-based progress, not reporting.	Keeps governance lean and trust-building.

Outcome: Predictability and accountability strengthened without over-bureaucratization.

A2. Co-Delivery Partnership

Rule (SRL)	Implementation in Co-Delivery Context	Reason / Effect
R2 – One System of Truth	Shared Jira/ADO boards and dashboards visible to both Client and Vendor.	Eliminates dual status versions.
R4 – Progress is Measured by Outcomes	Use shared OKRs tied to joint product roadmap milestones.	Aligns performance metrics across organizations.
R6 – Flow Requires Limits	Manage WIP jointly per epic, not per team.	Prevents overload in shared delivery responsibility.
R8 – Change is a Continuous Signal	Adjust backlog continuously based on data and stakeholder feedback.	Allows agility without renegotiation overhead.
R11 – Transparency Scales Trust	Open access to repositories, docs, and incident logs.	Reinforces partnership maturity through openness.

Outcome: Collaboration operates as one delivery organism with shared visibility and accountability.

A3. Continuous Product Evolution

Rule (SRL)	Implementation in Continuous Product Context	Reason / Effect
R1 – Clarity before Commitment	Use rolling quarterly goals and flexible prioritization.	Keeps intent clear while allowing iteration.
R5 – Quality is Built In	CI/CD pipelines enforce automated testing and code quality gates.	Maintains delivery speed without degrading stability.
R9 – Feedback Completes the Flow	Integrate analytics and user feedback loops directly into backlog grooming.	Ensures real-world insights drive development.
R10 – Governance Enables, not Controls	Product councils decide direction based on evidence, not hierarchy.	Keeps governance adaptive and data-driven.
R12 – Learning is the Only Sustainable Advantage	Post-release reviews treated as mandatory; improvements prioritized in next sprint.	Ensures continuous evolution and innovation.

Outcome: Stable yet fast-moving system where learning drives value growth.

A4. Rapid Discovery / Innovation Pilot

Rule (SRL)	Implementation in Innovation Context	Reason / Effect
R1 – Clarity before Commitment	Define problem statement and validation plan — not fixed deliverables.	Focuses effort on learning, not output.
R4 – Progress is Measured by Outcomes	Use hypothesis validation metrics (e.g., # validated assumptions).	Measures learning speed and quality.
R6 – Flow Requires Limits	Limit concurrent hypotheses to maintain focus.	Preserves discovery quality and prevents diffusion.
R7 – Risk Shared is Risk Reduced	Agree on fast feedback loops and shared decision checkpoints.	Enables safe experimentation within timebox.
R9 – Feedback Completes the Flow	Include early user feedback sessions before prototype completion.	Validates direction before full investment.

Outcome: Maximized learning per time unit, reduced risk of building the wrong thing.

A5. Managed Service / Sustainment

Rule (SRL)	Implementation in Managed Service Context	Reason / Effect
R2 – One System of Truth	Operational dashboards shared across vendor and client.	Promotes shared situational awareness.
R5 – Quality is Built In	Automate monitoring, alerting, and compliance checks.	Ensures stability and efficiency.
R7 – Risk Shared is Risk Reduced	Shared SLA ownership and incident root-cause analysis.	Fosters continuous reliability improvements.
R9 – Feedback Completes the Flow	Weekly review of incidents and performance trends.	Turns operations data into learning.
R12 – Learning is the Only Sustainable Advantage	Continuous improvement backlog integrated with run operations.	Evolves service quality and relationship maturity.

Outcome: Predictable, transparent operations aligned with continuous system evolution.

CRC Mapping Template

To create new contextual playbooks, use this mapping format:

If CDL Condition...	and SDLC Stage/ Dimension is...	then apply Practice(s)...	to uphold SRL Rule...	Effect / Outcome
Fixed-bid contract limits flexibility	Shape – Planning & Flow	Progressive estimation, explicit risk buffer	R1, R7	Predictable scope and shared risk mitigation
Low relationship trust	Discover – Collaboration	Transparent workshop cadence and visible notes	R2, R11	Builds early transparency and trust
Highly regulated industry	Release – Governance	Automated compliance gates and audit trails	R5, R10	Ensures quality and control within compliance
Continuous product delivery	Evolve – Learning & Adaptation	Ongoing telemetry and user research loops	R9, R12	Maintains improvement momentum

This table can be extended for organization-specific contexts and stored in a shared CRC knowledge base (e.g., Confluence, GitHub Wiki, or internal portal).

CRC and Maturity

As maturity increases, projects move from **rule application** to **rule design**:

Maturity Level	CRC Usage Behavior
Level 1 – Reactive	Teams rely on predefined CRC playbooks (Fixed-Bid, Co-Delivery).
Level 2 – Structured	Teams select and combine playbook elements consciously.
Level 3 – Adaptive	Teams co-create custom mappings based on real feedback.
Level 4 – Evolutionary	Organization continuously evolves CRC through lived experience and cross-project learning.

CRC Governance

To keep the catalog effective:

- **Review quarterly** to capture new lessons from projects.
- **Version control** CRC playbooks (v1.1, v1.2…) to track learning.
- **Link** CRC rules to RAC findings — improvement actions should modify or extend contextual mappings.
- **Integrate** CRC into onboarding and project kickoff processes.

Adapting Without Excusing

The purpose of contextual adaptation is **to preserve system integrity under constraint**, not to justify underperformance.
A common misuse of CRC occurs when teams interpret flexibility as permission to ignore stable rules — turning context into an excuse rather than a design factor.
Healthy adaptation always traces back to **which Stable Rule it protects** and **what learning it generates**.
When a contextual rule cannot explain its stabilizing purpose, it signals adaptation drift — a warning that coherence is being traded for comfort.

CRC and Relationship Maturity

CRC operationalizes relationship maturity:

- **Transactional projects** depend on structured CRC playbooks for predictability.
- **Aligned partnerships** use CRC dynamically to adjust governance.
- **Strategic partnerships** co-create new contextual rules, contributing back to the catalog.

Thus, CRC is not just documentation — it's the **institutional memory of delivery intelligence**.

Summary

- The **Contextual Rules Catalog (CRC)** translates stable system principles (SRL) into practical guidance for diverse delivery contexts (CDL).
- It ensures flexibility without chaos — adapting practices to context while preserving systemic integrity.
- CRC acts as the living knowledge base of 3SF — continuously evolving through audit (RAC) and experience.
- When mastered, CRC turns every project into a learning loop — where context refines rules, and rules sustain excellence.

Relationship and System Maturity Integration

Purpose

This section connects the **3-in-3 Relationship Maturity Model** with the **3SF delivery system** — showing how partnership growth is both the cause and result of a mature SDLC.
It also reconciles 3SF with **product-organization dynamics**, identifying capability gaps that commonly emerge when vendor teams operate without shared product ownership or user feedback.

The goal is to present a **unified maturity model** where:

- **Relationship maturity** (trust, autonomy, partnership)
- **System maturity** (delivery, feedback, governance)
- **Product maturity** (user value, sustainment, learning)

all evolve together through intentional practice and feedback loops.

This closing layer turns 3SF from a delivery model into a **growth system** — for both teams and relationships.

The Integrated Maturity Curve

Maturity Level	Relationship Focus	System Behavior	Client–Vendor Dynamics
1. Transactional Trust	Delivery based on control, compliance, and scope.	Processes reactive, success measured by output.	Client dictates, vendor executes; little shared context.
2. Aligned Autonomy	Mutual visibility and transparency built.	Delivery predictable; feedback begins to close loops.	Vendor earns trust through reliability and openness.
3. Shared Ownership	Joint accountability for decisions and outcomes.	System self-regulating; governance collaborative.	Teams act as one delivery unit with shared metrics.
4. Strategic Partnership	Partnership extends to shared vision and innovation.	System adaptive, learning-oriented, continuously improving.	Joint investment in evolution, experimentation, and long-term value.

Each maturity level requires **specific practices and system configurations** that strengthen both technical and relational resilience.

The Hidden Cost of Transactional Trust

Operating in a transactional state carries invisible costs.
For clients, it means time spent writing detailed contracts, auditing invoices, and resolving scope disputes instead of pursuing outcomes.
For vendors, it means constant justification of estimates, reduced margins, and limited opportunity to influence strategic direction.
Both sides lose energy defending control instead of creating value — a pattern 3SF defines as *trust debt*.
The longer this persists, the harder it becomes to transition toward partnership maturity.

The Six Capability Gap Categories

When comparing vendor-oriented delivery and product-oriented organizations, six recurring capability gaps were identified.
Each gap reflects a missing bridge between **delivery maturity** and **product maturity** — and therefore, between **process** and **partnership**.

Gap Category	Definition
1. Strategic Alignment	Ensuring that delivery outcomes tie directly to user and business goals.
2. User Insight & Validation	Integrating real user understanding and validation throughout the lifecycle.
3. Outcome-Based Planning	Prioritizing and measuring progress by outcomes, not activities.
4. Feedback & Learning Loops	Establishing fast, actionable learning cycles across system and users.
5. Sustainment Readiness	Ensuring that products and relationships can scale and improve post-release.
6. Governance Clarity	Making ownership, decision rights, and accountability explicit and balanced.

These six dimensions act as **growth levers** across the maturity curve — from reactive coordination to adaptive partnership.

Capability Maturity Mapping

Gap Category	Level 1 — Transactional	Level 2 — Aligned Autonomy	Level 3 — Shared Ownership	Level 4 — Strategic Partnership
Strategic Alignment	Scope-driven work; client defines direction.	Goals visible but not jointly prioritized.	Shared product vision and measurable outcomes.	Strategic co-creation of roadmap and innovation themes.
User Insight & Validation	Users absent from decision-making.	Proxy validation via business stakeholders.	Regular user testing and data feedback integrated.	Continuous research and insight loops informing evolution.
Outcome-Based Planning	Focus on tasks and deadlines.	Partial use of KPIs or OKRs, inconsistently applied.	Outcomes and value metrics drive backlog prioritization.	Portfolio-level outcome steering aligned with business impact.
Feedback & Learning Loops	Feedback delayed until post-release.	Regular retrospectives and sprint demos.	Continuous monitoring, telemetry, and learning backlog.	Data-driven experimentation and innovation culture.
Sustainment Readiness	Release = completion.	Support plans defined post-factum.	Ongoing Run & Evolve processes with improvement metrics.	Autonomous, self-improving delivery ecosystem.
Governance Clarity	Escalation-based, hierarchical control.	Defined cadences, but ownership fragmented.	Governance enables distributed decision-making.	Shared governance with adaptive decision flow and mutual accountability.

This mapping creates a **relationship–system symmetry**:
as delivery processes mature, so does the capacity for autonomy, feedback, and innovation.

Relationship Progression Goals (Practices That Move Rightward)

Each maturity transition can be intentionally facilitated through targeted practices.
Below are examples of **key enablers** for each step in the progression.

From Level 1 → Level 2: Building Transparency and Predictability

Goal	Supporting Practices / Mechanisms
Establish common understanding of goals.	Product Thinking workshops, shared goal trees (VMOSA).
Replace command–control with clarity.	Working Agreements, Definition of Done/Ready, RACI clarity.
Create transparency of work and status.	One System of Truth (shared backlog, dashboards).
Build trust through reliability.	Regular delivery cadence, short cycles, evidence-based reporting.

Focus: predictability, openness, early validation of trust.

From Level 2 → Level 3: Building Co-Ownership

Goal	Supporting Practices / Mechanisms
Align delivery and business outcomes.	Joint backlog prioritization, outcome-based roadmaps.
Introduce shared decision forums.	Steering cadences with dual representation (Client + Vendor).
Embed continuous validation.	Feedback & Learning integration into every stage.
Evolve governance into enablement.	SRL Rule R10 — Governance Enables, Not Controls.

Focus: transition from reporting to co-creation.

From Level 3 → Level 4: Building Strategic Partnership

Goal	Supporting Practices / Mechanisms
Jointly define strategy and product vision.	Strategic alignment workshops, Wardley Mapping.
Operate through mutual autonomy.	Cross-organizational squads or Centers of Excellence.
Institutionalize learning.	Continuous retrospectives, shared improvement roadmaps.
Co-invest in evolution and innovation.	Shared success metrics (ROI, OKRs), innovation sprints.

Focus: shared accountability, long-term value, and joint evolution.

Systemic Integration Across 3SF Layers

3SF Layer	How Relationship Maturity Integrates
3-in-3 Model	Defines partnership boundaries and responsibilities — the social structure of the system.
SDLC Stages	Each stage provides a new maturity checkpoint (Discover builds trust, Build proves autonomy, Evolve sustains partnership).
SDLC Practices	Cross-cutting disciplines ensure transparency, quality, and learning — prerequisites for partnership growth.
SDLC Stage Dimensions	Provide the configuration levers (planning, collaboration, governance) that influence maturity progress.
CDL	Contextual constraints determine how fast or far maturity can grow (e.g., contract rigidity, regulatory environments).
SRL	Stable Rules maintain integrity during growth — ensuring adaptation without chaos.
RAC & CRC	Measure and guide relationship evolution through evidence and contextual adaptation.

Together, these layers form a **closed system of evolution** — ensuring that delivery maturity is inseparable from relationship maturity.

Model Failure: The Collapsed Triangle

When one of the three relationship lines weakens, the 3-in-3 system loses balance.
If **Client ↔ Product** fails (e.g., absent product ownership or unclear business vision), the **Vendor ↔ Client** relationship becomes purely transactional.
If **Vendor ↔ Product** fails (e.g., poor engineering quality or limited feedback), trust erodes despite goodwill.
Recovery begins not by enforcing new controls but by **reconnecting the missing relationship** — restoring flow between value, delivery, and engagement.

Example Progression Scenario

"From Fixed-Bid Execution to Strategic Co-Delivery Partnership"

Stage	Before (Transactional)	After (Strategic Partnership)	Key Shift
Discover	Vendor interprets requirements from RFP.	Joint discovery; shared workshops and validation.	Clarity and trust built early.
Shape	Vendor estimates scope; client approves.	Teams co-shape scope and architecture.	Co-ownership of feasibility and trade-offs.
Build	Vendor delivers to spec; client reviews.	Continuous delivery with shared metrics and transparent progress.	Trust through visibility and predictability.
Validate	QA owned by vendor; sign-off by client.	Joint validation and user acceptance.	Mutual accountability for quality.
Release	Hand-off to client operations.	Shared release management and change communication.	Transparency and confidence.
Run & Evolve	Vendor disengages post-release.	Continuous product improvement with shared goals.	Partnership renewal through learning.

Result: trust scales into autonomy, autonomy into shared ownership, and shared ownership into innovation.

Measuring Relationship Maturity in Practice

To make partnership growth tangible, measure indicators across three lenses:

Lens	Example Indicators	Tools / Data Sources
Behavioral	Response speed, feedback participation, escalation frequency.	Meeting logs, stakeholder surveys.
Systemic	Shared KPIs, backlog visibility, governance adherence.	RAC scores, dashboard audits.
Outcome	Product impact, stability, improvement velocity.	Delivery metrics, post-release performance.

These metrics form the **Relationship Maturity Dashboard**, ideally reviewed in the same cadence as delivery retrospectives.

Behavioral Failure: The Scapegoat Culture

In immature systems, shared accountability can degrade into shared blame — a culture where every issue seeks a culprit instead of a cause.
This **scapegoat culture** destroys psychological safety, making transparency impossible.
Symptoms include defensive retrospectives, risk concealment, and loss of initiative.
3SF counters this through **R7 (Risk Shared is Risk Reduced)** and **R12 (Learning is the Only Sustainable Advantage)** — turning failure into data, not division.

Summary

- **Relationship maturity is system maturity.** It grows through the same feedback loops that improve delivery and product quality.
- The **six capability gaps** (strategy, user, outcomes, feedback, sustainment, governance) act as maturity levers across all SDLC stages.
- **Progression goals** move relationships from transactional to strategic through trust, co-creation, and shared accountability.
- Integrated within the 3SF layers, maturity becomes not a byproduct — but the *primary output* of effective delivery systems.

When teams master 3SF, they don't just deliver software;
they evolve the **partnerships, systems, and organizations** that make continuous value creation possible.

3SF Practice Architecture

> *"Theory guides understanding. Practice enables transformation."*

Purpose

The **3SF Practice Architecture** explains how to **apply the 3-in-3 SDLC Framework (3SF)** in real projects — for both **Client** and **Vendor** organizations.

While the **theory** defines *why* 3SF exists and *what* it represents, the **practice** defines *how* to use it — *when*, *by whom*, and *for what purpose*.

3SF practices are designed to:

- Help **clients** mature as commissioning and ownership partners.
- Help **vendors** mature as delivery and advisory partners.
- Create a **shared language** between both sides across the full SDLC lifecycle.

Objectives

- Provide a **map of 3SF usage** across delivery stages and relationship lines.
- Distinguish **client-side** and **vendor-side** applications of each tool.
- Define **modes of application** (Design, Diagnose, Assess, Reflect, Audit).
- Act as a **navigator** for all practice tools and related training modules.

Structure of the Practice Part

Each practice is a standalone tool (one file per tool) using a unified template.
Tools are grouped by **application mode** and mapped to **client / vendor contexts**.

Mode	Primary Focus	Typical User	Example
Design	Establish delivery & engagement model	Vendor **Delivery Facilitator** / Client **Product Leader**	Initial Delivery System Design
Diagnose	Reveal constraints and gaps	Vendor **Solution Architect** / Client **Product Leader**	Delivery System Diagnostic
Assess	Measure project & relationship maturity	**Delivery Facilitator** / **Account Lead** / **Executive Sponsor**	Quarterly Project Assessment
Reflect	Enable self-awareness and growth	**Delivery Facilitator**, **Technical Integrator**, **Solution Architect**, **Product Leader**	Self-Diagnostic Tool
Audit & Aggregate	Compare and improve across portfolio	**Engineering Director** / **Governance Officer**	Relationship Audit & Portfolio Dashboard

Dual-Perspective Application

Each 3SF tool distinguishes **Client View** and **Vendor View**:

Perspective	Purpose	Typical Roles	Example Application
Client	Ensure vendor alignment, internal dependencies, and ownership clarity.	**Product Leader**, **Executive Sponsor**, **Vendor Manager**	Use 3SF tools to verify engagement readiness and integration.
Vendor	Design, execute, and evolve delivery systems that build partnership trust.	**Delivery Facilitator**, **Solution Architect**, **Product Leader**	Use 3SF tools to structure engagements and measure maturity.
Shared (Client + Vendor)	Strengthen collaboration, feedback, and transparency.	Combined teams across both sides	Apply tools jointly to synchronize relationship evolution.

Each practice file includes sections:

> **Client-Side Application** and **Vendor-Side Application**, highlighting respective actions and insights.

Application Modes

Mode	Typical When	Used by (Client / Vendor)	3SF Layers Focused	Outcomes
Design	RFP → Kick-off	**Delivery Facilitator** / **Product Leader**	CDL + SRL	Intent and collaboration system established
Diagnose	Discovery / Early Delivery	**Solution Architect** / **Product Leader**	CDL + SRL	Constraints and dependencies identified
Assess	Quarterly / Milestone	**Delivery Facilitator** / **Account Lead** / **Executive Sponsor**	SRL + RAC	Delivery and relationship health tracked
Reflect	Anytime	**Delivery Facilitator**, **Technical Integrator**, **Product Leader**	RAC	Individual or team awareness improved
Audit	Periodic / Portfolio	**Engineering Director** / **Governance Officer**	CDL + SRL + RAC	Portfolio-level maturity comparison

Lifecycle Navigation

When	Mode	Tool (File)	Client Role(s)	Vendor Role(s)	Outcome
RFP / Pre-Engagement	Design	`210-initial-delivery-design.md`	**Executive Sponsor**, **Product Leader**	**Delivery Facilitator**, **Account Lead**	Shared engagement model & maturity baseline
Small Team Discovery	Diagnose	`220-delivery-system-diagnostic.md`	**Product Leader**, **Requirements Analyst**	**Solution Architect**, **Delivery Facilitator**	Validated context and delivery readiness
Kick-off	Design	`210-initial-delivery-design.md`	**Product Leader**, **Executive Sponsor**	**Delivery Facilitator**, **Solution Architect**	Clear interfaces and governance rules
Ongoing Delivery (MVP → v1)	Assess	`230-quarterly-assessment.md`	**Product Leader**, **Executive Sponsor**	**Delivery Facilitator**, **Account Lead**	Relationship and flow monitored
Anytime	Reflect	`240-self-diagnostic.md`	**Client Product Leader**, **Governance Officer**	**Delivery Facilitator**, **Technical Integrator**	Personal or team growth insight
Post-v1 / Portfolio	Audit	`250-relationship-audit.md`	**Executive Sponsor**, **Governance Officer**	**Delivery Facilitator**, **Engineering Director**	Lessons learned & next-stage maturity targets

3SF Layer Mapping

3SF Layer	Purpose	Client-Side Focus	Vendor-Side Focus	Example Tool
Contextual Drivers Layer (CDL)	Clarify environment, goals, and constraints	Business objectives & internal readiness	Engagement framing & feasibility	Initial Delivery Design
Stable Rules Layer (SRL)	Define repeatable collaboration patterns	Governance & dependencies	Team operations & flow	Delivery System Diagnostic
Rule Audit Checklist (RAC)	Reflect and verify maturity	Value realization & partnership signals	Relationship metrics & feedback	Self-Diagnostic / Relationship Audit

3SF Functional Role Model

Purpose

The Functional Role Model defines how both Client and Vendor teams participate in the SDLC system described by 3SF.
It establishes a shared functional language across all practices, enabling consistent use of roles in the Practice documents.
Roles are defined by their **system function**, not organizational title, and may exist on either side depending on engagement setup.

Co-Dependent SDLC System Functions

These are the core **joint execution functions** that drive the SDLC. They are necessary for managing the **Client ↔ Vendor ↔ Product** relationships.

Functional Core	Purpose (Joint SDLC Management)	Composite Name Examples
Product Leader	Owns *What* (Value) and *How* (Solution Strategy).	Client Product Owner / Vendor Product Manager
Delivery Facilitator	Owns *Flow*, removes impediments, and manages *systemic friction*.	Client Project Manager / Vendor Delivery Lead
Solution Architect	Owns *Coherence*, structural integrity, and architectural *trade-offs*.	Client Solution Architect / Vendor Solution Architect
Technical Integrator	Owns *Technical Execution*, operational readiness, and integration *quality*.	Client Technical Integrator / Vendor Technical Integrator
Requirements Analyst	Owns *Clarity*, domain expertise, and the *translation* of business needs.	Client Requirements Analyst / Vendor Requirements Analyst
Experience Designer	Owns *Usability*, user needs, and the design that enables **Outcome before Output**.	Client Experience Lead / Vendor Experience Lead

Client-Specific Fixed Cores

These roles provide the **governance, funding, and boundaries** for the engagement.

Functional Core (Client-Only)	Purpose
Executive Sponsor	Owns funding, strategic intent, and the highest-level **Trust before Control** decisions.
Vendor Manager	Owns commercial agreements, contracting, and performance against the **Maturity Model**.
Governance Officer	Owns adherence to regulatory, security, and internal audit standards.

Vendor-Specific Fixed Cores

These roles provide the **organizational capacity, quality assurance, and commercial stewardship** needed to sustain the partnership.

Functional Core (Vendor-Only)	Purpose
Account Lead	Owns the overall commercial health and portfolio alignment for the client relationship.
Engineering Director	Manages resource capacity, staff health, and supports the delivery unit's **systemic maturity**.
Practice Lead	Defines and upholds quality standards for specific disciplines (e.g., Architecture, DevEx, QA, DevOps).
Engineering Specialist	The core capacity responsible for hands-on technical implementation and verification.

Summary of Dimensions

1. **Co-Dependent System Roles** – the joint actors of the SDLC.
2. **Client Governance & Sponsorship Roles** – the structural boundary setters.
3. **Vendor Capacity & Stewardship Roles** – the structural sustainers.

Practice Template

Each individual practice file follows this shared structure:

Section	Description
Purpose	Why the tool exists and what maturity gap it closes
Applies To	SDLC stage, relationship line, and maturity level
Actors / Roles	Client + Vendor roles involved (see Functional Role Model)
Steps / Routines	How to apply it collaboratively
Inputs / Outputs	Artifacts or agreements produced
Metrics / Signals	Quantitative and qualitative indicators
Common Pitfalls	Typical misuses or blind spots
Scaling Notes	How to evolve use with maturity
Client-Side Application	Specific guidance for client users
Vendor-Side Application	Specific guidance for vendor users

Using Practice Architecture in Training

This file serves as the **starting point** in all training paths:

1. Identify your **side** (Client or Vendor).
2. Determine your **current stage** (RFP, Discovery, Delivery, Post-v1).
3. Choose the **application mode** (Design, Diagnose, Assess, Reflect, Audit).
4. Open the corresponding tool file.
5. Apply it collaboratively using shared terminology and maturity targets.

Training programs and certifications use these mappings to define **3SF Practitioner Paths** — separate for Client and Vendor, but sharing the same conceptual backbone.

Summary

3SF Practice Architecture bridges the *3SF Theory* and the *Practical Toolset*, creating a shared map for both **Client** and **Vendor** teams to:

- Understand *when* and *why* to use each tool,
- Distinguish respective responsibilities and contributions,
- Build maturity in a synchronized and measurable way.