

Where Simulation-Driven Software Engineering Creates the Most Value

Practical SDSE Use Cases Across Design, Development, Testing, Deployment,
and Operations

The central proposition

SDSE is most valuable when software behavior is consequential, interaction-heavy, state- or sequence-sensitive, autonomous, ambiguous, or expensive to discover after implementation.

By Marc Hornbeek, a.k.a. DevOps_the_Gray

Originator of Simulation-Driven Software Engineering

August 2026

Executive Summary

Software teams are increasingly able to produce code faster than they can establish confidence in what that code will do. Cloud platforms, distributed services, policy engines, event-driven architectures, AI agents, third-party APIs, and automated operations create behavior that emerges from interaction. Static requirements, architecture diagrams, unit tests, and green delivery pipelines remain necessary, but none alone demonstrates that the complete system will honor its promise across state, sequence, failure, recovery, and change.

Simulation-Driven Software Engineering (SDSE) closes that assurance gap by making consequential behavior executable before the production system exists. Teams start with stakeholder intent and a Candidate Specification, express important interactions in an executable behavior model, run normal, boundary, failure, misuse, and recovery scenarios, inspect evidence, and approve a versioned Validated Behavioral Baseline. That model then remains connected to design, code, testing, deployment, operations, and learning.

SDSE is not intended to model every detail of every system. It is a proportional engineering discipline. The strongest use cases combine meaningful consequence with behavioral ambiguity, interaction complexity, state or sequence sensitivity, autonomy, frequent change, or a high cost of late discovery. In those conditions, a modest investment in understanding behavior earlier can avoid a much larger investment in correcting behavior after implementation or production exposure. NASA systems engineering guidance similarly emphasizes balancing effectiveness, cost, schedule, and risk across the lifecycle [6].

Purpose of this paper

This paper gives software and business leaders a practical catalogue of twelve SDSE use cases, shows what evidence each use case should produce, and provides a scorecard and adoption method for selecting a valuable first implementation.

1. Use SDSE to improve consequential decisions, not to maximize the number of models produced.
2. Model the interactions, states, sequences, and controls that determine whether a business promise is kept.
3. Make simulation evidence reusable across development, testing, release, operations, governance, and learning.
4. Assign a Behavior Model Owner and define when operational evidence must trigger model revision and resimulation.

Contents

Executive Summary

- 1. Why Software Use Cases Need Behavioral Validation**
- 2. What Qualifies as a Strong SDSE Use Case?**
- 3. Clarifying Complex Requirements Before Development**
- 4. Validating Distributed Systems and Microservices**
- 5. Engineering AI Agents and Autonomous Systems**
- 6. Security and Intelligent Continuous Security**
- 7. Testing and Continuous Delivery**
- 8. Deployment and Release Decisions**
- 9. SRE, Incident Prevention, and Recovery**
- 10. Platform Engineering and Golden Paths**
- 11. Digital Twins with Executable Behavior**
- 12. Regulated and Safety-Critical Systems**
- 13. Legacy Modernization and System Replacement**
- 14. Cross-Organization and Partner Ecosystems**
- 15. The Living Behavior Model Across the Lifecycle**
- 16. The Role of BSN and the BSN Workbench**
- 17. Selecting the First SDSE Use Case**
- 18. Measuring Use-Case Value**
- 19. From Use Case to Sustainable Practice**

Conclusion

Appendix A. SDSE Use-Case Assessment Worksheet

References

About the Author

1. Why Software Use Cases Need Behavioral Validation

A conventional software use case describes how an actor and a system interact to achieve an outcome. That description is valuable because it organizes intent around observable behavior. Yet modern system behavior rarely follows one clean path. A customer request may traverse identity, policy, data, payment, messaging, third-party, and recovery services. An automated agent may interpret evidence, select an action, call tools, wait for external state, retry, escalate, or stop. The result depends not only on functional correctness, but also on timing, ordering, concurrency, incomplete information, authority, and failure handling.

The central challenge is that prose can describe each participant correctly while leaving their combined behavior underdetermined. Different readers fill gaps with different assumptions. Those differences often remain invisible until code, integration, or production makes them expensive. Tests help, but tests are usually derived after an implementation path has already shaped the team's expectations. A green pipeline proves that configured checks passed; it does not prove that the complete behavioral promise is right.

SDSE makes the use case executable early enough to challenge those assumptions. The model does not replace the requirement, architecture, code, test, or runbook. It connects them around one behavioral promise. A scenario can show which participant acts, in what state, under what guard, with what evidence, and what happens when the expected path does not hold. The simulation trace turns a discussion about interpretation into a review of observable evidence.

Behavioral assurance question

Before asking whether the code passes, ask whether the proposed interactions can produce the intended outcome safely, repeatedly, and recoverably under realistic conditions.

2. What Qualifies as a Strong SDSE Use Case?

The best SDSE candidate is not necessarily the largest system or the most technically sophisticated feature. It is a bounded behavioral decision whose uncertainty has material consequences. Teams should look for the following signals.

Signal	Why it matters
Consequence	Incorrect behavior affects customers, revenue, cost, security, compliance, safety, mission, or trust.
Interaction complexity	Several services, agents, organizations, policies, data sources, or human roles must coordinate.

Signal	Why it matters
State and sequence sensitivity	The same action is safe in one state or order and harmful in another.
Behavioral ambiguity	Stakeholders use the same words but hold different assumptions about response, timing, exception, or ownership.
Autonomy	Software can take consequential action without immediate human approval.
Costly late discovery	A mistake found after integration, release, certification, migration, or customer adoption is expensive to reverse.
Change and reuse	The behavior changes often enough, or is reused widely enough, that a maintained executable baseline compounds value.

Table 1. Strong SDSE use cases combine material consequence with behavioral uncertainty or interaction complexity.

A use case does not need to score high on every signal. A simple interaction may justify simulation if its consequence is severe. A low-consequence interaction may justify it if the behavior is reused across hundreds of products or teams. The decision should be proportional: model enough to expose the uncertainties that matter, and no more than the decision requires.

Selection principle

Choose the smallest behavior that can change an important decision and produce evidence a named owner will use.

3. Clarifying Complex Requirements Before Development

Requirements become risky when stakeholders agree on nouns but not behavior. Terms such as “available,” “approved,” “recovered,” “fraudulent,” or “safe” can hide different assumptions about timing, state, authority, exceptions, and evidence. The ambiguity may span product, domain, engineering, security, compliance, and operations. If it reaches implementation, each group may discover a different system than the one it believed it approved.

How SDSE changes the work

SDSE converts the consequential part of the Candidate Specification into executable scenarios before architecture and code harden. The team identifies participants, states, triggers, guards, expected outcomes, exception paths, time limits, and recovery responsibilities. Simulation workshops force ambiguous terms to become observable behavior. Conflicts are revised in the specification and model together, producing a baseline that stakeholders can challenge and approve.

Evidence produced

The primary evidence is a set of named scenarios and traces linked to requirements, assumptions, acceptance expectations, unresolved questions, and explicit decisions. The evidence should show normal flow, boundary conditions, invalid requests, partial information, timeout, retry, cancellation, human intervention, and recovery where they affect the promise.

Business and engineering value

The value is earlier convergence. Teams reduce late requirement change, architecture churn, acceptance surprises, and duplicated test interpretation. Product and business leaders gain a reviewable representation of the promise without needing to inspect code. Engineering gains a stronger source for design constraints and tests.

Illustrative application

A financial platform models account opening across identity verification, sanctions screening, funding, consent, document expiry, manual review, and abandonment. Simulation reveals that “approved” was being used for three different states and that one timeout path could fund an account before the final control completed.

Practical caution: Do not attempt to simulate the entire requirements catalogue. Start with the interaction where disagreement or consequence is concentrated.

4. Validating Distributed Systems and Microservices

Distributed systems fail in ways that are difficult to infer from static service diagrams. Requests arrive late or out of order; consumers process duplicate events; dependencies partially fail; retries amplify load; caches disagree; and local timeouts interact with global deadlines. Each service can satisfy its contract while the end-to-end behavior still violates the customer promise.

How SDSE changes the work

SDSE models the protocol between participants rather than the internal implementation of every component. The team represents messages, state transitions, time windows, retry and idempotency rules, compensation, concurrency, degraded modes, and ownership of recovery. Scenarios can deliberately reorder events, duplicate calls, withhold responses, or restore dependencies to show whether the system converges safely.

Evidence produced

Useful evidence includes interaction traces, state snapshots, expected and unexpected outcomes, invariant violations, unresolved assumptions, interface constraints, and proposed observability signals. Those artifacts can feed architecture decision records, consumer-driven contracts, integration tests, chaos experiments, and operational dashboards.

Business and engineering value

The business value appears as fewer integration surprises, more reliable architecture decisions, reduced cascading failure risk, and faster diagnosis. SDSE is particularly useful when a failure is emergent: no single service is “wrong,” but their combined retry, queue, cache, or compensation behavior is unsafe.

Illustrative application

An order platform simulates authorization, inventory reservation, fulfillment, cancellation, refund, and notification. A duplicate payment event combined with a delayed cancellation reveals a double-refund path that unit and contract tests did not expose.

Practical caution: Preserve implementation freedom. Model the interaction contract and consequential state, not unnecessary internal algorithms.

5. Engineering AI Agents and Autonomous Systems

AI agents are behavioral systems, not merely generated text or code. They interpret context, select tools, retain memory, act under uncertainty, and may affect production systems or people. Their behavior depends on prompts, models, policies, data, tool responses, state, and operating limits. Accuracy tests alone do not address authority, sequencing, escalation, reversibility, or the combined effects of repeated actions.

How SDSE changes the work

SDSE establishes a control envelope around agent behavior. The model expresses permitted actions, required evidence, approval boundaries, confidence thresholds, cooldown periods, blast-radius limits, escalation paths, override, and stop conditions. Simulation then challenges the agent with ambiguous evidence, stale data, tool failure, conflicting goals, malicious input, repeated retries, and human interruption. This supports the Govern, Map, Measure, and Manage orientation of the NIST AI Risk Management Framework [2][3].

Evidence produced

Evidence should include scenario coverage by consequence, decision traces, authority checks, rejected actions, escalation and override behavior, safe-stop behavior, residual risk, and links to operational telemetry. The record should make clear what was modeled, what was not, and which human remains accountable for deployment and continued operation.

Business and engineering value

The value is controlled autonomy. Teams can grant useful authority incrementally, compare agent and human decisions under the same scenarios, and make explicit tradeoffs between speed, scope, and risk. The model provides a stable behavioral reference even when prompts, models, tools, or data change.

Illustrative application

A production-remediation agent may restart workloads and scale capacity automatically, but requires approval for traffic rerouting and cannot alter more than one region without new evidence. Simulation shows that repeated restart under delayed telemetry can worsen saturation, leading to a cooldown and blast-radius rule before deployment.

Practical caution: A simulation is only as trustworthy as its assumptions and fidelity. Validate important model behavior against controlled observations and retain human authority for residual risk.

6. Security and Intelligent Continuous Security

Security is behavioral. An attacker, user, service, policy engine, identity provider, and response system interact over time. A configuration may be valid in isolation while an action sequence crosses an authorization boundary. Automated security systems also create risk when detection, containment, credential revocation, isolation, or recovery actions interact with business-critical workflows.

How SDSE changes the work

SDSE complements secure development practices by simulating adversarial and control behavior across the lifecycle. Teams model trust boundaries, identities, privileges, policy decisions, misuse paths, detection, escalation, containment, recovery, and evidence preservation. NIST's Secure Software Development Framework is designed to integrate security practices into any software lifecycle; SDSE adds executable behavioral evidence to that integration [4].

Evidence produced

The evidence includes attack and misuse scenarios, control-response traces, authorization invariants, false-positive and false-negative consequences, safe containment rules, recovery conditions, and links to security tests and operational detections. The model can also reveal controls that are individually reasonable but harmful in combination.

Business and engineering value

The value is earlier identification of exploitable sequences and safer automation of security response. Security, engineering, and operations gain a shared way to review how controls behave, not only whether controls exist. The resulting evidence strengthens threat modeling, assurance, incident preparation, and risk acceptance.

Illustrative application

A compromised service account triggers automated credential revocation and workload isolation. Simulation reveals that simultaneous revocation prevents the recovery controller from accessing the evidence needed to restore service. The design is changed to preserve a limited, auditable recovery path.

Practical caution: Do not expose sensitive operational detail more widely than required. Use sanitized scenarios and appropriate access control for security models and traces.

7. Testing and Continuous Delivery

Testing usually begins with an implemented system and asks whether configured expectations are met. That is essential, but it can reproduce the same misunderstanding present in the requirement or design. A pipeline can remain green while the test suite omits a critical sequence, state transition, or recovery path. Faster delivery increases the frequency with which those omissions can reach users.

How SDSE changes the work

SDSE creates a behavior-first source for test intent. Validated scenarios and EXPECT outcomes can be transformed into acceptance criteria, integration tests, property checks, fault-injection experiments, release gates, and production probes. Model changes can trigger targeted regression simulation before code changes are merged, while code and telemetry can later be compared with model expectations.

Evidence produced

Evidence includes traceable links from behavioral promise to scenario, expected outcome, test, pipeline result, deployment control, and telemetry. Coverage should be organized by consequence and uncertainty rather than by raw test count. Divergence between the model and implementation becomes an explicit decision: fix the code, revise the model, or change the promise.

Business and engineering value

The value is stronger assurance without relying on indiscriminate test growth. Teams find missing tests earlier, focus expensive environments on high-value scenarios, reduce flaky or redundant checks, and make release evidence easier to explain. The model also provides continuity when code is regenerated or refactored.

Illustrative application

A subscription service derives end-to-end scenarios for upgrade, downgrade, grace period, payment failure, reinstatement, and cancellation from the validated behavior model. A simulated sequence shows that reinstatement after delayed payment can bypass an entitlement update, and the missing test is added before release.

Practical caution: “Green” means the selected checks passed. Keep reviewing whether the selected checks still represent the behavioral promise.

8. Deployment and Release Decisions

Deployment changes the conditions in which software behaves. Version skew, data migration, feature flags, traffic shifting, dependency compatibility, rollback limits, and partial rollout create temporary states that are absent from the steady-state design. Release plans often describe steps without testing the combined behavior of those steps under interruption or failure.

How SDSE changes the work

SDSE models deployment as a consequential system behavior. Scenarios represent preconditions, rollout stages, health evidence, hold points, rollback triggers, irreversible actions, migration compatibility, operator authority, and recovery after partial completion. The team can compare blue-green, canary, rolling, or phased strategies against the same behavioral risks.

Evidence produced

Evidence includes a simulated release trace, decision gates, required telemetry, stop and rollback conditions, accepted irreversibility, version-compatibility assumptions, and responsibility for intervention. The evidence can feed release review and automated policy where appropriate.

Business and engineering value

The value is a better release decision: whether to proceed, pause, rollback, or collect more evidence. Teams can reduce change failure, shorten uncertain approval discussions, and expose release designs whose recovery path is weaker than their forward path.

Illustrative application

A database migration is simulated while old and new application versions run concurrently. The model reveals that rollback after a particular schema transformation would restore software but corrupt a downstream reconciliation process. The release plan adds a hold point and alternate recovery path.

Practical caution: A model cannot guarantee production safety. Treat it as one evidence source alongside testing, observability, change history, and accountable operational judgment.

9. SRE, Incident Prevention, and Recovery

Reliability emerges from how systems degrade, detect, respond, and recover. Runbooks describe intended actions, but severe incidents rarely follow a clean script. Signals may be delayed or misleading, dependencies may fail together, responders may act concurrently, and remediation can amplify harm. Post-incident actions often remain disconnected from design and test evidence.

How SDSE changes the work

SDSE makes resilience and recovery paths executable before or after an incident. Teams model failure propagation, detection, alerting, human and automated response, service degradation, failover, rollback, data reconciliation, and return to normal operation. Game-day scenarios can be rehearsed against the model before consuming production-like environments, and incident findings can update the living baseline.

Evidence produced

Evidence includes failure and recovery traces, detection gaps, unsafe remediation sequences, recovery-time assumptions, observability requirements, runbook decisions, and model-to-reality differences. The most valuable output may be a counterexample showing why a plausible recovery action is unsafe under a particular state.

Business and engineering value

The value is fewer preventable incidents, faster and safer recovery, better prepared responders, and a shorter learning loop. Operational evidence no longer ends in a retrospective document; it becomes a change to the behavioral baseline, scenarios, tests, controls, and telemetry.

Illustrative application

A failover scenario shows that two regional controllers can both become active after a network partition. The team adds a lease rule, evidence threshold, and reconciliation behavior, then carries the scenario into chaos testing and the operational runbook.

Practical caution: Do not confuse simulated recovery time with production recovery time. Calibrate the model with observed latency, load, organizational response, and dependency behavior.

10. Platform Engineering and Golden Paths

Internal platforms encode recommended ways to build, deploy, observe, and operate software. A golden path may provide templates and controls, yet teams still need to understand how those elements behave together under change and failure. Platform capabilities also create broad consequence because one defect or unsafe default can propagate across many product teams.

How SDSE changes the work

SDSE lets platform teams define a behavioral golden path. The model captures expected service onboarding, identity, policy, secrets, deployment, observability, scaling, failure handling, and recovery. Teams can simulate platform changes against representative product scenarios before rolling them out widely. Product teams can reuse platform behavior models and extend them only where their domain differs.

Evidence produced

Evidence includes reusable scenarios, platform invariants, expected service-platform interactions, version compatibility, exception behavior, adoption constraints, and release criteria for platform capabilities. Model reuse creates a common vocabulary between the platform and its consumers.

Business and engineering value

The value is leverage. A validated behavioral pattern can reduce duplicated design and testing across teams, make guardrails easier to explain, and expose unsafe defaults before they scale. The platform becomes a provider of reusable evidence, not only reusable infrastructure.

Illustrative application

A platform team models the default deployment path with service identity, secret rotation, canary analysis, automatic rollback, and telemetry registration. Simulation reveals that a secret rotation during rollback can leave the restored version unable to authenticate. The platform sequence is corrected once for all consuming teams.

Practical caution: Golden paths need escape mechanisms. Model exceptions and ownership explicitly so teams do not bypass controls informally.

11. Digital Twins with Executable Behavior

Many digital twins emphasize geometry, configuration, sensor state, or analytics while treating behavior as code hidden elsewhere. Without executable behavior, the twin may show what the asset is and what has happened but provide limited evidence about what it will do under changed conditions, competing commands, faults, or recovery actions.

How SDSE changes the work

SDSE supplies the behavioral layer. The model represents participants, control logic, state transitions, timing, coordination, faults, degradation, intervention, and recovery. It can execute against synthetic inputs during design and against bounded operational data during validation. The behavioral twin becomes a place to test proposed actions before applying them to the real system.

Evidence produced

Evidence includes scenario traces, expected versus observed behavior, calibration differences, safe operating envelopes, control assumptions, and explicit limits of fidelity. The model should distinguish between known system behavior, estimated behavior, and behavior not represented.

Business and engineering value

The value is safer change, better operational planning, earlier control validation, and more useful digital-twin investment. It is especially strong when software behavior coordinates physical, cyber, or organizational participants and when real-world experimentation is costly or hazardous.

Illustrative application

A warehouse twin models robots, human pickers, chargers, doors, inventory services, and emergency stops. A simulated charger outage combined with a priority-order surge exposes a congestion and deadlock path that a spatial model alone would not reveal.

Practical caution: Never present a behavioral twin as more precise than its calibration supports. State uncertainty and validate important decisions against real evidence.

12. Regulated and Safety-Critical Systems

Regulated and safety-critical software must do more than work; it must support a defensible assurance argument. Requirements, hazards, controls, design, tests, operational limits, evidence, and approvals

must remain connected as the system changes. Static documents can be complete yet inconsistent, and manual traceability can lag behind implementation.

How SDSE changes the work

SDSE provides executable support for assurance, not a replacement for applicable standards or independent assessment. Teams model safety- or compliance-relevant behavior, hazardous sequences, control actions, degraded modes, human intervention, evidence requirements, and recovery. The validated behavioral baseline is versioned and traced to requirements, hazards, controls, tests, and operating procedures. NIST SP 800-160 similarly frames trustworthy systems engineering around lifecycle processes and assurance [5].

Evidence produced

Evidence includes scenario-to-requirement and scenario-to-control traceability, invariant checks, review records, assumptions, model limitations, residual risk, approval, and change impact. The evidence package should make it easier to explain why a control is adequate and how a material change was evaluated.

Business and engineering value

The value is stronger assurance continuity, earlier discovery of unsafe interactions, reduced evidence reconstruction, and clearer change impact. For business leaders, this can reduce certification surprises and improve confidence that accelerated development has not outrun governance.

Illustrative application

A medical workflow models device state, clinician action, patient context, alert timing, network loss, override, and safe shutdown. Simulation exposes a sequence in which a delayed acknowledgement suppresses a second alert. The requirement, model, control, test, and operating guidance are revised together.

Practical caution: The model and tool are not themselves certification. Map their outputs to the governing standard, qualified process, independent review, and organizational risk authority.

13. Legacy Modernization and System Replacement

Legacy systems contain behavior that is only partly documented. Business rules may live in code, configuration, batch schedules, operator workarounds, downstream expectations, and exception handling learned over years. Replacing the visible functions without discovering these behaviors can create a modern system that is technically cleaner but operationally wrong.

How SDSE changes the work

SDSE supports behavioral archaeology. Teams observe important transactions and exceptions, interview domain and operations experts, capture sequences and state rules, and simulate the resulting model against historical or synthetic cases. The model becomes a negotiated description of which behavior must be preserved, intentionally changed, or retired before the new implementation is selected.

Evidence produced

Evidence includes discovered rules, scenario traces, known deviations, source observations, decision ownership, compatibility constraints, and a migration baseline. Side-by-side scenarios can compare legacy, target, and transitional behavior. Unknowns are retained explicitly instead of silently converted into assumptions.

Business and engineering value

The value is reduced migration surprise, better scope control, safer parallel operation, and less dependency on undocumented expertise. The team can decide where compatibility is essential and where change creates business value, rather than treating every legacy behavior as either sacred or accidental.

Illustrative application

A claims platform replacement models intake, duplicate detection, manual review, reserve changes, payment, reopening, and regulatory notices. Historical cases reveal that an overnight batch sequence created a de facto grace period relied on by operations. Leaders decide whether to preserve or formally remove it.

Practical caution: Observed legacy behavior is not automatically desired behavior. Record whether each behavior is required, tolerated, prohibited, or undecided.

14. Cross-Organization and Partner Ecosystems

Business outcomes often cross organizational boundaries. Partners, suppliers, customers, regulators, and shared platforms operate under different systems, incentives, clocks, and failure practices. Interface specifications describe messages, but they may not define what happens when one party retries, delays, disputes, revokes, or recovers. No single organization controls the complete behavior.

How SDSE changes the work

SDSE creates a shared interaction model focused on observable contracts. Each party can retain implementation privacy while agreeing on messages, states, time windows, obligations, evidence, dispute paths, compensation, and recovery. Joint scenarios reveal incompatible assumptions before integration or contract commitments become expensive.

Evidence produced

Evidence includes agreed scenarios, role and responsibility boundaries, protocol traces, timing assumptions, version rules, exception ownership, service evidence, and escalation conditions. The model can support contract language, conformance tests, onboarding, joint readiness, and incident coordination.

Business and engineering value

The value is reduced integration delay, fewer partner disputes, faster onboarding, and clearer accountability. Simulation is especially helpful when the business process depends on several organizations completing a sequence and when a failure cannot be assigned to one component alone.

Illustrative application

A logistics ecosystem models retailer, carrier, customs broker, payment service, and customer notifications. Simulation reveals that a late customs rejection arrives after the retailer considers fulfillment final, leaving no agreed compensation path. The partners revise state definitions and evidence obligations before launch.

Practical caution: Keep the shared model at the contract boundary. Do not require parties to expose proprietary implementation detail that is unnecessary to validate the interaction.

15. The Living Behavior Model Across the Lifecycle

The greatest SDSE value does not come from one pre-development simulation. It comes from continuity. The same behavioral promise is refined and reused as the system moves from intent to architecture, development, testing, deployment, and operations. Each stage produces different evidence, but the model preserves the thread that makes those artifacts coherent.

Lifecycle stage	Behavioral contribution	Decision or evidence
Intent and specification	Candidate Specification, participants, promises, assumptions, constraints	Stakeholder acceptance of modeled scope and unresolved questions
Architecture and design	Interaction contracts, state, sequence, control, timing, recovery	Architecture decisions and explicit tradeoffs
Development	Scenario-derived acceptance examples and implementation traceability	Code and configuration linked to behavioral obligations
Testing	Normal, boundary, failure, misuse, and recovery scenarios	Test evidence aligned to the same EXPECT outcomes
Deployment	Rollout, compatibility, hold, rollback, and stop behavior	Release decision supported by simulation and environment evidence
Operations and learning	Telemetry, incidents, overrides, drift, and model-reality differences	Revised model, regression scenarios, controls, and new baseline

Table 2. The behavior model is a living engineering asset, not a one-time design document.

Continuity also controls change. A material change to the promise, participant, authority, state, timing, interface, failure mode, or operating environment should trigger impact analysis and targeted resimulation. Operational evidence can invalidate an assumption or reveal missing behavior; the model must then be revised rather than defended. Versioned baselines allow teams to know which behavior was approved for which release and which evidence justified the decision.

Living-model rule

If a production observation materially changes what the team believes the system can do, the behavioral baseline, scenarios, tests, controls, and operational guidance should change together.

16. The Role of BSN and the BSN Workbench

Behavior Simulation Notation (BSN) provides a practical notation for expressing interacting behavior in a form that domain, engineering, assurance, and operations stakeholders can discuss and execute. A useful behavioral notation must be precise enough for simulation while remaining readable enough to

support collaborative review. It should make participants, actions, messages, state, guards, expectations, and outcomes visible without requiring the behavior to be buried in implementation code.

The BSN Workbench supports the workflow around that notation: authoring and organizing models; executing scenarios; inspecting traces and outcomes; recording expected behavior; revising assumptions; maintaining versions; and connecting the validated model to downstream artifacts. The single-user edition is intended to be available as an open-source download under the MIT License, lowering the barrier for learning, experimentation, and bounded pilots.

Workbench function	Practical purpose
Model	Express participants, interactions, states, decisions, timing, failure, and recovery.
Simulate	Execute normal, boundary, failure, misuse, adversarial, and recovery scenarios.
Challenge	Inspect traces, counterexamples, violated expectations, ambiguity, and model limits.
Validate	Approve a versioned behavioral baseline with named ownership and residual risk.
Trace	Connect behavior to requirements, architecture, code, tests, controls, deployment, and telemetry.
Learn	Use operational evidence to revise the model and trigger targeted regression simulation.

Table 3. BSN Workbench supports an evidence-producing lifecycle rather than a diagram-only modeling exercise.

Tooling principle

The Workbench should make the behavioral decision easier to understand, simulate, review, trace, and maintain. Tool usage is not the outcome; better evidence is.

SDSE Use Cases Across the Software Lifecycle

SDSE USE CASES ACROSS THE SOFTWARE LIFECYCLE

A living behavior model connects intent, delivery evidence, and operational learning.

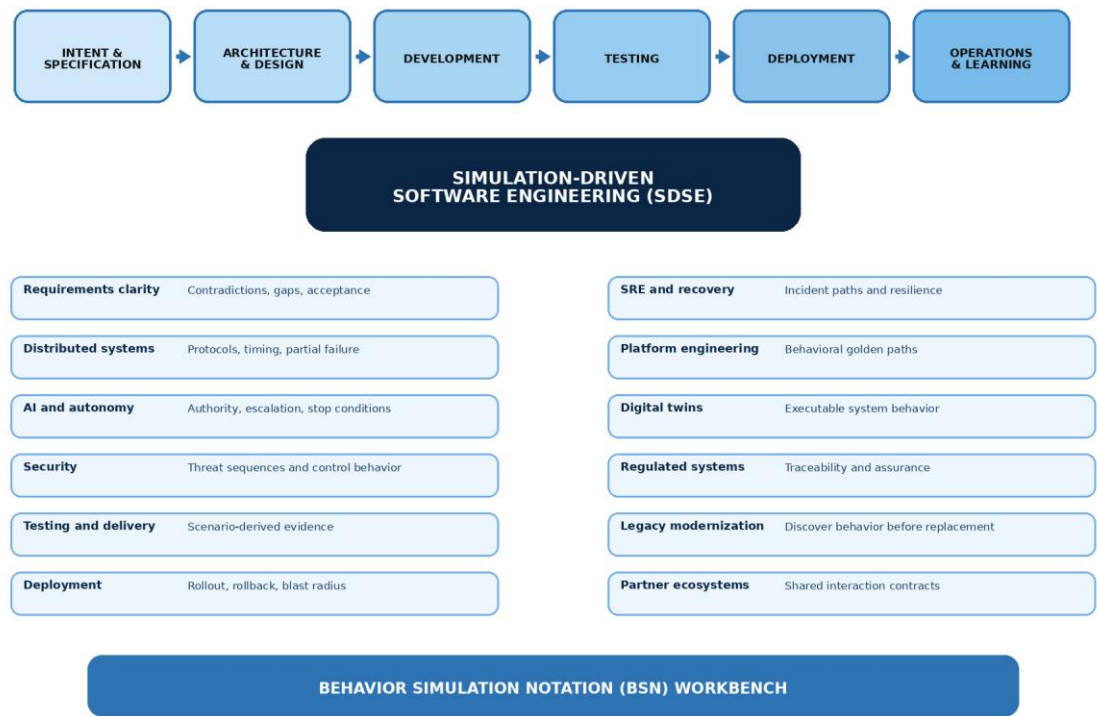


Figure 1. SDSE applies a living behavior model across the lifecycle; the twelve use cases show where that model can change consequential decisions.

17. Selecting the First SDSE Use Case

The first use case should demonstrate decision value without requiring the organization to transform its entire lifecycle. Select a bounded behavior with a named business consequence, visible uncertainty, accessible stakeholders, and evidence that can be observed within a practical period. Avoid both extremes: a trivial example that proves only that the tool runs, and an enterprise-wide model whose scope prevents learning.

Score each criterion from 0 to 2. Zero means low or absent; one means moderate; two means high. The score is a structured conversation, not a statistical prediction. A high total is useful only when a sponsor can define the decision the evidence will change and a Behavior Model Owner can maintain the result.

Criterion	Question	Score
Business consequence	Customer, revenue, cost, security, compliance, safety, mission, or trust impact	0–2
Behavioral ambiguity	Different interpretations, implicit rules, or uncertain expectations	0–2
Interaction complexity	Multiple services, agents, roles, organizations, policies, or data sources	0–2
State and sequence sensitivity	Outcome depends materially on timing, order, concurrency, or prior state	0–2
Autonomy	Software can act without immediate human approval	0–2
Change frequency or reuse	Behavior changes often or is reused broadly enough for a living model to compound value	0–2
Evidence availability	Baseline and post-pilot outcomes can be observed credibly	0–2
Ownership and sponsorship	Named owner, decision authority, and cross-functional participation exist	0–2

Table 4. Use-case selection scorecard; maximum score 16.

Practical interpretation

12–16: strong candidate. 8–11: promising if scope and evidence are tightened. 0–7: select a different behavior unless consequence alone justifies an evidence-focused pilot.

A five-step pilot definition

1. **Name the promise.** State the outcome that customers, operators, regulators, or business leaders believe the system must keep.
2. **Bound the behavior.** Identify the participants, decisions, states, interactions, and exceptions that materially determine that promise.
3. **Name the decision.** Define what the simulation evidence can cause leaders to accept, revise, reject, restrict, or observe.

4. **Define the evidence.** Agree on required scenarios, traces, expectations, baseline measures, review, and acceptance before modeling begins.
5. **Assign lifecycle ownership.** Name who maintains the model, who approves the baseline, and which changes or observations trigger resimulation.

18. Measuring Use-Case Value

Measure whether SDSE changed an important decision or outcome. Counts of models, scenarios, or simulation runs are activity measures; they can help manage the practice but do not establish value. A credible measurement design begins with a baseline and identifies the mechanism through which earlier behavioral evidence should change delivery, risk, cost, or trust.

Measure	Definition	Evidence source
Earlier discovery	Material ambiguities, unsafe sequences, missing controls, or recovery defects found before code, integration, or release	Issue and decision records; phase of discovery
Decision latency	Time from a consequential behavioral question to evidence-backed acceptance, revision, rejection, or escalation	Decision log and review timestamps
Avoided late rework	Reduction in requirement, design, integration, test, release, or incident work caused by behavioral correction	Change records, tickets, time sampling
Assurance coverage	Critical promises linked to scenarios, expectations, tests, controls, deployment, and telemetry	Traceability and evidence record
Change and incident outcomes	Behavior-related change failures, customer-impact time, unsafe automation, or recovery effort	Delivery, incident, security, and SRE systems
Learning-loop speed	Time from material operational observation to model revision, resimulation, and approved new baseline	Model history, incident actions, release records
Economic realization	Verified cash, redeployed capacity, avoided expected loss, or earlier value attributable to the use case	Finance-approved calculations and owners

Table 5. Value measures connect behavioral evidence to operating and business outcomes.

Use ranges and attribution factors when several interventions affect the same outcome. Do not treat every simulated counterexample as a production incident certainly prevented. Do not treat recovered engineering hours as cash savings unless a realization plan changes actual spend or produces measurable contribution. The SDSE ROI and Business Cases white paper provides a fuller financial method; this use-case paper focuses on selecting and measuring the operational intervention.

Measurement discipline

Establish definitions before the pilot, preserve unfavorable evidence, record confounding changes, and let observed results replace optimistic assumptions.

19. From Use Case to Sustainable Practice

A successful pilot proves that behavioral simulation can improve a decision. Sustainable adoption requires a lightweight operating model that makes the behavior model part of normal engineering rather than a specialist artifact. The objective is not centralized modeling of every team. It is a reusable method, shared notation, available coaching, clear ownership, and integration with existing product, architecture, development, testing, release, security, and reliability work.

Stage	Practice	Outcome
1. Prove	Run one bounded, consequential use case with a baseline and explicit evidence gate.	Pilot evidence and scale/revise/stop decision
2. Standardize	Define minimum model content, scenario taxonomy, review, versioning, traceability, and ownership.	Team-ready playbook and reusable templates
3. Integrate	Connect simulation outputs to requirements, architecture, tests, pipelines, controls, release, and telemetry.	Less duplication and visible lifecycle continuity
4. Reuse	Create libraries of common platform, security, recovery, partner, and domain behaviors.	Faster onboarding and broader leverage
5. Govern proportionally	Apply stronger evidence and review to higher-consequence behavior while keeping low-risk work lightweight.	Assurance effort aligned to risk
6. Learn	Use operational observations and incidents to update models, scenarios, controls, and baselines.	Shorter model-reality feedback loop

Table 6. Adoption scales through evidence, integration, reuse, proportional governance, and learning.

Minimum operating roles

6. Business or product sponsor: owns the promise, consequence, and decision the use case must improve.
7. Behavior Model Owner: maintains the model, scenarios, baselines, traceability, and change triggers.
8. Domain and operational experts: contribute real rules, exceptions, failure experience, and model-reality evidence.
9. Engineering and assurance reviewers: connect the baseline to architecture, code, tests, security, release, and operations.
10. Risk or decision authority: accepts residual risk and approves scope, authority, release, or scaling where required.

Conclusion

SDSE creates the most value where software behavior cannot be understood reliably from isolated components or static descriptions alone. The twelve use cases in this paper span requirements, distributed systems, AI agents, security, continuous delivery, deployment, reliability, platforms, digital twins, regulated systems, modernization, and partner ecosystems. Their common feature is not technology. It is the need to make a consequential behavioral decision before reality makes the decision expensive.

The practical path is deliberately modest: select one bounded promise; express the critical behavior; simulate normal, boundary, failure, misuse, and recovery scenarios; inspect the evidence; revise what is wrong; approve a baseline; connect it to implementation and operations; and measure whether the decision or outcome improved. The BSN Workbench provides a practical foundation for that living behavior model, while accountable people retain ownership of the promise and residual risk.

Closing principle

When the cost of being wrong is greater than the cost of understanding behavior before implementation, SDSE becomes an essential engineering investment.

Appendix A. SDSE Use-Case Assessment Worksheet

Complete this worksheet with the business sponsor, Behavior Model Owner, domain experts, and the people responsible for design, assurance, deployment, and operation. Keep answers short enough to review in one working session; attach supporting evidence separately.

Field	Working answer
Use-case name	What bounded behavior is being considered?
Business promise	What outcome must the system keep, for whom, and under what conditions?
Decision	What can the simulation evidence cause us to accept, revise, reject, restrict, or observe?
Consequence	What happens if the behavior is wrong, delayed, unsafe, unrecoverable, or disputed?
Participants	Which services, agents, people, partners, policies, data sources, or physical assets interact?
State and sequence	Which prior state, timing, ordering, concurrency, retry, or timeout conditions matter?
Normal scenarios	Which representative success paths must be demonstrated?
Challenge scenarios	Which boundary, failure, misuse, adversarial, degraded, and recovery paths matter?
EXPECT outcomes	What observable result, invariant, evidence, or stop condition defines acceptability?
Model limits	What behavior, fidelity, data, or operating condition is not represented?
Baseline evidence	What current rework, delay, incident, cost, decision, or assurance data is available?
Success measures	Which decision, delivery, risk, learning, and economic measures should change?
Owners	Who owns the promise, model, review, approval, residual risk, and operational learning?
Change triggers	Which changes or observations require impact analysis and resimulation?
Pilot decision gate	What evidence will justify scale, revision, more observation, or stopping?

Table 7. Practical worksheet for defining an SDSE use case and pilot evidence gate.

Appendix B. Use-Case Catalogue at a Glance

Use case	Primary trigger	Core evidence	Expected value
Requirements	Ambiguous promise or acceptance	Executable scenarios and agreed baseline	Earlier convergence and less late rework
Distributed systems	Emergent interaction and partial failure	Protocol traces and invariants	Fewer integration and resilience surprises
AI/autonomy	Consequential action under uncertainty	Authority, escalation, override, stop evidence	Controlled autonomy and clearer accountability
Security	Adversarial sequence and automated response	Misuse and control-response traces	Earlier vulnerability and safer-response discovery

Use case	Primary trigger	Core evidence	Expected value
Testing/CD	Green checks with incomplete behavioral coverage	Scenario-derived tests and traceability	Stronger assurance with focused coverage
Deployment	Temporary states, version skew, rollback	Release and recovery decision traces	Lower change risk and better release decisions
SRE	Complex failure and recovery behavior	Failure, response, and recovery evidence	Fewer preventable incidents and faster learning
Platform	Shared defaults with enterprise blast radius	Reusable behavioral golden paths	Cross-team leverage and safer standards
Digital twin	State representation without executable behavior	Calibrated behavioral scenarios	Safer change and more useful twins
Regulated/safety	Need for defensible lifecycle assurance	Traceable scenarios, controls, and residual risk	Stronger assurance continuity
Modernization	Undocumented legacy rules and workarounds	Discovered behavior and migration baseline	Reduced replacement surprise
Partner ecosystem	Shared outcome without shared control	Observable interaction contract and scenarios	Faster integration and clearer accountability

Table 8. Summary of the twelve SDSE use cases.

References

Sources were accessed on August 15, 2026. References support the contextual claims in this paper; the SDSE method and use-case framework are presented by the author.

- [1] Google Cloud. “Announcing the 2025 DORA Report.” [Official report announcement](#)
- [2] National Institute of Standards and Technology. “AI Risk Management Framework.” [Official NIST resource](#)
- [3] NIST AI Resource Center. “AI RMF Playbook.” [Official playbook](#)
- [4] National Institute of Standards and Technology. “SP 800-218, Secure Software Development Framework (SSDF) Version 1.1.” [Official publication](#)
- [5] National Institute of Standards and Technology. “SP 800-160 Vol. 1 Rev. 1, Engineering Trustworthy Secure Systems.” [Official publication](#)
- [6] National Aeronautics and Space Administration. “Systems Engineering Handbook 2.5: Cost Effectiveness Considerations.” [Official NASA guidance](#)
- [7] Hornbeek, Marc, with Debashis Bhattacharyya and Jitin Dhanani. Simulation-Driven Software Engineering: A Behavior-First Discipline for Modern Systems. [Purchase on Amazon](#)

About the Author

Marc Hornbeek, a.k.a. DevOps_the_Gray, is an engineer, author, educator, and the originator of Simulation-Driven Software Engineering. His work emphasizes the practical connection between

engineering intent, executable simulation, software tools, lifecycle evidence, DevOps, security, reliability, and operational learning. He is the principal author of Simulation-Driven Software Engineering: A Behavior-First Discipline for Modern Systems and the creator of Behavior Simulation Notation and the BSN Workbench.

PrimaModus® provides consulting and training to help teams learn and apply SDSE and the BSN Workbench to consequential software decisions.

Contact: mhornbeek@engineeringdevops.com