

How Agile Is Affected by Simulation-Driven Software Engineering

How Executable Behavioral Evidence Strengthens Discovery, Planning,
Delivery, Feedback, and Continuous Improvement

The central proposition

SDSE does not replace Agile or make it less iterative. It adds an earlier executable feedback loop so teams can learn about consequential behavior before that behavior is embedded in code.

By Marc Hornbeek, a.k.a. DevOps_the_Gray

Originator of Simulation-Driven Software Engineering

August 2026

Executive Summary

Agile transformed software development by shortening feedback cycles, increasing collaboration, delivering working software incrementally, and enabling teams to respond to change. Those contributions remain essential. Yet Agile teams can still discover consequential behavioral misunderstandings only after they have designed, implemented, integrated, and tested enough software to observe them. The process is iterative, but the first executable representation of intent may still be the code itself.

Simulation-Driven Software Engineering (SDSE) adds an earlier executable feedback loop. Teams create a Candidate Specification, express selected consequential behavior in a model, simulate normal and challenging scenarios, inspect traces and expectation results, revise flawed assumptions, and approve a versioned Validated Behavioral Baseline. Behavior Simulation Notation (BSN) and the BSN Workbench provide a practical language and environment for this work. The baseline remains connected to architecture, code, tests, delivery controls, telemetry, incidents, and operational learning.

SDSE therefore affects Agile without replacing it. Product discovery expands into behavioral discovery. Backlog refinement exposes states, sequences, permissions, failure, recovery, and cross-team interactions. Important acceptance criteria become executable expectations. Iteration planning begins with better information. Definitions of Ready and Done can include proportional behavioral evidence. Sprint Reviews can demonstrate both the implemented increment and its relationship to the approved behavioral promise. Retrospectives and operational reviews can improve the living model as well as the team's process.

The most important change is conceptual. Agile helps teams learn through working software. SDSE lets teams begin learning before working software exists and preserve that learning across the lifecycle. The combination supports earlier discovery, more credible commitments, stronger testing, safer delivery, and faster adaptation when production evidence contradicts the model. It is especially valuable when behavior crosses services, teams, partners, policies, or autonomous agents, or when discovering an error late would be expensive.

Core conclusion

SDSE makes Agile more evidence-driven without making it less adaptive. It moves selected learning earlier while retaining iterative delivery, customer collaboration, working software, and continuous improvement.

Contents

Executive Summary

- 1. Agile Solved an Important Problem**
- 2. The Behavioral Gap in Agile Delivery**
- 3. What SDSE Adds to Agile**
- 4. How SDSE Affects Agile Values and Principles**
- 5. How SDSE Changes Product Discovery**
- 6. How SDSE Changes the Product Backlog**
- 7. How SDSE Changes User Stories and Acceptance Criteria**
- 8. How SDSE Affects Sprint and Iteration Planning**
- 9. An SDSE-Enhanced Definition of Ready**
- 10. An SDSE-Enhanced Definition of Done**
- 11. How SDSE Affects Agile Engineering Practices**
- 12. How SDSE Changes Agile Events and Activities**
- 13. How SDSE Changes Agile Roles**
- 14. SDSE in Kanban, Flow-Based, and Continuous Delivery**
- 15. SDSE in Scaled Agile and Multi-Team Delivery**
- 16. SDSE and AI-Native Agile Delivery**
- 17. What SDSE Does Not Change About Agile**
- 18. Measuring the Effect of SDSE on Agile Performance**
- 19. Practical Adoption Model for Agile Teams**
- 20. Risks, Misconceptions, and Safeguards**
- 21. The Integrated Agile–SDSE Model**

Conclusion

Appendices and References

1. Agile Solved an Important Problem

The Agile Manifesto emerged as a response to development systems in which plans, requirements, documentation, contracts, and organizational boundaries often received more attention than customer collaboration and working software. Its four value statements emphasize individuals and interactions, working software, customer collaboration, and responding to change [1]. Its principles add early and continuous delivery, frequent working software, sustainable pace, technical excellence, simplicity, self-organizing teams, and regular reflection [2].

1.1 The conditions that made Agile necessary

Large requirements batches were approved before teams could test their assumptions. Feedback arrived late. Integration accumulated risk. Customers saw the result after substantial investment. Changing direction threatened plans and contracts. Development, testing, security, operations, and business stakeholders often interacted through documents and handoffs rather than through a shared delivery system.

1.2 What Agile changed

1. Working software became the primary measure of progress.
2. Delivery moved toward smaller increments and shorter feedback cycles.
3. Customers and business stakeholders collaborated more continuously with delivery teams.
4. Plans became adaptive rather than fixed predictions of distant work.
5. Cross-functional teams gained greater ownership of how outcomes were achieved.
6. Inspection, adaptation, and reflection became recurring operating practices.

1.3 The success of Agile changes the next problem

Agile made it possible to learn sooner from implemented increments. Modern engineering has compressed that loop further through cloud platforms, continuous delivery, infrastructure automation, service reuse, and AI-assisted coding. The next bottleneck is increasingly behavioral understanding: can the organization clarify and validate what a complex system should do before rapid implementation gives uncertain assumptions the appearance of progress?

2. The Behavioral Gap in Agile Delivery

Agile is not defective because teams sometimes discover requirements and design through implementation. That discovery is often intentional and productive. The problem arises when the behavior is consequential, the cost of implementation is high, or the resulting commitment is difficult to reverse. In those situations, waiting for code to become the first executable expression of intent can be unnecessarily expensive.

2.1 Working software is powerful evidence—but sometimes late evidence

A working increment demonstrates actual implementation behavior. It is stronger evidence than a document or presentation. Yet the team must first choose an architecture, write or generate code, create test environments, integrate components, and resolve enough dependencies to observe the result. If the disagreement concerns the intended behavior itself, much of that work may need to be changed.

2.2 User stories compress complex behavior

A concise story supports conversation, but it can conceal states, participants, timing, ordering, permissions, data obligations, exception paths, recovery, and competing rules. “As a customer, I want automatic approval so that I receive a faster answer” leaves important questions unanswered. Which customers are eligible? Which evidence is sufficient? What happens if data changes during review? When must a human intervene? How may a customer appeal? What is preserved after a partial failure?

2.3 Acceptance criteria may be illustrative rather than sufficient

Examples are useful, but happy-path criteria can create false confidence. Important behavior may emerge only after retries, duplicate events, timeouts, concurrency, degraded dependencies, conflicting policies, misuse, or recovery. Criteria can also be unobservable, contradictory, or dependent on state that nobody has defined. SDSE makes selected criteria executable so those weaknesses appear before the team relies on them.

2.4 Implementation-centered discovery creates rework

When a behavioral misunderstanding is discovered late, the correction may touch product intent, architecture, code, tests, controls, data, documentation, deployment, observability, and stakeholder commitments at once. The team may still adapt, but its iteration is correcting an avoidable misunderstanding instead of advancing the product. The issue is not change itself; it is paying implementation cost to discover something that could have been learned earlier.

Dimension	Typical Agile practice	Agile strengthened by SDSE
Primary executable artifact	Implemented increment	Behavior model before implementation; increment afterward
Behavioral feedback	After enough software is built and integrated	During discovery and throughout delivery
Ambiguity	Resolved through conversation and implementation	Also exposed through scenarios, traces, and counterexamples
Acceptance	Narrative examples and implementation tests	Executable expectations plus implementation conformance
Change	Backlog and code revised	Baseline, affected scenarios, backlog, and implementation revised
Learning continuity	May remain across separate tools and ceremonies	Preserved in a living behavior model and evidence history

Table 1. SDSE adds earlier executable evidence without removing implementation feedback.

3. What SDSE Adds to Agile

SDSE is a behavior-first engineering discipline. It is neither a replacement lifecycle nor a demand for complete specification. It provides a targeted loop for making consequential behavior executable before substantial implementation commitment. Teams choose fidelity and coverage according to consequence, uncertainty, interaction complexity, autonomy, reversibility, and the cost of discovering an error later.

3.1 The SDSE behavioral evidence loop

1. **Establish intent.** State the customer, business, operational, and governance outcome and the consequence of being wrong.
2. **Create a Candidate Specification.** Describe participants, responsibilities, constraints, assumptions, expectations, and unresolved questions without pretending they are final.
3. **Express consequential behavior.** Use BSN to model selected interactions, states, decisions, guards, timing, expectations, failure, and recovery.
4. **Simulate meaningful scenarios.** Execute normal, boundary, failure, misuse, adversarial, degraded, concurrent, and recovery situations according to risk.
5. **Review evidence.** Inspect traces, outcomes, counterexamples, expectation results, coverage, ambiguity, and model limitations.
6. **Revise the proposal.** Change the specification, model, product choice, architecture concept, control, or assumption while alternatives remain affordable.

7. **Approve a baseline.** Version the validated promise, evidence, scope, owners, approvals, limitations, and residual risk.
8. **Maintain continuity.** Connect the baseline to backlog, architecture, code, tests, delivery controls, telemetry, incidents, and learning.

3.2 The new feedback relationship

Agile commonly creates feedback through an implemented increment: plan a small step, build it, inspect the result, and adapt. SDSE adds a nested loop in which a team can express proposed behavior, execute it, inspect evidence, and adapt before implementation. The two loops reinforce one another. Simulation improves the decision to build; working software and production improve the model.

3.3 The living behavior model

The behavior model is not discarded when implementation begins. It becomes a persistent reference for important promises, scenarios, expectations, decisions, and limits. Teams can compare implementation and operational evidence with the baseline, identify material divergence, and revise the model when reality teaches something new. A named Behavior Model Owner maintains continuity, while product, engineering, security, quality, operations, and decision authorities retain their responsibilities.

3.4 BSN and the BSN Workbench

Behavior Simulation Notation (BSN) supplies concepts for participants, links, messages, state, logic, decisions, expectations, timing, failure, and recovery. The BSN Workbench supports authoring, execution, traces, comparison, revision, and versioning. Its purpose is not to make Agile tool-centric. It makes important behavior observable and discussable before code becomes the only executable source of truth.

4. How SDSE Affects Agile Values and Principles

SDSE should be judged by whether it supports Agile values in practice. Used correctly, it strengthens collaboration, working outcomes, customer learning, and response to change. Used as a mandatory documentation gate, it can undermine them. The method therefore depends on proportionality, real decisions, cross-functional participation, and short feedback cycles.

Agile value or principle	How SDSE can strengthen it
Individuals and interactions	Stakeholders collaborate around executable behavior and traces. The model supports conversation; it does not replace judgment.
Working software	Working software remains essential. SDSE adds behavioral execution before software and conformance evidence afterward.
Customer collaboration	Customers and domain experts can challenge proposed outcomes, exceptions, and recovery through scenarios.
Responding to change	A versioned baseline reveals which scenarios, decisions, backlog items, tests, controls, and telemetry are affected by change.
Early and continuous delivery	Earlier clarification can make small releases safer and reduce corrective work after delivery.
Working software as progress	Progress includes working software; consequential readiness also requires evidence that the behavior deserves implementation and release.
Sustainable development	Earlier discovery can reduce preventable emergencies, clarification churn, and late behavioral rework.
Technical excellence	Architecture, testing, security, reliability, and observability are connected to explicit behavioral expectations.
Simplicity	Simulation can reveal that a proposed feature or control is unnecessary, allowing the team to avoid work.
Reflection and adaptation	Retrospectives and operations improve the behavior model as well as the delivery process.

Table 2. SDSE is compatible with Agile when evidence serves collaboration and adaptation.

Boundary to preserve

The model must remain subordinate to the product decision and the people responsible for it. A model created after commitments are made, maintained by isolated specialists, or measured by volume is not evidence-driven agility.

5. How SDSE Changes Product Discovery

Agile product discovery asks whether a problem matters, whether a proposed solution is desirable and usable, and whether the organization can plausibly deliver it. SDSE adds behavioral discovery: under

which conditions will the proposed system produce the intended outcome, what interactions may prevent it, and which consequences make further evidence necessary?

5.1 From solution hypotheses to behavioral hypotheses

A behavioral hypothesis names an outcome, participants, relevant conditions, constraints, unacceptable outcomes, and evidence. It may state: when an eligible customer requests approval and the required evidence is available, the system should provide a timely decision, preserve an appeal path, prevent unauthorized override, and recover safely from dependency failure. The hypothesis remains revisable until evidence justifies a decision.

5.2 Simulation as an Agile experiment

A simulation can be a small, time-boxed experiment. Its purpose is not to complete a model; it is to answer a question that could change the product decision. A team might compare two approval policies, test a recovery rule, examine an AI agent's authority, or determine whether a promised experience remains possible when a partner is unavailable.

5.3 Behavioral discovery questions

9. What happens before and after the visible customer interaction?
10. Which participants, policies, and systems influence the outcome?
11. Which states and event orderings change the result?
12. What happens during timeout, retry, duplicate, partial failure, and recovery?
13. Who may approve, override, appeal, or stop the behavior?
14. Which outcomes are unacceptable even if the feature otherwise creates value?

5.4 Evidence-based product choices

Simulation evidence can cause the product team to accept the proposal, revise it, narrow its scope, add a control, run another experiment, observe a limited release, or reject the idea. Rejecting an attractive concept before implementing it can be a successful outcome if the evidence shows that the behavior is unsafe, economically weak, contradictory, or incompatible with the product promise.

6. How SDSE Changes the Product Backlog

The Product Backlog remains a dynamic, ordered source of product work in Scrum [3]. SDSE does not require every item to contain a complete specification. It changes the supporting evidence for consequential items and makes behavioral learning visible as legitimate work.

6.1 From feature inventory to behavioral commitments

A consequential backlog item can reference the relevant behavioral promise, Candidate Specification, scenarios, executable expectations, baseline version, open questions, and decision record. The item stays concise while the evidence remains accessible. This reduces duplicated prose and interpretation drift.

6.2 New kinds of backlog work

Backlog work type	Outcome
Behavioral discovery	Resolve a question that could change the product or delivery decision.
Candidate Specification	Make participants, assumptions, constraints, and expectations explicit enough to simulate.
Scenario design	Create normal and challenging situations tied to consequence and uncertainty.
Model authoring	Express selected behavior in BSN at the required fidelity.
Simulation and analysis	Execute scenarios, inspect traces, record findings, and revise.
Baseline decision	Review evidence and accept, revise, reject, restrict, defer, or observe.
Implementation conformance	Connect code and tests to the approved behavioral promise.
Operational learning	Use telemetry, incidents, customer outcomes, and cost to update the model.

Table 3. Behavioral evidence work belongs in the same product system as delivery work.

6.3 Refinement becomes behavioral clarification

Refinement is an opportunity to identify missing participants, undefined state, conflicting rules, hidden authority, nonobservable criteria, cross-team behavior, and recovery obligations. The team does not need to answer every question before implementation. It does need to identify which unanswered questions are likely to become expensive coding decisions.

6.4 Prioritize learning when it changes commitment

A simulation experiment may be the highest-value next backlog item even though it delivers no visible feature. If it can prevent a poor commitment, reveal a simpler solution, reduce a major dependency, or establish that the idea should stop, it creates product value through decision quality.

7. How SDSE Changes User Stories and Acceptance Criteria

User stories remain useful as invitations to conversation. SDSE does not turn every story into a formal model. It provides a way to deepen the conversation when the behavior is consequential, interaction-dependent, uncertain, or costly to discover late.

7.1 From narrative examples to executable expectations

Important criteria can be connected to a scenario with known starting conditions, events, state transitions, decisions, and expected outcomes. The team can see whether the expectation is observable, whether another rule contradicts it, and whether failure or recovery changes the outcome. The same expectation can later guide implementation tests and telemetry.

7.2 A behavior-centered story conversation

15. Which promise does this story advance?
16. Which participants and states influence the result?
17. What must be true before the behavior begins?
18. What outcome is expected, prohibited, or recoverable?
19. Which scenario would expose a serious misunderstanding?
20. Which material question may remain for implementation?

7.3 Traceability without excessive documentation

A story can reference the applicable baseline and scenario rather than restating the complete behavior. The goal is not a chain of documents. It is a navigable relationship among the product promise, the executable evidence, the work, the implementation, and the operational result.

8. How SDSE Affects Sprint and Iteration Planning

Iteration planning turns prioritized intent into a near-term goal and a feasible plan. SDSE improves planning when important behavior has been clarified enough to expose the real work. It also gives the team permission to plan learning when detailed implementation planning would communicate precision without understanding.

8.1 Planning with better information

Simulation may reveal states, interfaces, controls, partner behavior, recovery paths, observability, human review, data obligations, and tests that the feature description concealed. This can increase the estimate. It can also make the estimate more credible and prevent a commitment built on invisible work.

8.2 Separate learning commitments from delivery commitments

A team may commit to answering a behavioral question within an iteration before it commits to implementing the full solution. The result of the iteration may be an accepted baseline, a revised concept, a restricted scope, or a justified stop. This is empirical product development, not analysis without delivery.

8.3 Plan modeling proportionally

Planning factor	Question
Consequence	What happens if the behavior is wrong, unavailable, unsafe, abused, or unrecoverable?
Uncertainty	Which important states, interactions, decisions, or expectations remain unclear?
Interaction complexity	How many services, teams, partners, policies, or autonomous participants influence the result?
Autonomy	Can software or an AI agent take consequential action without immediate human approval?
Reversibility	Can the decision be changed cheaply after release?
Cost of late discovery	Which implementation, integration, customer, contract, or control commitments would need revision?

Table 4. Teams can scale behavioral evidence to the decision rather than model every story.

9. An SDSE-Enhanced Definition of Ready

The Scrum Guide does not prescribe a Definition of Ready. Many teams use one as a local working agreement. An SDSE-enhanced version should help the team decide whether an item is understood enough for the intended commitment without becoming a rigid gate that blocks discovery.

Readiness element	SDSE-enhanced working agreement
Outcome	The customer or business outcome is clear enough for the intended commitment.
Participants	Consequential users, services, partners, agents, policies, and authorities are visible.
Conditions and state	Important starting conditions, state, timing, and ordering assumptions are explicit.
Constraints	Security, privacy, safety, compliance, operational, and commercial limits are known.
Unacceptable outcomes	The team knows what must not happen and why.
Scenarios	Required normal and challenging situations are proportional to risk.
Expectations	Important outcomes are observable and suitable for simulation or later testing.
Open questions	Material uncertainty has an owner and will not silently become a coding decision.
Baseline	The applicable behavioral baseline and material-change threshold are identified.
Decision	The team knows what evidence can cause it to accept, revise, reject, restrict, defer, or observe.

Table 5. A Definition of Ready should improve conversation, not certify perfect knowledge.

Important qualification

“Ready” means ready for the next responsible commitment. It does not mean complete, final, or immune to change.

10. An SDSE-Enhanced Definition of Done

The Scrum Guide defines the Definition of Done as a formal description of the state of the Increment when required quality measures are met [3]. SDSE can strengthen that transparency by connecting completion to the approved behavioral promise and the evidence required to operate it responsibly.

Completion element	SDSE-enhanced evidence
Conformance	Implementation and tests conform to the applicable behavioral baseline for the agreed scope.
Material deviation	Differences discovered during implementation are reflected in the model and approved.
Scenario evidence	Required normal, boundary, failure, misuse, degraded, and recovery scenarios have appropriate evidence.
Operational visibility	Telemetry can observe critical expectations and unacceptable outcomes.
Release behavior	Deployment, migration, rollback, retry, and recovery behavior meet the required expectations.

Completion element	SDSE-enhanced evidence
Authority	Human review, override, appeal, safe stop, and delegated permissions work as intended where applicable.
Residual risk	Known limitations and residual risk are recorded and owned.
Learning	The condition that will trigger review, model revision, or resimulation is identified.

Table 6. Done connects implemented quality with the validated behavioral promise.

10.1 Do not turn Done into an impossible assurance claim

No simulation or test set proves that production will always behave correctly. The Definition of Done should identify the evidence required for the release scope, not promise universal correctness. Operational protection, controlled rollout, telemetry, recovery, and learning remain part of the system.

11. How SDSE Affects Agile Engineering Practices

SDSE does not replace architecture, development, testing, security, DevOps, SRE, or platform engineering. It supplies a shared behavioral reference that those practices can use and challenge. The value comes from continuity: the promise examined during discovery remains visible when engineers make implementation and operational decisions.

Practice	Effect of SDSE
Architecture	Compare responsibilities, interactions, failure propagation, recovery, and control alternatives against scenarios.
Development	Implement from a validated behavioral reference instead of independently interpreting fragmented requirements.
Testing	Derive focused tests from consequential scenarios and executable expectations while continuing exploratory and implementation-specific testing.
Security	Simulate misuse, adversarial sequences, permission boundaries, control failure, escalation, and recovery.
DevOps and delivery	Connect pipeline evidence, release policy, rollout, rollback, and deployment behavior to the baseline.
SRE and operations	Compare incidents, telemetry, degradation, recovery, and customer impact with expected behavior.
Platform engineering	Offer reusable behavior patterns, scenario libraries, evidence integrations, and behavioral golden paths.

Table 7. The living behavior model creates continuity across Agile engineering practices.

11.1 Test-first and simulation-first are complementary

Test-driven development helps engineers shape implementation through fast executable feedback. SDSE operates at a different level: it challenges whether the selected system behavior deserves implementation and clarifies cross-participant expectations. A team may simulate consequential behavior, derive tests from the baseline, and still use test-first practices for implementation detail.

11.2 Secure development remains integrated

NIST's Secure Software Development Framework recommends high-level practices that can be integrated into each software lifecycle implementation [6]. SDSE provides an additional means to express and evaluate selected security behavior, but secure design, code review, dependency management, testing, vulnerability response, provenance, and operational controls remain necessary.

12. How SDSE Changes Agile Events and Activities

SDSE should enter existing Agile conversations where it improves a decision. It should not create a parallel calendar of ceremonies unless the risk and scale justify dedicated review. The lightest useful integration is preferable.

Agile activity	SDSE contribution
Product discovery	Add executable behavioral hypotheses and simulation experiments.
Backlog refinement	Expose assumptions, states, interactions, dependencies, exceptions, and evidence needs.
Sprint or iteration planning	Plan delivery from clearer behavior or explicitly plan a bounded learning outcome.
Daily coordination	Make behavioral blockers, model changes, and evidence needs visible.
Development	Use the baseline to guide implementation and identify material deviation.
Review or demonstration	Show the working increment and its relationship to the approved behavioral promise.
Retrospective	Improve the team's evidence process and identify model or scenario gaps revealed during delivery.
Release review	Confirm implementation conformance, operational protection, and residual risk.
Incident review	Update the living model with production behavior and resimulate affected scenarios.

Table 8. SDSE strengthens existing feedback points instead of multiplying ceremonies.

12.1 Sprint Review becomes a behavioral conversation

The Sprint Review can examine not only what was implemented but whether the increment advances the Product Goal and behavioral promise. Stakeholders can see important scenario results, material

deviations, residual uncertainty, and operational evidence. The purpose remains inspection and adaptation, not approval theater.

12.2 Retrospectives extend into the model

If implementation exposed a recurring ambiguity, if tests found an unmodeled interaction, or if release required an unexpected recovery path, the team can improve the Candidate Specification template, model pattern, scenario library, or readiness policy. The learning becomes reusable rather than remaining only in meeting notes.

13. How SDSE Changes Agile Roles

13.1 Product Manager

Conventional emphasis. Frames customer problems, product strategy, outcomes, value propositions, roadmap choices, and priorities.

SDSE-enhanced emphasis. Frames consequential capabilities as behavioral promises and uses simulation evidence during discovery, prioritization, commitment, and operational learning.

- 21. Selects behavioral questions that can change the product decision.
- 22. Defines expected and unacceptable outcomes.
- 23. Distinguishes exploratory, behaviorally validated, committed, and operationally verified roadmap states.
- 24. Uses operations to revise the promise.

Primary benefit. Earlier invalidation of weak concepts and more honest product commitments.

13.2 Product Owner

Conventional emphasis. Orders Product Backlog work, clarifies items, collaborates with stakeholders, and maximizes value.

SDSE-enhanced emphasis. Maintains links among backlog items, behavioral promises, scenarios, expectations, baseline versions, and open decisions.

- 25. Separates behavioral learning from implementation work.
- 26. Makes material behavioral change visible.
- 27. Brings evidence and uncertainty into refinement and review.
- 28. Prevents acceptance criteria from drifting across disciplines.

Primary benefit. Clearer backlog conversations and less interpretation churn during delivery.

13.3 Scrum Master or Agile Coach

Conventional emphasis. Improves team effectiveness, facilitates empiricism, removes impediments, and supports Agile practice.

SDSE-enhanced emphasis. Helps the team integrate simulation-driven learning without creating a specialist gate or process burden.

- 29. Protects time for consequential discovery.
- 30. Challenges model-count and ceremony-driven adoption.
- 31. Facilitates cross-functional evidence reviews.
- 32. Helps teams tune proportionality and feedback flow.

Primary benefit. SDSE becomes an Agile learning capability rather than a documentation process.

13.4 Software Engineer

Conventional emphasis. Designs, builds, tests, integrates, and improves software increments.

SDSE-enhanced emphasis. Participates earlier in clarifying intent, modeling behavior, designing scenarios, analyzing traces, and maintaining conformance.

- 33. Challenges incomplete state and interaction assumptions.
- 34. Uses the baseline to guide design and code.
- 35. Reports material deviation instead of silently redefining behavior.
- 36. Feeds implementation learning back into the model.

Primary benefit. Less speculative implementation and greater system-level ownership.

13.5 Quality Engineer

Conventional emphasis. Evaluates implementation quality through test strategy, automation, exploration, and evidence.

SDSE-enhanced emphasis. Challenges behavioral intent before code and connects implementation tests to meaningful scenarios and expectations.

- 37. Designs boundary, failure, misuse, and recovery scenarios.
- 38. Examines observability and oracle quality.
- 39. Compares model and implementation behavior.

40. Uses production findings to improve coverage.

Primary benefit. Earlier quality influence and stronger evidence relevance.

13.6 Architect, Security, and Reliability Practitioners

Conventional emphasis. Guide structure, controls, resilience, operational quality, and cross-system decisions.

SDSE-enhanced emphasis. Use simulated behavior to evaluate interactions, authority, failure propagation, recovery, and operational limits.

41. Compare architecture alternatives through scenarios.

42. Model adversarial and degraded behavior.

43. Define rollout, rollback, safe-stop, and recovery expectations.

44. Connect telemetry and incidents to the baseline.

Primary benefit. Earlier discovery of systemic risk and better lifecycle continuity.

13.7 Behavior Model Owner

Conventional emphasis. A new or assigned accountability for preserving the integrity and continuity of the behavior model.

SDSE-enhanced emphasis. Maintains the BSN model, scenario catalogue, evidence, versions, decisions, limitations, and links across the lifecycle.

45. Coordinates changes without absorbing product or engineering authority.

46. Ensures the applicable baseline is identifiable.

47. Facilitates review and resimulation after material change.

48. Prevents the model from becoming stale documentation.

Primary benefit. A living single source of behavioral truth with distributed decision accountability.

14. SDSE in Kanban, Flow-Based, and Continuous Delivery

SDSE is not limited to Scrum. It can operate within Kanban, Extreme Programming, Lean product development, continuous delivery, Shape Up, scaled approaches, and hybrid systems. The integration point is the decision and its evidence, not a prescribed iteration structure.

14.1 Behavioral evidence within flow

A flow-based team can make behavioral states visible without creating a separate value stream. Possible states include behavioral question identified, Candidate Specification ready, simulation in progress, evidence under review, behavior validated, ready for implementation, conformance verified, and operational evidence collected. The team should use only the states that improve coordination.

14.2 Work-in-progress effects

Early behavioral discovery can reduce implementation work in progress by preventing teams from starting poorly understood work. Modeling itself can also become excess work in progress if teams begin too many investigations or seek comprehensive models. Limit behavioral work to active decisions and set explicit exit conditions.

14.3 Continuous delivery

A validated baseline can inform automated tests, policy checks, rollout constraints, human approvals, observability, and rollback. The goal is not to turn every deployment into a manual model review. Stable scenarios can become automated evidence; material behavioral change should trigger focused review and resimulation.

15. SDSE in Scaled Agile and Multi-Team Delivery

Multi-team systems fail at boundaries that local backlogs and component tests may not represent. Shared state, event ordering, identity, policy, retries, asynchronous processing, data ownership, partial failure, and recovery create behavior that belongs to no single team. SDSE provides a way to make those interactions visible before integration.

15.1 Cross-team behavioral dependencies

- 49. An event must occur only after authorization, not merely eventually.
- 50. A downstream action must be idempotent when an upstream participant retries.
- 51. A human override must remain available during partial platform failure.
- 52. Two independent policies must not produce contradictory actions.
- 53. Recovery must preserve the business state promised to the customer.

15.2 Shared baselines without centralized design

A shared behavioral baseline can define cross-boundary promises while allowing teams to choose their internal design. Program or product leadership can review common scenarios and residual risk without prescribing every implementation decision. This is coordination through behavior, not command-and-control architecture.

15.3 Reusable behavioral assets

Platform and enablement teams can provide reusable BSN patterns, scenario libraries, failure models, security expectations, recovery patterns, and evidence integrations. Reuse allows product teams to obtain better evidence with less modeling effort and creates consistency across a portfolio.

16. SDSE and AI-Native Agile Delivery

AI-assisted engineering can generate code, tests, documentation, models, and design alternatives at extraordinary speed. AI-native products may also contain probabilistic models, agents, tool use, delegated authority, and behavior that changes with data or environment. These capabilities increase the value of Agile feedback—and increase the danger of accelerating ambiguous intent.

16.1 AI amplifies the delivery system

Google Cloud's 2025 DORA research characterizes AI as an amplifier: strong organizational capabilities can be magnified, and weak systems can be amplified as well [4]. If the product promise is clear and feedback is strong, AI can accelerate useful work. If requirements are ambiguous, tests are weak, or authority is undefined, AI can accelerate the wrong behavior into production.

16.2 The Validated Behavioral Baseline as a control reference

The baseline gives engineers and coding agents a reference for what behavior is authorized, which scenarios matter, which outcomes are unacceptable, and where human authority applies. AI can assist with model alternatives, scenario proposals, implementation, test generation, evaluation, and trace analysis. It does not own the promise or accept its consequences.

16.3 Human-in-Control responsibilities

- 54. Define intended outcomes and unacceptable consequences.
- 55. Set permission, escalation, override, appeal, and safe-stop boundaries.
- 56. Approve the Validated Behavioral Baseline and material changes.

57. Judge whether evidence is sufficient for release or delegated authority.

58. Accept residual risk and define operational protection.

59. Revise the model when production reality contradicts assumptions.

NIST's AI Risk Management Framework emphasizes governance and management of AI risks [5], and its SSDF profile for generative AI extends secure development practices to AI model development [7]. SDSE complements such guidance by making selected authority, escalation, failure, and recovery behavior executable.

AI-native Agile rule

Generation speed is not decision authority. AI can produce artifacts and evidence; accountable humans decide which behavior deserves implementation, release, and delegated action.

17. What SDSE Does Not Change About Agile

SDSE should not be presented as a correction that makes Agile obsolete. It addresses a specific evidence gap. The foundational disciplines of Agile remain necessary because no model can replace customer contact, team judgment, implementation reality, or operational learning.

Agile foundation	Why it remains essential
Customer collaboration	Simulation improves the conversation but cannot define value without stakeholders.
Working software	The implementation remains the product and must be evaluated directly.
Iterative delivery	Baselines are versioned and revisable; teams continue to deliver and learn in small increments.
Cross-functional teams	Behavioral evidence requires domain, product, engineering, quality, security, and operational perspectives.
Technical excellence	Models do not replace good architecture, code, tests, controls, automation, or operations.
Continuous integration and delivery	The baseline strengthens evidence in the pipeline; it does not replace delivery automation.
Retrospectives	Teams still inspect how they work and adapt their practices.
Human judgment	People decide model scope, evidence sufficiency, tradeoffs, authority, and residual risk.

Table 9. SDSE adds a behavioral evidence layer while preserving Agile foundations.

18. Measuring the Effect of SDSE on Agile Performance

SDSE should be measured by whether it improves decisions, delivery, outcomes, and learning—not by the number of models or scenarios. A balanced evaluation combines leading indicators of earlier

discovery with downstream evidence about rework, flow, quality, operational performance, and product outcomes.

Dimension	Possible measure	Evidence source
Behavioral clarity	Material questions, contradictions, and missing expectations resolved before implementation	Refinement and decision records
Decision latency	Time from consequential question to accept, revise, reject, restrict, defer, or observe	Workflow timestamps
Planning quality	Estimate change after clarification; dependency discoveries; forecast stability	Planning history
Flow	Blocked implementation, work-item age, and time lost to behavioral clarification during delivery	Delivery system
Early discovery	Unsafe interactions, missing controls, and recovery gaps found before code or release	Simulation findings
Rework	Behavior-related redesign, recoding, retesting, integration correction, and release delay	Change and time records
Quality	Acceptance failures and escaped behavioral defects	Test and defect systems
Delivery	Behavior-related change failures, rollback, and recovery effort	Pipeline and incident systems
Learning	Time from operational observation to model revision, resimulation, and approved response	Model and release history
Product outcome	Customer, business, operational, or policy result associated with the validated promise	Product analytics

Table 10. Measurement should connect behavioral evidence to Agile and business outcomes.

18.1 Measurement safeguards

- 60. Establish the baseline and comparison period before the pilot begins.
- 61. Include training, facilitation, modeling, review, maintenance, and delay in the cost.
- 62. Record other process, architecture, staffing, tooling, and market changes that could explain results.
- 63. Do not treat every counterexample as a production incident that was certainly prevented.
- 64. Report unfavorable evidence and stop modeling that does not affect a worthwhile decision.
- 65. Never use model or scenario quotas as individual performance measures.

19. Practical Adoption Model for Agile Teams

The best first adoption is one consequential backlog item and one decision. The pilot should be important enough to matter, bounded enough to finish, and measurable enough to reveal whether SDSE improves the team's work. A broad process mandate before teams experience the value is likely to create compliance rather than learning.

1. **Select a consequential item.** Choose behavior with meaningful uncertainty, interaction complexity, autonomy, or expensive late discovery.

2. **Define the decision.** State what evidence can cause the team to accept, revise, reject, restrict, defer, or observe.
3. **Create a small Candidate Specification.** Capture participants, conditions, constraints, expectations, assumptions, and open questions.
4. **Set proportional scope.** Model only the behavior required to support the decision.
5. **Simulate meaningful scenarios.** Include normal and challenging situations according to consequence and uncertainty.
6. **Review and decide.** Approve a baseline or change the proposed behavior, scope, control, or product choice.
7. **Connect the Agile work.** Link the baseline to backlog items, architecture, code, tests, delivery controls, and telemetry.
8. **Measure the result.** Compare clarification, planning, flow, rework, quality, delivery, and outcome evidence with the prior baseline.
9. **Scale through reuse.** Develop coaching, patterns, scenario libraries, integrations, and proportionality policies.

19.1 Characteristics of a good pilot

10. A real product decision and accountable sponsor.
11. Observable behavioral uncertainty that current conversations have not resolved.
12. Cross-functional participation, including product, engineering, quality, security, or operations as relevant.
13. A scenario scope small enough to complete within an Agile cadence.
14. Evidence that can genuinely change the decision.
15. A baseline and measures for evaluating the result.

19.2 Adoption at scale

Scale should come from reusable patterns, scenario libraries, shared model elements, evidence integrations, training, and clear policies for proportionality. A center of enablement can coach teams and preserve standards without becoming an approval bottleneck or the only group capable of modeling.

20. Risks, Misconceptions, and Safeguards

Misconception	Practical safeguard
---------------	---------------------

Misconception	Practical safeguard
"SDSE is mini-waterfall."	Keep models small, iterative, revisable, and tied to a current decision.
"Simulation requires complete specifications."	Use Candidate Specifications that expose uncertainty rather than hide it.
"Every story must be modeled."	Apply SDSE according to consequence, uncertainty, interaction complexity, autonomy, and late-discovery cost.
"The baseline freezes change."	Treat it as a versioned reference that supports controlled adaptation.
"Simulation replaces working software."	Continue to test implementation and production behavior directly.
"A successful simulation proves correctness."	Document model scope, fidelity, assumptions, exclusions, and residual risk.
"The model owner owns product behavior."	Preserve distributed product, engineering, assurance, operational, and risk accountability.
"More scenarios always mean more confidence."	Target high-value uncertainty; remove redundant or decision-irrelevant scenarios.
"SDSE will reduce velocity."	Measure total flow, rework, failure, and outcomes—not story completion alone.
"AI can approve the behavior it generated."	Require accountable human authority for intent, baseline approval, material change, and residual risk.

Table 11. SDSE remains Agile when it is proportional, decision-centered, and continuously revised.

20.1 Model risk

A model can be incomplete, incorrect, stale, or misleadingly precise. Teams should document scope, fidelity, assumptions, exclusions, validation, and divergence from production. Important evidence should be independently challenged, and material operational differences should trigger review.

20.2 Process and cultural risk

SDSE can become bureaucracy when leaders mandate modeling without naming a decision, when specialists work after commitments are fixed, or when discovery of contradiction is treated as failure. Simulation is valuable precisely because it exposes misunderstanding. Leaders must reward early counterexamples and honest revision.

21. The Integrated Agile–SDSE Model

AGILE WITH AN SDSE BEHAVIORAL EVIDENCE LOOP

SDSE adds executable learning before implementation and preserves the behavioral promise through operations.

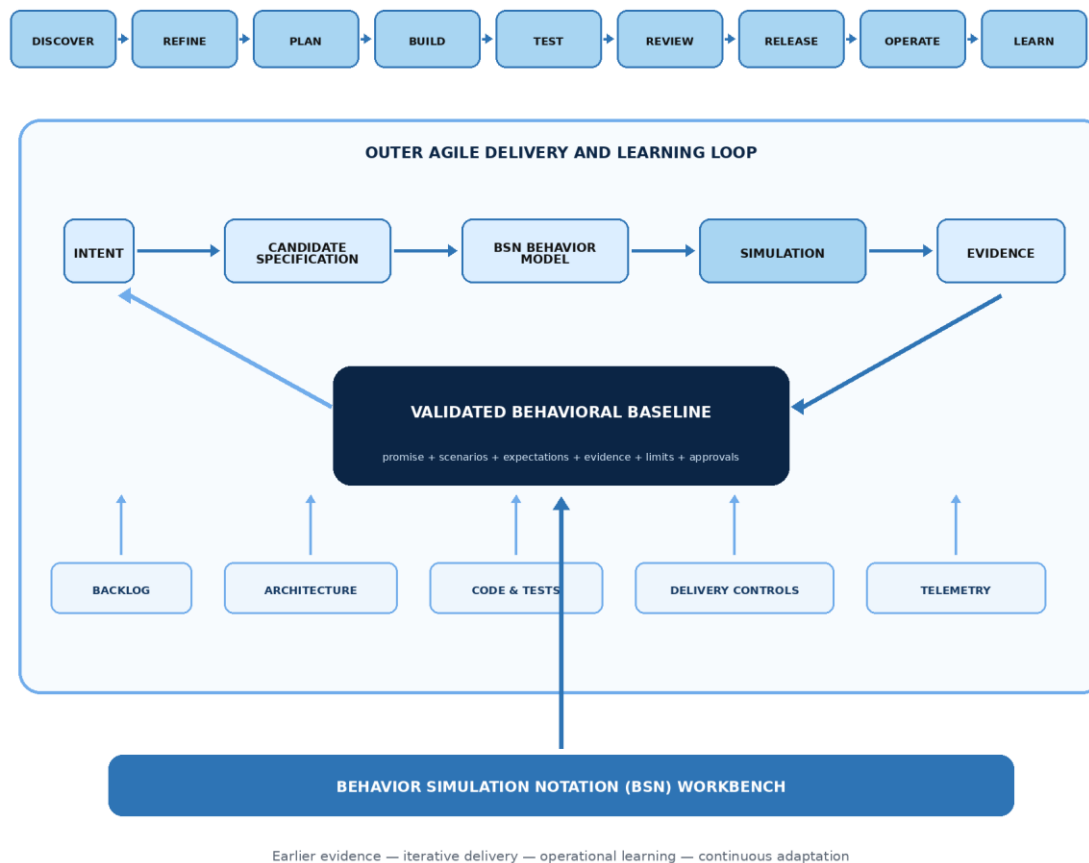


Figure 1. SDSE adds an earlier executable feedback loop and preserves behavioral learning across Agile delivery and operations.

The model shows one system of work with two connected learning loops. The inner loop challenges proposed behavior before substantial implementation. The outer loop produces and operates working software, creating stronger evidence about the actual system and product outcome. The Validated Behavioral Baseline connects the loops. It informs backlog, architecture, code, tests, delivery controls, and telemetry; operational learning then changes the model when reality contradicts the promise or its assumptions.

Conclusion

Agile changed software engineering by reducing the distance between implementation and feedback. SDSE reduces the distance between behavioral intent and executable evidence. The combination allows teams to learn twice: first from a proposed behavior while change is inexpensive, and then from working software and operations where reality supplies evidence that no model can fully anticipate.

This changes discovery, backlogs, stories, acceptance criteria, planning, readiness, completion, engineering practice, events, roles, flow, and multi-team coordination. It does not require every story to be modeled or every decision to pass through a specialist gate. Teams should use the smallest model and scenario set that can change a consequential decision, then preserve the resulting promise and evidence through implementation and operations.

The need becomes more urgent in AI-native engineering. AI can generate artifacts faster than organizations can understand, validate, and govern behavior. A living behavioral baseline provides a human-approved reference for intent, limits, authority, and evidence. AI may assist throughout the loop, but people remain accountable for the behavior they authorize and the consequences they accept.

Closing statement

Agile supported by SDSE is not less adaptive. It can adapt earlier, with better evidence, before avoidable assumptions become expensive software commitments.

Appendix A. SDSE-Enhanced Backlog Refinement Checklist

Use this checklist for one consequential backlog item. Select only the questions relevant to the decision and level of risk.

Refinement topic	Team question
Outcome	Is the customer or business outcome clear enough to discuss behavior?
Promise	What observable behavior is the product expected to provide?
Participants	Which people, systems, partners, agents, policies, and data sources influence it?
Conditions	What must be true before the behavior begins?
State	Which state, transition, sequence, timing, or concurrency affects the outcome?
Constraints	Which security, privacy, safety, accessibility, regulatory, commercial, or operational limits apply?
Authority	Who may approve, override, appeal, retry, stop, or recover?
Unacceptable outcomes	What must not happen, and what consequence would follow?
Failure and recovery	What happens during delay, timeout, duplicate, partial failure, loss, and restart?
Dependencies	Which cross-team or external behaviors must remain compatible?
Expectations	Are important outcomes observable and testable?
Priority scenarios	Which normal and challenging situations provide the highest-value evidence?
Open questions	Which uncertainty could change scope, estimate, architecture, control, or release?
Decision	What may the evidence cause the team to accept, revise, reject, restrict, defer, or observe?
Baseline	Which model version applies, and what change is material?

Table A1. SDSE-enhanced backlog refinement checklist.

Appendix B. Behavioral Definition of Ready and Definition of Done Worksheet

Element	Definition of Ready prompt	Definition of Done prompt
Outcome understood	Ready: value and behavioral promise are clear enough	Done: implementation advances the intended outcome
Participants and state	Ready: consequential actors, conditions, and state are visible	Done: implementation behavior conforms for agreed scope
Constraints	Ready: important limits and unacceptable outcomes are known	Done: controls and evidence satisfy applicable limits
Scenarios	Ready: proportional scenarios are selected	Done: required scenarios have implementation evidence
Expectations	Ready: important outcomes are observable	Done: expectations pass or deviations are approved
Open uncertainty	Ready: material questions have owners	Done: residual uncertainty and risk are recorded
Authority	Ready: approval and override boundaries are identified	Done: permissions, human control, and safe stop operate
Release and recovery	Ready: consequential release behavior is understood	Done: rollout, rollback, recovery, and telemetry are ready
Baseline and learning	Ready: applicable version and change threshold are known	Done: version, evidence, and next learning trigger are recorded

Table B1. Adapt the worksheet to the team's context; do not treat it as a universal gate.

Appendix C. Agile Behavioral Evidence Canvas

Canvas field	Working answer
Product outcome	Which customer, business, operational, or policy result matters?
Behavioral promise	What observable behavior should produce that outcome?
Participants	Who and what influence the result?
Constraints	Which limits must the behavior respect?
Unacceptable outcomes	What must never happen?
Behavioral questions	Which uncertainty could change the decision?
Priority scenarios	Which situations provide the highest-value evidence?
Evidence threshold	What is sufficient to accept, revise, reject, restrict, defer, or observe?
Applicable baseline	Which version and scope apply?
Decision owner	Who has authority and who accepts residual risk?
Operational protection	Which telemetry, threshold, override, recovery, and safe-stop controls apply?
Learning trigger	Which event or evidence requires review or resimulation?

Table C1. Agile Behavioral Evidence Canvas.

Appendix D. Agile Roles and SDSE Responsibilities

Role	Primary accountability	SDSE contribution
Product manager	Value, outcome, promise, priority	Behavioral hypotheses and evidence state
Product owner	Backlog value and clarity	Scenario-linked expectations and baseline traceability
Scrum Master / coach	Team effectiveness and empiricism	Proportional integration and evidence flow
Software engineer	Design, code, tests, implementation quality	Model challenge, scenario evidence, conformance
Quality engineer	Test strategy and product evidence	Early behavioral challenge and implementation comparison
Architect	System structure and technical decisions	Interaction, failure, recovery, and alternative analysis
Security / reliability	Controls, resilience, operational protection	Adversarial, degraded, authority, and recovery scenarios
Behavior Model Owner	Model continuity and integrity	BSN model, evidence, versions, decisions, and links
Decision authority	Approval and risk acceptance	Evidence sufficiency, limits, and residual risk

Table D1. SDSE adds responsibilities without centralizing product or engineering authority.

References

Sources were accessed on August 15, 2026. They support contextual claims about Agile, Scrum, AI-assisted development, AI risk, secure development, and lifecycle economics. The SDSE integration framework is presented by the author.

- [1] Beck, Kent, et al. “Manifesto for Agile Software Development.” [Official Agile Manifesto](#)
- [2] Beck, Kent, et al. “Principles behind the Agile Manifesto.” [Official Agile principles](#)
- [3] Schwaber, Ken, and Jeff Sutherland. “The Scrum Guide,” November 2020. [Official Scrum Guide](#)
- [4] Google Cloud. “Announcing the 2025 DORA Report: State of AI-Assisted Software Development.” [Official report announcement](#)
- [5] National Institute of Standards and Technology. “Artificial Intelligence Risk Management Framework (AI RMF 1.0).” [Official NIST resource](#)
- [6] National Institute of Standards and Technology. “SP 800-218, Secure Software Development Framework (SSDF) Version 1.1.” [Official publication](#)
- [7] National Institute of Standards and Technology. “SP 800-218A, Secure Software Development Practices for Generative AI and Dual-Use Foundation Models.” [Official publication](#)
- [8] IEEE Computer Society. “Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 4.” [Official SWEBOK resource](#)

- [9] National Aeronautics and Space Administration. “Systems Engineering Handbook 2.5: Cost Effectiveness Considerations.” [Official NASA guidance](#)
- [10] 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. He is the principal author of Simulation-Driven Software Engineering: A Behavior-First Discipline for Modern Systems and creator of Behavior Simulation Notation and the BSN Workbench. His experience spans software, networks, automation, DevOps, security, reliability, AI, simulation, and engineering tools. Through PrimaModus®, he helps teams apply SDSE to consequential product and delivery decisions.

Contact: mhornbeek@engineeringdevops.com