

How Platform Engineering Is Affected by SDSE

From Golden Paths to Behaviorally Validated Platform Capabilities

The central proposition

Platform engineering provides reusable delivery capabilities. Simulation-Driven Software Engineering provides executable evidence that those capabilities will behave as intended before their assumptions, policies, and automation are scaled across the organization.

By Marc Hornbeek, a.k.a. DevOps_the_Gray

Originator of Simulation-Driven Software Engineering

August 2026

Executive Summary

Platform engineering has emerged as a practical response to the complexity of modern software delivery. A platform team assembles reusable capabilities—developer portals, templates, deployment workflows, APIs, infrastructure services, policy controls, observability, and documentation—into an internal product that enables development teams to deliver with greater consistency and less cognitive load. CNCF describes platforms as collections of capabilities exposed through intentional interfaces and experiences; DORA similarly treats platform engineering as a sociotechnical discipline that includes internal developer platforms, shared services, workflows, and Golden Paths [1][2].

The approach is powerful, but scale magnifies both value and error. A flawed template can replicate an insecure default. A convenient API can obscure unsafe retry behavior. A mandatory control can block recovery. A Golden Path can optimize the happy path while leaving degraded modes, exceptions, ownership, and rollback undefined. Platform teams can verify that automation executes, yet still lack early evidence that the end-to-end developer, system, and operational behavior is the right behavior.

Simulation-Driven Software Engineering (SDSE) closes part of this gap. SDSE begins with outcomes, obligations, constraints, and unacceptable results; develops a Candidate Specification; represents consequential behavior in an executable model; simulates normal, boundary, failure, recovery, misuse, and intervention scenarios; and establishes a versioned Validated Behavioral Baseline. Behavior Simulation Notation (BSN) and the BSN Workbench provide a notation and environment for representing participants, state, interactions, conditions, expectations, traces, and revisions [12].

Core conclusion

SDSE does not replace platform engineering. It makes platform intent executable before it is productized and scaled. Platform engineering turns validated behavior into reusable capabilities; operational evidence keeps the model and platform honest.

1. Golden Paths become behaviorally validated journeys rather than collections of automation steps.
2. Platform APIs, policies, guardrails, and recovery mechanisms can be challenged before broad adoption.
3. Simulation traces can derive conformance tests, telemetry, SLOs, evidence, and runbook expectations.
4. Exceptions and Human-in-Control decisions can be designed as explicit behavior instead of undocumented escape routes.
5. Platform usage, incidents, support demand, and developer feedback can revise both the behavioral model and the platform product.

Contents

Executive Summary

- 1. Why Platform Engineering Matters**
- 2. The Behavioral Gap in Platform Engineering**
- 3. What Simulation-Driven Software Engineering Adds**
- 4. Platform Engineering and SDSE: The Fundamental Comparison**
- 5. The Platform as a Behavioral System**
- 6. Platform-as-a-Product in an SDSE Environment**
- 7. From Golden Paths to Behaviorally Validated Journeys**
- 8. Self-Service, Guardrails, and Policy**
- 9. Platform APIs and Contracts**
- 10. Reliability Engineering for the Platform**
- 11. Observability and Behavioral Evidence**
- 12. Security and Compliance by Simulated Behavior**
- 13. Automation and Human-in-Control**
- 14. AI-Native Platform Engineering**
- 15. The Integrated SDSE-Platform Lifecycle**
- 16. How the Platform Engineer Role Changes**
- 17. Platform Product Management**
- 18. Organizational and Team Implications**
- 19. Practical Adoption Approach**
- 20. Measuring Value**
- 21. Leadership Implications**

Conclusion

Appendices, References, and Author

1. Why Platform Engineering Matters

Modern software teams depend on a growing set of cloud services, security controls, deployment mechanisms, data platforms, observability systems, and compliance obligations. If every product team must discover, assemble, and maintain these independently, delivery becomes fragmented and cognitive load increases. Platform engineering creates a product-oriented capability that makes approved ways of working easier to find, adopt, operate, and improve.

The platform is more than infrastructure. Its product surface may include portals, command-line tools, APIs, templates, workflow engines, policy services, documentation, support, and paved roads. CNCF emphasizes that platform teams provide interfaces and experiences around underlying capabilities rather than merely operating a technology stack [1]. Team Topologies identifies reduced cognitive load and faster flow as central platform outcomes [8].

What disciplined platform engineering provides

6. Self-service access to recurring delivery and operational capabilities.
7. Golden Paths that encode supported patterns without eliminating informed choice.
8. Consistent interfaces to infrastructure, security, data, and operational services.
9. Product management based on developer needs, adoption, experience, and outcomes.
10. Reusable controls and evidence that improve reliability, security, and governance.
11. A managed place to evolve shared capabilities instead of duplicating them in every team.

The platform promise

A platform creates value when teams can achieve important delivery outcomes with less unnecessary effort, while the resulting systems remain reliable, secure, supportable, and governable.

2. The Behavioral Gap in Platform Engineering

Platform engineering commonly validates technology through unit, integration, security, performance, and user testing. It can also observe adoption, satisfaction, support demand, throughput, reliability, and change outcomes. These mechanisms remain essential. The gap appears when the organization must determine whether interacting platform rules will produce an acceptable end-to-end result before the platform capability is implemented or widely consumed.

A template may create a compliant repository but omit the ownership behavior required during an incident. A deployment workflow may succeed technically while permitting an unsafe sequence across dependencies. A policy engine may enforce the intended rule under normal conditions but deadlock

emergency recovery. A self-service action may be simple for the requester while transferring hidden toil and risk to another team. These are system behaviors, not isolated feature defects.

The scaling asymmetry

A product-team defect may affect one service. A platform defect can be reproduced automatically across hundreds of services. The earlier the behavioral evidence, the more valuable the correction.

Typical upstream gaps

12. Golden Paths defined as implementation sequences without explicit outcome or failure semantics.
13. API contracts that specify syntax and responses but not state, timing, idempotency, or responsibility transfer.
14. Guardrails without modeled exception, escalation, revocation, and emergency-access behavior.
15. Metrics that count adoption while missing whether users achieve the intended outcome.
16. Automation that reduces visible steps but obscures authority, dependencies, and recovery obligations.
17. AI-generated or agent-executed changes whose acceptable autonomy has not been made executable.

3. What Simulation-Driven Software Engineering Adds

SDSE is a behavior-first engineering discipline. It creates executable evidence about a proposed system before production implementation exists. The objective is not to simulate every technical detail. It is to represent enough consequential behavior to expose misunderstanding, contradiction, missing outcomes, unsafe paths, and ungoverned authority while changes remain comparatively inexpensive.

The process begins with stakeholder intent and unacceptable outcomes. A Candidate Specification makes the proposed behavior explicit. A BSN model represents relevant participants, interfaces, state, conditions, interactions, scenarios, and expectations. Simulation produces observable traces and counterexamples. After accountable review of the evidence, assumptions, exclusions, model fidelity, and residual uncertainty, the accepted result becomes a Validated Behavioral Baseline.

For platform engineering, that baseline can guide portal journeys, templates, APIs, policy, automation, conformance tests, telemetry, SLOs, documentation, runbooks, and support. The platform product remains the source of implemented capability. The baseline supplies the accepted behavioral promise against which design and operation can be compared.

Platform behaviors suitable for simulation

- 18. Service creation, environment provisioning, deployment, promotion, rollback, retirement, and recovery.
- 19. API ordering, timeout, retry, duplication, compensation, concurrency, and version transition.
- 20. Policy evaluation, exception, escalation, revocation, emergency access, and evidence retention.
- 21. Dependency loss, capacity pressure, partial failure, stale data, and degraded platform modes.
- 22. Developer, platform, security, operations, and automated-agent responsibility transfers.

4. Platform Engineering and SDSE: The Fundamental Comparison

Dimension	Platform engineering	SDSE
Primary question	How can reusable capabilities improve developer flow and organizational outcomes?	Will the proposed capability behave acceptably across the scenarios that matter?
Primary evidence	Adoption, user research, delivery flow, reliability, support demand, cost, tests, and operational experience.	Simulation traces, modeled state, scenario outcomes, expectations, counterexamples, and limitations.
Typical timing	Product discovery through implementation, adoption, operation, and improvement.	Intent through operations; executable evidence can exist before platform code.
Primary artifacts	Portal, APIs, templates, workflows, policies, documentation, services, roadmaps, and support models.	Executable behavior model, scenario portfolio, review evidence, and Validated Behavioral Baseline.
Main strength	Productizes and scales useful delivery capabilities.	Challenges interacting behavior before automation magnifies it.
Principal limitation	Implementation and usage evidence may arrive after important assumptions harden.	A model cannot prove that the implemented platform conforms or that users will adopt it.

Table 1. Platform engineering and SDSE create different but complementary evidence.

The practices overlap in discovery, design, testing, measurement, and learning, but they should not be collapsed. Platform engineering owns the product and its user experience. SDSE supplies an earlier executable medium for examining behavior. Operational results remain authoritative evidence about the implemented platform; simulation remains authoritative evidence about what the organization explicitly chose to promise.

5. The Platform as a Behavioral System

A platform is a system of participants and decisions. A developer requests a capability. A portal or API collects intent. Identity and policy services determine authority. Workflow engines coordinate

infrastructure, repositories, pipelines, secrets, registries, and runtime services. Security and operations teams receive evidence or exceptions. Automated actors change state. The user experiences the combined behavior, not the internal component map.

Participant	Locally successful evidence	End-to-end behavioral risk
Developer portal	Form accepted	The request creates an unsupported architecture or silently omits ownership.
Provisioning workflow	Resources created	Partial failure leaves costly or exposed resources without a responsible owner.
Policy service	Rule enforced	Emergency recovery is blocked because the exception path has no accountable authority.
Deployment service	Release completed	Dependent services observe an incompatible state transition.
Observability stack	Telemetry received	The evidence cannot reconstruct who changed what, why, or whether the outcome was acceptable.

Table 2. Local platform success can coexist with an unacceptable system outcome.

A BSN model can represent the journey across these participants, the state each decision depends on, and the evidence the platform must preserve. The goal is not a more detailed architecture diagram. It is an executable explanation of what must happen, what must never happen, and how the system should respond when assumptions fail.

6. Platform-as-a-Product in an SDSE Environment

Platform-as-a-product means the platform has identifiable users, outcomes, a product strategy, discovery, prioritized capabilities, service expectations, adoption mechanisms, and feedback. SDSE strengthens this product discipline by converting critical product hypotheses into executable behavioral propositions before roadmap commitment.

Product element	Behavioral validation question
User need	Which developer or delivery outcome should become easier, safer, faster, or more reliable?
Product promise	What observable behavior constitutes successful fulfillment of that need?
Unacceptable outcome	What harm, lock-in, hidden toil, loss of control, or governance failure must be prevented?
Adoption hypothesis	Which users, contexts, incentives, constraints, and migration states determine use?
Operating promise	What reliability, support, recovery, change, and deprecation behavior will users depend on?
Evidence plan	What simulation, usability, conformance, telemetry, and outcome evidence will justify investment?

Table 3. SDSE turns important platform product hypotheses into reviewable behavior.

Product discipline remains essential

Simulation can expose a flawed promise; it cannot create user demand. Platform discovery, usability research, adoption support, and outcome measurement remain indispensable.

7. From Golden Paths to Behaviorally Validated Journeys

A Golden Path is an opinionated, supported way to complete a recurring engineering task. CNCF's maturity model links paved roads and Golden Paths to tested, compliant patterns, while Backstage software templates illustrate how platforms can create components from standardized skeletons and publish them into development ecosystems [3][4].

The risk is to treat the path as a sequence of automated steps. A useful path also has preconditions, outcomes, boundaries, degraded modes, cancellation, rollback, ownership, evidence, and an off-path strategy. SDSE makes those dimensions executable before the path is promoted as a default.

Journey dimension	Question to simulate
Entry	Who may initiate the journey, with what intent, authority, and required context?
Happy path	Which interactions and state transitions produce the intended developer and system outcome?
Boundary	When is the path unsuitable, and how does the user recognize that condition?
Failure	What happens after partial provisioning, timeout, policy denial, dependency loss, or user cancellation?
Recovery	How are resources reconciled, retried, rolled back, reassigned, or safely resumed?
Exit	What evidence proves completion, ownership, supportability, compliance, and readiness?
Evolution	How are versions, migrations, exceptions, deprecation, and backward compatibility handled?

Table 4. A Golden Path becomes stronger when its full behavioral journey is explicit.

8. Self-Service, Guardrails, and Policy

Self-service shifts routine decisions and actions toward the user while the platform preserves organizational constraints. Guardrails make preferred behavior easy, prevent unacceptable behavior where necessary, and supply evidence for review. The design challenge is not simply enforcement; it is the complete behavior of request, evaluation, explanation, exception, approval, revocation, remediation, and audit.

23. Simulate permitted, denied, ambiguous, stale, and conflicting policy outcomes.

24. Exercise exception paths, including ownership, expiration, evidence, and compensating controls.

- 25. Test how policy changes affect in-flight journeys and existing resources.
- 26. Represent emergency access and recovery without normalizing permanent bypasses.
- 27. Verify that users receive actionable explanations and can identify the responsible decision owner.
- 28. Detect whether self-service merely transfers hidden labor, cost, or risk to another group.

Guardrail design rule

A guardrail is a behavioral contract. It must define not only what is blocked, but how safe progress, exception, recovery, accountability, and evidence work.

9. Platform APIs and Contracts

Platform APIs expose reusable capabilities to portals, pipelines, product teams, and increasingly to AI agents. An interface specification defines callable operations and data structures, but many failures arise from temporal and stateful behavior: a retry after a timeout, an event delivered twice, a request accepted during version transition, or a partial response that transfers responsibility ambiguously.

Contract dimension	Behavior that must be explicit
Identity and authority	Who or what may call, delegate, approve, revoke, and audit the operation?
State	Which preconditions, transitions, invariants, and terminal states govern the operation?
Time	What do timeout, expiry, scheduling, ordering, and eventual completion mean?
Repetition	Are retries, duplicates, replay, and concurrent requests safe and idempotent?
Failure	What is atomic, compensatable, resumable, or irreversibly committed?
Versioning	How do callers and providers coexist during migration and deprecation?
Evidence	Which correlation, decision, state, and outcome records support conformance and diagnosis?

Table 5. Platform API contracts require behavioral semantics as well as schemas.

Simulation traces can become examples for consumer-driven tests, integration tests, policy tests, and operational dashboards. Production conformance then compares the implemented interactions with the accepted baseline rather than assuming that schema validity implies system correctness.

10. Reliability Engineering for the Platform

A platform is both a product and a dependency. Its reliability affects every team that consumes it, yet not every platform function requires the same objective. Service creation may tolerate delay while credential issuance, deployment control, rollback, or runtime routing may be operationally critical. Reliability engineering must therefore begin with the behavior and consequence of each user journey.

Google’s SRE guidance frames service-level objectives as explicit targets for service behavior [9]. SDSE can connect those targets to modeled developer and system outcomes before sufficient production history exists. Scenarios can exercise dependency failure, overload, degraded service, recovery, data consistency, and user intervention; the resulting expectations can shape SLIs, SLOs, capacity policy, resilience tests, runbooks, and support commitments.

Platform journey	Reliability behavior to validate
Provisioning	Completion, partial failure, orphan prevention, reconciliation, and ownership assignment.
Deployment	Ordering, progressive exposure, compatibility, rollback, abort, and recovery evidence.
Runtime dependency	Availability, latency, freshness, capacity, fallback, and failure containment.
Control plane	Safe degradation, queued work, state persistence, replay, and resumption.
Support	Detection, triage, escalation, communication, restoration, and learning.

Table 6. Platform reliability is defined by journey behavior, not one aggregate uptime number.

11. Observability and Behavioral Evidence

Observability allows teams to understand internal system state from emitted evidence. OpenTelemetry identifies traces, metrics, and logs as core signals and emphasizes deliberate instrumentation [7]. Platform teams also need product signals: journey completion, wait time, failure point, abandonment, exception demand, support burden, adoption, satisfaction, and downstream delivery outcomes.

A simulation trace reveals the state and causal information required to explain why a platform outcome occurred. That trace can be used to define instrumentation before APIs and workflows harden. The organization can then distinguish “the workflow ran” from “the user obtained the intended, supportable, compliant result.”

Model evidence	Derived platform observability
Participant and interface	Actor, service, workflow, API, dependency, and boundary identity.
State and transition	State-change event, version, cause, invariant, timestamp, and decision result.
Journey	Correlation across portal, API, workflow, infrastructure, pipeline, and runtime.
Expectation	Conformance rule, SLI, alert, adoption signal, and product outcome.
Authority	Requester, delegated actor, approval, exception, revocation, and audit evidence.
Recovery	Detection, retry, compensation, rollback, reconciliation, and safe completion.

Table 7. Behavioral traces can define telemetry for product, reliability, and governance questions.

12. Security and Compliance by Simulated Behavior

Platforms often become the organization’s mechanism for embedding secure and compliant practices into delivery. NIST’s Secure Software Development Framework recommends integrating secure practices throughout each software lifecycle [6]. Platform capabilities can make those practices repeatable, but repeatability alone does not establish that the combined behavior is safe.

SDSE allows security and compliance specialists to examine misuse, privilege escalation, separation of duties, approval, evidence retention, policy conflict, emergency access, revocation, supply-chain trust, and remediation before the behavior is automated. Requirements become scenario expectations that can be traced into controls and tests.

Control family	Behavior to simulate
Preventive	Policy enforcement, least privilege, approved provenance, isolation, and safe defaults.
Detective	Recognizable misuse, anomalous authority, drift, failed control, and evidence loss.
Responsive	Containment, credential revocation, rollback, remediation, communication, and escalation.
Evidentiary	Identity, decision basis, artifact lineage, state history, exception, and accountable approval.
Adaptive	Policy revision, migration, revalidation, and response to new threats or obligations.

Table 8. Security controls become more defensible when their behavior is exercised end to end.

Compliance outcome

A control catalog describes what should exist. Behavioral evidence shows how controls interact, fail, recover, and produce auditable outcomes in the platform journey.

13. Automation and Human-in-Control

Platform value depends heavily on automation. Automation creates consistency and speed, but it also compresses decision time and multiplies the consequences of incorrect rules. SDSE can model what an automated actor may observe, decide, change, and communicate; when it must stop; who may override it; and how control is returned safely.

Automation dimension	Behavioral requirement
Authority	Which actions are permitted, prohibited, delegated, time-bound, or scope-bound?
Confidence	What evidence or uncertainty threshold permits automatic execution?
Escalation	When is human approval, specialist review, or incident command required?
Intervention	How can a person pause, modify, cancel, or take control without creating unsafe state?

Automation dimension	Behavioral requirement
Revocation	How are delegated permissions and active operations withdrawn?
Accountability	Which intent, evidence, decisions, actions, and outcomes must be preserved?

Table 9. Human-in-Control is an executable operating design, not a manual approval added at the end.

The objective is proportional control. Low-consequence, reversible actions may be fully automated. High-consequence or poorly understood actions may require bounded authority, preview, approval, staged execution, or mandatory review. Simulation helps stakeholders see the operational consequences before selecting the control model.

14. AI-Native Platform Engineering

AI assistants and autonomous agents are becoming platform users and platform capabilities. They can generate templates, configure infrastructure, diagnose failures, propose policy, open pull requests, operate pipelines, and remediate incidents. This expands productivity and also changes the platform trust boundary: callers may be probabilistic, dynamically planned, and capable of acting at machine speed.

NIST has extended the SSDF with AI-specific secure development practices for generative AI and dual-use foundation models [11]. An SDSE-enabled platform can make acceptable agent behavior explicit through executable scenarios. The platform then enforces identity, least privilege, resource limits, tool boundaries, approval, traceability, rollback, and revocation as product capabilities rather than relying on prompt wording alone.

AI-native condition	Platform behavior to validate
Agent proposes	Validate intent, evidence, policy, impact, reversibility, and responsible owner.
Agent executes	Bound tools, scope, rate, environment, budget, data access, and change authority.
Agent encounters uncertainty	Stop, ask, escalate, or select a safe degraded action.
Agent fails or drifts	Detect, contain, revoke, preserve evidence, reconcile state, and recover.
Human intervenes	Transfer context and control without losing traceability or creating conflicting actions.
Model changes	Revalidate affected scenarios, controls, evaluations, and deployment policy.

Table 10. Agentic delivery requires platform-enforced behavioral boundaries.

15. The Integrated SDSE-Platform Lifecycle

The integrated lifecycle joins two feedback systems. SDSE moves executable evidence toward intent and design. Platform engineering turns accepted behavior into a supported product and learns from real use. Neither model nor implementation is permanently correct; both are revised when new evidence exposes drift, changed assumptions, emerging needs, or unacceptable outcomes.

THE SDSE-PLATFORM BEHAVIORAL VALIDATION LOOP

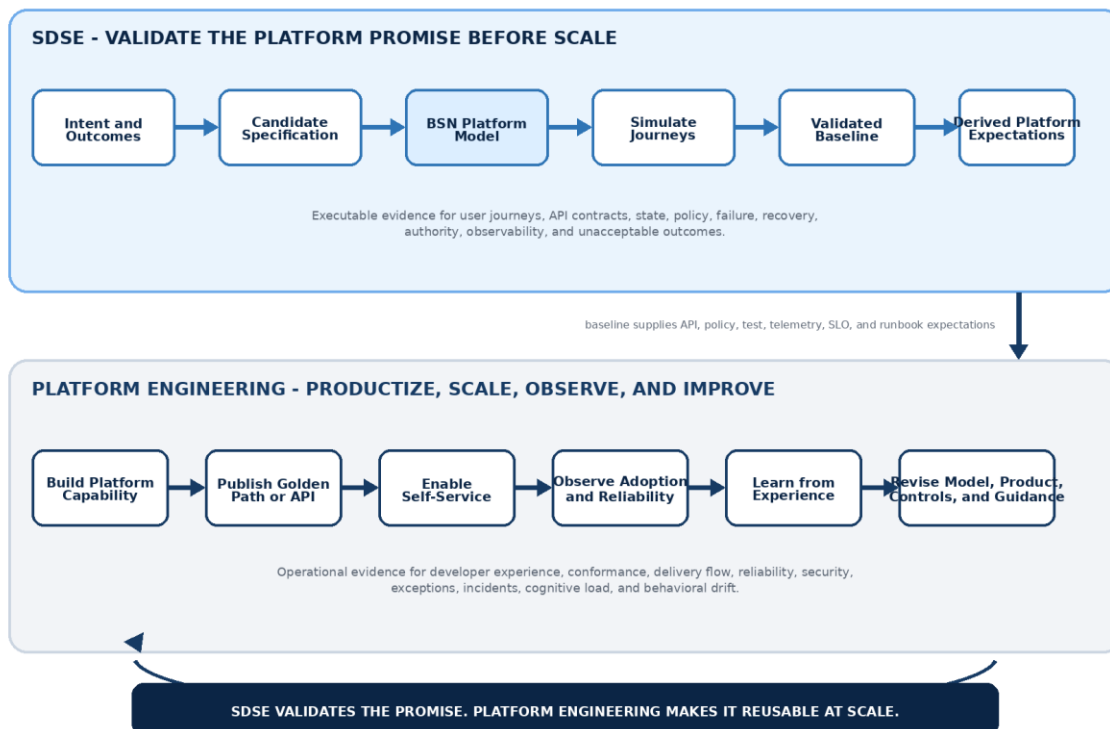


Figure 1. The SDSE-platform lifecycle connects early behavioral validation with productization, operation, and learning.

1. **Select the outcome.** Define the developer, delivery, operational, security, or business result the platform should enable.
2. **State unacceptable outcomes.** Make harm, loss of control, unsafe automation, hidden toil, and governance failure explicit.
3. **Create the Candidate Specification.** Identify participants, boundaries, conditions, interactions, state, obligations, and decision owners.
4. **Build the BSN model.** Represent the consequential platform journey at a fidelity appropriate to the decision.
5. **Simulate scenarios.** Exercise normal, boundary, failure, recovery, misuse, exception, migration, and intervention behavior.

6. **Validate the baseline.** Review traces, assumptions, limits, dissent, residual uncertainty, and acceptance authority.
7. **Derive product expectations.** Translate behavior into APIs, templates, policies, tests, telemetry, SLOs, documentation, and runbooks.
8. **Build and release the capability.** Implement the platform product and verify conformance with the accepted behavior.
9. **Observe use and operation.** Measure experience, adoption, flow, reliability, security, exceptions, support demand, and outcomes.
10. **Revise both systems.** Use evidence to update the model, product, controls, guidance, roadmap, and organizational agreements.

One platform promise

The model, portal journey, API, template, policy, test, telemetry, SLO, runbook, and documentation should remain connected to one versioned behavioral promise.

16. How the Platform Engineer Role Changes

SDSE does not turn platform engineers into enterprise modelers or require every capability to be simulated. It gives them an executable medium for decisions that cross teams, services, policies, or automated actors. Their knowledge of delivery friction, infrastructure, failure, support, interfaces, and product constraints becomes useful earlier.

Lifecycle area	Expanded platform engineer contribution
Discovery	Frame user outcomes, unacceptable outcomes, platform boundaries, and consequence.
Modeling	Represent journeys, interfaces, state, automation, policy, failure, recovery, and responsibility.
Validation	Challenge traces, exceptions, usability assumptions, model fidelity, and residual uncertainty.
Productization	Derive templates, APIs, controls, tests, telemetry, SLOs, documentation, and support behavior.
Operations	Compare actual journeys and outcomes with the baseline; detect drift and emerging needs.
Learning	Translate incidents, feedback, and adoption evidence into model and product revisions.

Table 11. SDSE broadens platform engineering from capability implementation to behavioral stewardship.

The platform engineer increasingly becomes a steward of interfaces and consequences. Technical expertise remains essential, but success also requires facilitation, product thinking, systems reasoning, evidence design, and the ability to make tradeoffs visible to accountable stakeholders.

17. Platform Product Management

Platform product managers prioritize limited capacity across competing needs: usability, reliability, migration, security, cost, support, and new capabilities. Simulation can improve those decisions by showing the consequences of a proposed behavior before implementation estimates and stakeholder commitments create momentum.

11. Express roadmap items as outcome hypotheses with explicit unacceptable outcomes.
12. Use scenario evidence to distinguish a feature request from a systemic platform need.
13. Compare alternative journeys, policy designs, ownership models, and exception mechanisms.
14. Include migration, deprecation, support, and recovery behavior in the product definition.
15. Link adoption measures to whether users achieve the intended result, not merely whether they click or invoke the platform.
16. Preserve model assumptions and limitations as part of roadmap risk and discovery evidence.

Roadmap principle

A platform roadmap should prioritize changes to user and system outcomes. Simulation helps reveal whether the proposed feature can plausibly produce those outcomes and what else must change.

18. Organizational and Team Implications

Platform behavior crosses organizational boundaries. Stream-aligned teams consume capabilities. Enabling teams help develop new practices. Complicated-subsystem and capability teams provide specialized services. Security, SRE, architecture, finance, risk, and operations contribute constraints and evidence. Team Topologies cautions that interaction modes and team boundaries affect flow and cognitive load [8]. SDSE creates a shared executable subject for these groups without dissolving accountability.

17. Assign one accountable owner for accepting each consequential behavioral baseline.
18. Include representative platform users in scenario selection and trace review.
19. Keep capability ownership, platform product ownership, and policy decision rights explicit.
20. Treat exceptions and off-path journeys as designed product behavior, not informal support work.
21. Connect model changes to affected APIs, templates, policies, tests, evidence, and communications.
22. Allow unfavorable evidence to change scope, roadmap, architecture, rollout, and delegated authority.

Team outcome

SDSE supplies a shared behavioral model; it does not create a centralized committee. Decisions should remain close to the teams with the knowledge and accountability to make them.

19. Practical Adoption Approach

Begin with one consequential platform journey, not a transformation program or modeling mandate. Good candidates include a service-creation Golden Path, deployment rollback across dependencies, a policy exception, environment recovery, an API with difficult retry semantics, or an AI agent acting through platform tools. The target decision must still be open to change.

Pilot element	Leadership question
Decision	What platform choice can still change because of the evidence?
Outcome	Which developer, system, operational, security, or business result matters?
Boundary	Which users, actors, services, policies, dependencies, and exclusions are included?
Scenarios	Which normal, failure, recovery, exception, misuse, migration, or intervention cases matter most?
Baseline	What current lead time, support, failure, rework, adoption, risk, or experience evidence exists?
Derived artifacts	Which API, template, policy, test, telemetry, SLO, runbook, or documentation should change?
Decision rule	What evidence will cause the organization to stop, revise, repeat, integrate, or scale?

Table 12. A credible pilot is bounded around a platform decision, not a tool demonstration.

Adoption traps

- 23. Modeling a trivial happy path that cannot reveal platform value.
- 24. Beginning after API, policy, and rollout decisions are effectively fixed.
- 25. Treating every platform capability as equally deserving of simulation effort.
- 26. Counting models or scenarios instead of decisions and outcomes improved.
- 27. Allowing the model and platform implementation to become competing sources of truth.
- 28. Automating traceability before the team proves a useful working practice.

20. Measuring Value

DORA reports that internal developer platforms can improve individual, team, and organizational performance, while poorly designed implementations can reduce change throughput or stability when

developer independence is weakened [2][5]. Measurement must therefore cover platform experience, delivery, control, and business outcomes—not adoption alone.

Value layer	Measures to consider
Capability	Time to specify, model, simulate, review, derive artifacts, and maintain the baseline.
Leading evidence	Conflicts, missing outcomes, unsafe paths, policy gaps, failure gaps, and observability gaps found early.
Decision quality	Material changes to API, journey, architecture, ownership, policy, control, rollout, or roadmap.
Developer experience	Time-to-first-value, task success, cognitive load, wait, abandonment, support demand, and satisfaction.
Delivery outcome	Lead time, rework, change failure, recovery, integration delay, and exception handling.
Platform product	Adoption by target segment, retention, off-path demand, reliability, cost, and deprecation progress.
Business outcome	Time to value, risk exposure, service reliability, compliance effort, customer impact, and avoidable cost.

Table 13. Measure decisions and outcomes improved—not the volume of platform artifacts.

Example value hypothesis

If the team behaviorally validates a consequential Golden Path before implementation, then late clarification, workflow rework, policy exceptions, support demand, and platform-caused failures should decline relative to a comparable baseline, while modeling cost remains lower than the cost avoided or risk reduced.

21. Leadership Implications

Leaders should not position SDSE as another mandatory gate or platform engineering as centralized control. The strategic opportunity is to combine earlier behavioral evidence with reusable, product-managed capabilities. This improves leverage precisely where platform engineering is most powerful: one well-designed capability can help many teams, and one poorly designed capability can harm many teams.

29. Treat the platform as a product whose behavior and operating promises are explicit.
30. Prioritize simulation where scale, consequence, uncertainty, coupling, or automation authority is high.
31. Require Golden Paths to include boundary, failure, recovery, exception, ownership, and evidence behavior.
32. Connect behavioral baselines to platform APIs, policies, tests, telemetry, SLOs, runbooks, and roadmaps.
33. Preserve developer independence by providing supported choices and explicit off-path behavior.
34. Establish Human-in-Control boundaries for consequential automation and AI agents.

35. Scale only after a bounded pilot demonstrates better decisions and measurable product outcomes.

Leadership question

Which platform decision would improve most if stakeholders could execute and challenge its end-to-end behavior before the organization automated and scaled it?

Conclusion

Platform engineering and SDSE address complementary forms of uncertainty. Platform engineering discovers, builds, operates, and improves reusable capabilities that reduce unnecessary developer effort and improve organizational outcomes. SDSE determines whether the proposed behavior of those capabilities is coherent, acceptable, observable, recoverable, secure, and governable before broad implementation and adoption.

The combination changes Golden Paths from static recommendations into validated journeys. It turns API contracts into explicit stateful behavior, guardrails into complete decision systems, observability into evidence of outcomes and causality, and automation into bounded authority with intervention and recovery. For AI-native delivery, these properties become essential because machine actors can exercise platform capabilities dynamically and at scale.

The practical starting point is small but consequential: one important platform decision, one bounded journey, one executable model, one validated scenario portfolio, and one connected set of product artifacts. Build the capability, observe its use, compare reality with the promise, and revise both. That is how platform engineering becomes not merely standardized, but behaviorally trustworthy.

Closing statement

SDSE validates the platform promise. Platform engineering makes that promise reusable at scale. Operational evidence keeps both aligned with reality.

Appendix A. SDSE-Platform Readiness Checklist

Review area	Question
Outcome	Is the platform promise expressed as a developer, delivery, operational, or business outcome?
Users	Have representative consumers and off-path users participated in defining the behavior?
Boundary	Are relevant services, people, policies, suppliers, and automated actors included?
Conditions	Are normal, boundary, degraded, failure, misuse, recovery, migration, and intervention represented?
Contracts	Are state, time, retry, concurrency, version, authority, and responsibility explicit?
Evidence	Can telemetry explain journey, state, causality, decision, recovery, and outcome?
Conformance	Can tests and operational evidence be compared with the accepted baseline?
Learning	Do feedback and incidents revise the model, platform product, controls, and guidance?

Table A1. Use the checklist to identify breaks in the platform behavioral evidence chain.

Appendix B. Platform Journey Scenario Template

Field	Working answer
Intended outcome	_____
Unacceptable outcome	_____
Users, actors, and dependencies	_____
Initial state and conditions	_____
Initiating request or event	_____
Expected normal behavior	_____
Failure, exception, or degraded behavior	_____
Authority and intervention	_____
Recovery, reconciliation, and evidence	_____
Assumptions and residual uncertainty	_____

Table B1. Describe the complete journey, including decisions, failure, recovery, and evidence.

Appendix C. Mapping BSN Evidence to Platform Artifacts

BSN evidence	Derived platform artifact
Participant and interface	Portal role, service boundary, API, ownership, dependency, and access control
State transition	Workflow step, invariant, event, test, reconciliation, and diagnostic evidence
Scenario expectation	Golden Path criterion, policy rule, SLI, SLO, alert, and release condition
Failure behavior	Error contract, resilience test, fallback, compensation, and support procedure
Recovery behavior	Rollback, retry, runbook, reconciliation, handback, and completion evidence
Authority decision	Permission, approval, delegation, exception, escalation, revocation, and audit
Model limitation	Residual risk, monitoring requirement, user guidance, and resimulation trigger

Table C1. The behavioral baseline can organize a coherent platform product evidence portfolio.

Appendix D. SDSE-Platform Pilot Worksheet

Field	Working answer
Platform decision and owner	_____
Target users and outcome	_____
Journey boundary and exclusions	_____
Priority scenarios	_____
Validated expectations	_____
Derived product artifacts	_____
Baseline measures and pilot duration	_____
Observed results and learning	_____
Continuation decision and date	_____

Table D1. Bound the pilot around one consequential platform decision.

Appendix E. Leadership Review Checklist

36.The platform capability has named users, outcomes, unacceptable outcomes, and an accountable owner.

37.The proposed behavior can still change before implementation or broad rollout.

- 38. Representative consumers, platform engineers, domain experts, security, and operations reviewed the scenarios.
- 39. Golden Path boundaries, failure, recovery, exception, migration, and off-path behavior are explicit.
- 40. APIs and automation define state, time, repetition, authority, intervention, and evidence.
- 41. The model traces into tests, telemetry, SLOs, controls, runbooks, documentation, and product decisions.
- 42. Operational learning can revise both the baseline and the platform capability.
- 43. The pilot ends with an explicit decision to stop, revise, repeat, integrate, or scale.

References

The following sources support the descriptions of platform engineering, Golden Paths, developer portals, secure development, observability, team design, SRE, operational excellence, AI-specific secure development, and SDSE. The integrated SDSE-platform framework is presented by the author.

- [1] CNCF Technical Advisory Group App Delivery. Platforms White Paper. [Official white paper](#)
- [2] DORA. “Platform Engineering.” Capability guidance. [Official guidance](#)
- [3] CNCF Technical Advisory Group App Delivery. Platform Engineering Maturity Model. [Official maturity model](#)
- [4] Backstage. “Software Templates.” [Official documentation](#)
- [5] DORA. Accelerate State of DevOps 2024. [Official research report](#)
- [6] National Institute of Standards and Technology. Secure Software Development Framework (SSDF) Version 1.1, SP 800-218, 2022. [Official publication](#)
- [7] OpenTelemetry. “Observability Primer” and “Signals.” [Official documentation](#)
- [8] Team Topologies. “Platform Engineering” and “Key Concepts.” [Official guidance](#)
- [9] Google. “Service Level Objectives.” Site Reliability Engineering. [Official chapter](#)
- [10] Amazon Web Services. AWS Well-Architected Framework: Operational Excellence Pillar. [Official guidance](#)
- [11] National Institute of Standards and Technology. Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: SSDF Community Profile, 2024. [Official publication](#)
- [12] Hornbeek, Marc, with Debashis Bhattacharyya and Jitin Dhanani. Simulation-Driven Software Engineering: A Behavior-First Discipline for Modern Systems, 2026. [Purchase on Amazon](#)

About the Author

Marc Hornbeek, a.k.a. DevOps_the_Gray, is an award-winning engineer, author, educator, executive consultant, and keynote speaker with five decades of experience across systems engineering, packet networks, software development, behavioral simulation, testing, automation, DevOps, DevSecOps, SRE, platform engineering, and intelligent software delivery. His early engineering work included integrated tools for packet-network simulation, performance, conformance, observation, and test automation. He is the originator of Simulation-Driven Software Engineering and leads the development of Behavior Simulation Notation and the BSN Workbench. Marc received the IEEE Region 6 Outstanding Engineer Award in 2016 and has authored multiple books and professional certification courses. He founded PrimaModus® to help organizations model, simulate, validate, and engineer software that behaves as intended.

Contact: mhornbeek@engineeringdevops.com