

How Site Reliability Engineering Is Affected by SDSE

From Production Observation to Lifecycle Behavioral Confidence

The central proposition

Site Reliability Engineering measures and improves the reliability of real services. Simulation-Driven Software Engineering adds executable behavioral evidence before implementation and deployment. Together they create a continuous reliability discipline from intent through operations.

By Marc Hornbeek, a.k.a. DevOps_the_Gray

Originator of Simulation-Driven Software Engineering

August 2026

Executive Summary

Site Reliability Engineering (SRE) applies software engineering to operations so organizations can deliver dependable services while balancing reliability, change velocity, and cost. Its strongest mechanisms—service-level indicators, service-level objectives, error budgets, observability, automation, incident response, postmortems, capacity planning, and toil reduction—turn production experience into actionable engineering evidence [1][2]. SRE is not merely a production support function; Google’s SRE guidance explicitly places reliability engagement throughout the service lifecycle [3].

Even so, an important evidence gap remains. Many reliability questions become executable only after a design, implementation, deployment environment, or production service exists. By then, cross-service assumptions, degraded modes, recovery semantics, instrumentation gaps, ownership ambiguity, or unsafe automation authority may already be embedded in architecture and plans. A service can satisfy local availability or latency targets while the end-to-end customer outcome is wrong.

Simulation-Driven Software Engineering (SDSE) moves part of this evidence earlier. It begins with stakeholder intent and unacceptable outcomes, develops a Candidate Specification, represents consequential behavior in an executable model, simulates normal and adverse scenarios, and establishes a versioned Validated Behavioral Baseline. Behavior Simulation Notation (BSN) and the BSN Workbench provide a notation and working environment for representing participants, state, conditions, interactions, scenarios, expectations, traces, and revisions [10].

Core conclusion

SDSE does not replace SRE. It extends SRE upstream. SDSE validates the reliability behavior the organization intends to build; SRE measures, protects, and improves that promise in real operation.

1. SLOs can be derived from validated customer and system behavior rather than selected only after implementation.
2. Simulation scenarios can expose failure, recovery, dependency, capacity, and intervention requirements before production risk exists.
3. Model traces can define the telemetry needed to recognize state, causality, and behavioral drift.
4. Incident and operational evidence can revise the model, tests, controls, SLOs, runbooks, and implementation.
5. SREs gain earlier influence over intent, design, resilience, observability, and authority decisions.

Contents

Executive Summary

- 1. The Continuing Importance of Site Reliability Engineering**
- 2. The Reliability Problem SRE Was Not Designed to Solve Alone**
- 3. What Simulation-Driven Software Engineering Adds**
- 4. SRE and SDSE: The Fundamental Comparison**
- 5. Reliability Must Be Designed as Behavior**
- 6. From SLO Definition to SLO Simulation**
- 7. How SDSE Changes Error-Budget Management**
- 8. Engineering Failure, Degradation, and Recovery Before Deployment**
- 9. How SDSE Affects Observability Engineering**
- 10. How SDSE Changes Incident Preparedness and Response**
- 11. SDSE, Chaos Engineering, and Resilience Testing**
- 12. Reducing Operational Toil Through Behavioral Modeling**
- 13. Capacity, Performance, and Dependency Planning**
- 14. SRE for AI-Native and Agentic Systems**
- 15. The Integrated SDSE-SRE Reliability Lifecycle**
- 16. Changing the Role of the Site Reliability Engineer**
- 17. Organizational and Team Implications**
- 18. Practical Adoption Approach**
- 19. Measuring the Value of SDSE-Enabled SRE**
- 20. Leadership Implications**

Conclusion

Appendices, References, and Author

1. The Continuing Importance of Site Reliability Engineering

SRE is the disciplined application of software engineering to the design and operation of reliable services. It treats reliability as a measurable product property and an economic decision, not as a vague aspiration or an unlimited demand for uptime. The goal is dependable behavior at a level that supports user needs while preserving the organization's ability to change.

Google's SRE literature connects risk, SLOs, error budgets, monitoring, automation, release engineering, incident response, capacity, and learning into an operating system for reliability [1][2]. Error budgets are especially important because they create a shared mechanism for deciding when the organization can accept change risk and when it must invest in stability. This replaces the impossible target of zero failure with explicit, reviewable tradeoffs.

What disciplined SRE provides

6. User-centered reliability targets expressed through SLIs and SLOs.
7. An error-budget mechanism for balancing reliability and change.
8. Telemetry and alerting that support timely operational decisions.
9. Incident command, recovery, postmortem learning, and systemic improvement.
10. Automation that reduces repetitive manual work and makes safe response repeatable.
11. Capacity, load, dependency, release, and resilience engineering across the service lifecycle.

Reliability is behavior

A service is not reliable merely because it is reachable. Reliability means that it consistently produces the outcomes users and dependent systems expect under the conditions that matter.

2. The Reliability Problem SRE Was Not Designed to Solve Alone

SRE practices can and should influence systems before launch, but many operational mechanisms require an implemented subject. Telemetry requires instrumentation. Load testing requires an executable system. Chaos experiments require a running environment. Error-budget consumption requires observations over time. These sources of evidence are powerful because they are real; they are also constrained by how much has already been committed.

Some consequential reliability defects originate before those mechanisms can operate. A retry policy may conflict with payment semantics. A recovery procedure may restore infrastructure while preserving corrupt business state. A locally healthy dependency may return stale data that drives an unacceptable

decision. An AI agent may remain available while acting beyond its intended authority. These are behavioral failures that ordinary availability measures can miss.

Late evidence is still evidence—but it costs more

Production evidence is indispensable. The question is whether the organization could have exposed the same misunderstanding before architecture, code, release commitments, or customer impact made correction more expensive.

Common upstream reliability gaps

- 12. Conflicting expectations across teams, services, suppliers, and policies.
- 13. Missing degraded-mode, compensation, rollback, or recovery behavior.
- 14. SLOs that measure local technical health instead of end-to-end user outcomes.
- 15. Instrumentation selected after key causal or state information can no longer be reconstructed.
- 16. Automation without explicit authority, revocation, escalation, or human intervention behavior.
- 17. Runbooks that document familiar failures but omit novel combinations of state and dependency behavior.

3. What Simulation-Driven Software Engineering Adds

SDSE is a behavior-first engineering discipline. It creates executable evidence about a proposed system before production implementation exists. Its purpose is not to predict every possible event or model every detail. Its purpose is to represent enough consequential behavior to challenge assumptions and improve decisions while change remains affordable.

The SDSE flow begins with intent, outcomes, obligations, constraints, and unacceptable outcomes. A Candidate Specification makes the proposed behavior explicit. A BSN model then represents participants, interfaces, state, operating conditions, logic, interactions, scenarios, and expectations. Simulation produces traces and outcomes that reviewers can challenge. Once accountable stakeholders accept the evidence, assumptions, limitations, and residual uncertainty, the result becomes a versioned Validated Behavioral Baseline.

The baseline is not a static requirements document. It is a behavioral promise that should guide architecture, tests, SLOs, telemetry, controls, runbooks, release criteria, operations, and subsequent revisions. The BSN Workbench supports authoring, execution, trace review, comparison, versioning, and maintenance of that living model [10].

Reliability behaviors suitable for simulation

18. Normal, boundary, degraded, overload, failure, misuse, recovery, and intervention scenarios.
19. State history, ordering, concurrency, timeout, retry, duplication, compensation, and rollback.
20. Dependency availability, latency, freshness, correctness, fallback, and responsibility transfer.
21. Capacity thresholds, prioritization, backpressure, admission control, and load shedding.
22. Permissions, delegated authority, escalation, revocation, and Human-in-Control decisions.

4. SRE and SDSE: The Fundamental Comparison

Dimension	SRE	SDSE
Primary question	Is the operating service reliable enough, and how should it improve?	Is the proposed end-to-end reliability behavior the right behavior to build and accept?
Primary evidence	Telemetry, SLO performance, error-budget use, tests, incidents, and operational experience.	Simulation traces, modeled state, scenarios, expectations, counterexamples, and limitations.
Typical timing	Design influence through implementation, deployment, and operations; strongest evidence comes from executable systems.	Intent through operations; executable evidence can exist before production code.
Primary artifacts	SLIs, SLOs, alerts, dashboards, automation, runbooks, incident records, and postmortems.	Executable behavior model, scenario portfolio, review evidence, and Validated Behavioral Baseline.
Main strength	Measures and improves the behavior of real operating systems.	Challenges intended behavior before expensive commitments and exposure.
Principal limitation	Cannot independently remove misunderstandings already embedded in the intended behavior.	Cannot prove that production code and operations implement the model correctly.

Table 1. SRE and SDSE produce different but complementary reliability evidence.

The boundary is not rigid. Experienced SREs participate in architecture and readiness work long before launch [3]. SDSE strengthens that participation by giving the team an executable behavioral subject before the service exists. Conversely, SDSE depends on SRE to learn whether the implemented service conforms to the accepted promise under real workload, dependency, organizational, and environmental conditions.

5. Reliability Must Be Designed as Behavior

Reliability is often summarized by availability, latency, correctness, or freshness. These indicators are useful because they turn experience into measurable evidence. They are not the whole behavioral system. A customer outcome may depend on a sequence of services, policies, people, and external

providers. Each component can appear healthy while the combined outcome is delayed, duplicated, inconsistent, unauthorized, or unrecoverable.

Participant	Locally healthy evidence	End-to-end reliability risk
Checkout API	99.95% availability	Accepts a retry after payment succeeded but confirmation was delayed.
Payment service	Low error rate	Creates a second authorization because the retry carries a new intent identifier.
Inventory	Fresh and responsive	Releases the reservation before late payment confirmation arrives.
Notification	Messages delivered	Sends two receipts that imply two valid orders.
Operations	Incident resolved	Restores service but leaves payment, order, and inventory state inconsistent.

Table 2. Local SRE signals can remain healthy while the customer outcome is unreliable.

SDSE makes the sequence executable before deployment. SRE then determines whether the implemented and operating system produces the same state transitions, correlations, outcomes, and recovery behavior. Reliability becomes a chain of evidence instead of a collection of disconnected component targets.

6. From SLO Definition to SLO Simulation

SLOs specify a target level of reliability and support data-driven decisions [2]. The difficult work is not only choosing a percentage. The organization must decide which user journey matters, what counts as a good event, which population and time window apply, how dependencies affect the outcome, and what behavior should occur when the objective is threatened or missed.

SDSE gives teams an environment for exercising those meanings. A proposed SLO can be linked to modeled outcomes, and scenarios can test whether the indicator distinguishes successful, degraded, misleading, and unacceptable behavior. This is particularly valuable before sufficient production history exists.

SLO element	SDSE validation question
Outcome	Which user or dependent-system result is the reliability promise intended to protect?
Indicator	What observable event, state, duration, freshness, or correctness evidence represents that outcome?
Population	Which requests, transactions, users, regions, dependencies, or operating modes are included?
Threshold	What target is justified by consequence, expectation, feasibility, cost, and uncertainty?
Decision rule	What release, remediation, escalation, or investment behavior follows good or bad evidence?

SLO element	SDSE validation question
Model link	Which BSN expectation and scenarios justify the indicator and expose its limitations?

Table 3. Simulation helps validate the behavioral meaning behind an SLO.

SLO caution

A precise SLO can still govern the wrong behavior. Measurement quality does not compensate for an indicator that is disconnected from the outcome stakeholders actually need.

7. How SDSE Changes Error-Budget Management

An error budget converts an SLO into a quantified allowance for unreliability and helps balance feature delivery with stability work [1][4]. SDSE does not replace observed budget consumption. It makes the policies around that consumption explicit and executable before the organization depends on them.

23. Simulate which failure classes consume the budget and which are excluded.
24. Exercise release, freeze, remediation, escalation, and exception decisions at different burn rates.
25. Test whether local budget decisions create harmful system-wide incentives.
26. Explore dependency failures, customer segments, maintenance windows, and degraded modes that complicate attribution.
27. Connect budget exhaustion to accountable owners, authorized actions, and recovery evidence.

The resulting policy remains subject to production evidence. When actual incidents reveal that an exclusion hides customer harm, that one failure consumes disproportionate budget, or that the response policy delays recovery, the model and policy should be revised together. The budget becomes part of a living behavioral control system rather than a detached calculation.

8. Engineering Failure, Degradation, and Recovery Before Deployment

Reliable systems do not merely avoid failure. They contain, degrade, recover, reconcile, and return to safe operation. AWS defines reliability in terms of performing intended functions correctly and consistently and recovering quickly from failure [5]. The most expensive gaps often occur in the transitions: what happens after a timeout, during partial dependency loss, while data is stale, or when restoration completes but business state remains inconsistent.

Scenario family	Behavior that must be made explicit
Timeout and retry	Duplicate effects, lost correlation, retry storms, inconsistent acknowledgement.
Dependency loss	Fallback selection, stale data, degraded functionality, responsibility transfer.

Scenario family	Behavior that must be made explicit
Overload	Admission control, prioritization, backpressure, load shedding, recovery hysteresis.
Partial failure	Isolation, compensation, rollback, state reconciliation, customer communication.
Disaster recovery	Restoration order, data recovery point, dependency readiness, safe re-entry.
Human intervention	Detection, authority, escalation, manual control, evidence retention, handback.

Table 4. Failure simulation focuses on transitions and outcomes, not only component faults.

Simulation can reveal missing expectations and contradictory recovery rules without exposing a production environment. The validated scenarios then drive unit, integration, resilience, recovery, game-day, and conformance tests. Operational exercises remain essential because infrastructure, data, code, and people can behave differently from the model.

9. How SDSE Affects Observability Engineering

Observability is the ability to understand a system's internal state by examining its outputs. OpenTelemetry identifies traces, metrics, and logs as core telemetry signals and emphasizes that systems must be instrumented to emit them [6]. SDSE adds a prior question: what state, causality, and outcome must be observable if the organization is to recognize conformance, failure, recovery, or drift?

A simulation trace exposes the information reviewers use to understand why an outcome occurred. That trace can be translated into instrumentation requirements before code and interfaces harden. Correlation identifiers, state-transition events, dependency outcomes, authority decisions, recovery progress, and business results can therefore be designed into the service rather than reconstructed after an incident.

Model evidence	Derived observability requirement
BSN participant and interface	Service, actor, dependency, endpoint, or boundary identity.
State and transition	State-change event, invariant, version, timestamp, and transition cause.
Interaction sequence	Trace/span relationship, correlation and causation identifiers, handoff outcome.
Expectation	SLI calculation, conformance condition, alert criterion, or release control.
Failure and recovery	Detection event, degraded mode, retry/compensation result, recovery milestone.
Authority decision	Permission, delegation, approval, escalation, revocation, and accountable actor.

Table 5. Model traces can define telemetry that supports meaningful operational questions.

Observability design rule

Instrument the decisions and state transitions needed to explain behavior—not merely the components that are easiest to count.

10. How SDSE Changes Incident Preparedness and Response

Incident response depends on rapid recognition, clear leadership, coordinated action, communication, and controlled recovery [7]. SDSE allows teams to exercise these behaviors before an incident occurs. A scenario can include technical state, detection signals, escalation, authority, human decisions, containment, customer communication, restoration, reconciliation, and handback.

Validated scenarios can supply the behavioral foundation for runbooks and exercises. They do not prescribe every command. They define what must be recognized, protected, decided, achieved, and evidenced. This makes runbooks less likely to restore infrastructure while missing the customer or business state that made the incident consequential.

The closed learning loop

1. **Prepare.** Simulate consequential failure and recovery behavior; derive signals, roles, decision points, and runbook expectations.
2. **Respond.** Use operational evidence and incident command to contain harm, restore service, and preserve decision records.
3. **Compare.** Contrast the actual trace, state, timing, decisions, and outcome with the Validated Behavioral Baseline.
4. **Learn.** Separate implementation defects, operational gaps, incorrect assumptions, and incomplete behavioral intent.
5. **Revise.** Update the model, tests, SLOs, controls, telemetry, runbooks, automation, and code.

Google's postmortem guidance emphasizes consistent learning and systemic improvement rather than blame [8]. Connecting postmortem findings to the behavioral model gives that learning a durable place in the engineering baseline.

11. SDSE, Chaos Engineering, and Resilience Testing

Chaos engineering experiments on a system to build confidence that it can withstand turbulent conditions [9]. SDSE and chaos engineering operate at different evidence layers. SDSE explores the proposed behavior of a model before or alongside implementation. Chaos engineering evaluates the behavior of an implemented system under injected disruption, often in a realistic environment.

Aspect	SDSE simulation	Chaos experiment
Subject	Executable behavior model	Implemented service and environment
Primary timing	Before and throughout implementation	After an executable environment exists
Main question	What should happen, and what consequences follow from interacting rules and state?	Does the real system withstand the selected disruption as expected?
Risk	Model error or insufficient fidelity	Operational impact from the experiment
Connection	Prioritizes consequential hypotheses and expected outcomes	Validates or challenges the model and implementation

Table 6. Simulation and chaos engineering are complementary reliability experiments.

The strongest approach uses both. Simulation selects high-value conditions and clarifies acceptable outcomes. Resilience and chaos tests determine whether the actual system, infrastructure, data, procedures, and people produce them. Differences drive revision rather than being forced into one source of truth.

12. Reducing Operational Toil Through Behavioral Modeling

Toil is repetitive, manual, automatable work that scales with service growth and provides limited enduring value. SRE seeks to reduce it through engineering. SDSE adds the ability to model the operational behavior that creates toil before the burden becomes normalized.

6. Represent repetitive detection, triage, approval, handoff, remediation, and verification steps.
7. Expose missing signals or ambiguous ownership that force manual investigation.
8. Simulate proposed automation, self-healing, and exception handling before deployment.
9. Validate when automation may act, when it must escalate, and how humans regain control.
10. Identify toil transferred between teams rather than genuinely eliminated.

Automation caution

Automation can remove toil, but it can also make incorrect behavior faster and less visible. Behavioral simulation tests the authority and consequences of automation before operational dependence develops.

13. Capacity, Performance, and Dependency Planning

Capacity and performance are behavioral because correct responses change with load, resource constraints, dependency limits, priority, and time. A system may remain technically available while violating latency, freshness, fairness, cost, or recovery expectations. SDSE provides a place to make

policies such as admission control, backpressure, load shedding, prioritization, throttling, and graceful degradation explicit.

The model does not replace quantitative performance engineering. It identifies the scenarios, policy decisions, state transitions, and unacceptable outcomes that performance tests and capacity models must evaluate. Production measurements then calibrate thresholds and reveal behaviors the model omitted.

Planning concern	Behavioral question
Demand transition	What happens when load rises, spikes, shifts region, or changes request mix?
Resource threshold	Which behavior changes before, at, and after saturation?
Priority	Which users, transactions, or safety functions are protected first?
Dependency constraint	How are rate limits, latency, quota, failure, and recovery propagated?
Recovery	How does the system avoid oscillation, retry storms, or unsafe rapid re-entry?
Evidence	Which SLI, trace, event, or capacity signal proves the policy operated correctly?

Table 7. Capacity policies should be simulated as system behavior, not treated only as resource forecasts.

14. SRE for AI-Native and Agentic Systems

AI-native systems introduce probabilistic outputs, changing models, external tools, delegated action, and behavioral drift. Availability and latency remain important, but a highly available agent can still be unreliable if it selects the wrong tool, exceeds authority, fails to escalate uncertainty, communicates externally without approval, or produces an unacceptable business outcome.

SDSE provides an earlier environment for modeling permissions, boundaries, uncertainty, escalation, revocation, human intervention, recovery, and evidence retention. SRE provides the production mechanisms for detecting whether the agent and its controls remain within that accepted behavioral envelope.

Concern	SDSE definition	SRE operational evidence
Output quality	Expected outcome, uncertainty threshold, unacceptable result	Quality, refusal, correction, escalation, and business-outcome signals
Tool use	Allowed tools, preconditions, limits, side effects, rollback	Tool-call traces, outcomes, denied actions, compensation evidence
Delegated authority	Permission, approval, separation of duties, revocation	Authorization decisions, identity, intervention, audit evidence

Concern	SDSE definition	SRE operational evidence
Behavioral drift	Accepted envelope and material-change conditions	Drift indicators, resimulation trigger, renewed validation
Human control	When a person must review, approve, stop, or assume control	Escalation time, response, handoff, accountability, recovery evidence

Table 8. AI reliability requires evidence about behavior, authority, and human control.

Human-in-Control principle

AI may observe, propose, simulate, test, or execute within approved boundaries. Accountable humans must retain authority over intent, unacceptable outcomes, material change, delegated limits, and residual risk.

15. The Integrated SDSE-SRE Reliability Lifecycle

THE INTEGRATED SDSE-SRE RELIABILITY EVIDENCE LOOP

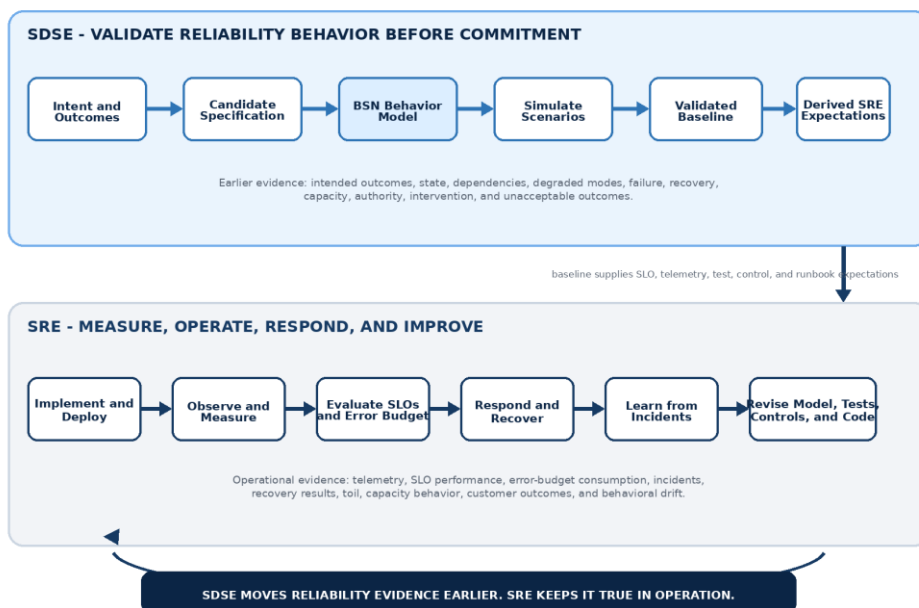


Figure 1. SDSE moves reliability evidence earlier; SRE keeps the behavioral promise true in operation.

The integrated lifecycle creates one evidence thread. Reliability intent supplies the model. Simulation supplies validated expectations. Those expectations drive design, tests, SLOs, telemetry, runbooks, controls, and release criteria. Production evidence then tests both the implementation and the assumptions behind the baseline.

1. **Establish intent.** Identify stakeholders, user outcomes, unacceptable outcomes, obligations, constraints, and the reliability decision requiring evidence.
2. **Bound the system.** Select consequential participants, dependencies, operating conditions, and exclusions.
3. **Create the Candidate Specification.** State the proposed behavior and unresolved reliability assumptions.
4. **Build the BSN model.** Represent state, interactions, logic, scenarios, expectations, recovery, and authority.
5. **Simulate by consequence.** Exercise normal, boundary, degraded, overload, failure, misuse, recovery, and intervention behavior.
6. **Validate the baseline.** Review traces, counterexamples, limitations, residual uncertainty, and acceptance accountability.
7. **Derive the SRE portfolio.** Define SLOs, telemetry, alerts, tests, runbooks, controls, capacity rules, and release evidence.
8. **Implement and operate.** Build, deploy, observe, respond, recover, automate, and manage error budgets.
9. **Compare and learn.** Assess production behavior and incidents against the baseline.
10. **Revise the evidence system.** Update the model, expectations, SLOs, tests, controls, runbooks, and implementation.

Single reliability promise

The behavior model, SLOs, telemetry, tests, controls, runbooks, releases, and operational learning should remain connected to one versioned behavioral promise.

16. Changing the Role of the Site Reliability Engineer

SDSE does not turn SREs into requirements analysts or require them to model every feature. It gives their production knowledge an earlier executable subject. SREs can challenge how proposed behavior fails, degrades, recovers, scales, emits evidence, transfers responsibility, and returns to safe operation before those decisions harden.

Lifecycle area	Expanded SRE contribution
Intent and discovery	Identify reliability outcomes, unacceptable outcomes, operating conditions, and decision consequences.
Behavior modeling	Contribute failure, recovery, dependency, load, observability, intervention, and operational scenarios.

Lifecycle area	Expanded SRE contribution
Validation	Challenge traces, SLO meaning, residual uncertainty, model fidelity, and acceptance evidence.
Implementation readiness	Derive telemetry, alerting, runbook, capacity, release, resilience, and conformance requirements.
Operations	Compare actual traces and outcomes with the baseline; detect drift and missing assumptions.
Learning	Translate incidents and operational experience into model, test, control, and implementation revisions.

Table 9. SDSE makes SRE influence more preventive without weakening operational accountability.

The result is not “shift SRE left” as a transfer of work. It is earlier participation in consequential decisions, supported by executable evidence that product, domain, engineering, security, operations, and governance stakeholders can review together.

17. Organizational and Team Implications

Reliability is a shared system outcome. SDSE makes that fact difficult to avoid because a behavior model crosses team and service boundaries. Product leaders define outcomes and unacceptable outcomes. Domain experts validate rules. Developers and architects shape implementation responsibilities. Security and risk specialists challenge controls. SREs contribute operating knowledge. Accountable decision owners accept the baseline and residual uncertainty.

11. Assign one accountable owner for accepting each consequential behavioral baseline.
12. Treat the BSN model as a working engineering artifact rather than presentation documentation.
13. Connect model changes to affected SLOs, tests, telemetry, controls, runbooks, and releases.
14. Preserve dissent, model limits, assumptions, and residual uncertainty in the evidence record.
15. Give unfavorable simulation or operational evidence authority to change commitments.
16. Keep modeling depth proportional to consequence, uncertainty, and decision leverage.

Governance outcome

Good reliability governance emerges when decision rights, evidence, controls, operating feedback, and accountability remain connected. It is not created by adding a separate approval gate.

18. Practical Adoption Approach

The strongest starting point is a bounded reliability experiment, not a broad tool rollout. Choose a use case important enough to expose real value but narrow enough to complete before the target decision.

Examples include a cross-service retry and recovery flow, a degraded operating mode, an error-budget response policy, a difficult dependency, or an AI-agent authority boundary.

Pilot element	Leadership question
Decision	What consequential reliability choice can still change because of the evidence?
Behavior	Which end-to-end outcome, failure, recovery, capacity, or authority behavior is uncertain?
Boundary	Which participants, dependencies, conditions, and exclusions are included?
Baseline	What current rework, incident, SLO, toil, recovery, or customer-impact evidence exists?
Participants	Which product, domain, engineering, security, SRE, operations, and governance roles are required?
Derived artifacts	Which SLOs, telemetry, tests, controls, runbooks, and release criteria should change?
Decision rule	What evidence will cause the organization to stop, revise, repeat, integrate, or scale?

Table 10. A credible pilot is bounded around a reliability decision, not a demonstration of a tool.

Adoption traps

17. Selecting trivial behavior that cannot demonstrate meaningful reliability value.
18. Beginning after architecture and SLO choices are already fixed.
19. Modeling broadly without identifying the decision the evidence must improve.
20. Treating model count, scenario count, alert count, or SLO count as the business result.
21. Allowing the model and operational artifacts to diverge into competing specifications.
22. Automating integration before proving that the combined practice improves decisions.

19. Measuring the Value of SDSE-Enabled SRE

Measurement should connect engineering activity to decisions, delivery outcomes, operational performance, and business consequences. The evaluation must include the investment in modeling, facilitation, training, review, tooling, integration, and maintenance. It should also avoid claiming that every simulated counterexample certainly would have become an incident.

Value layer	Measures to consider
Capability	Time to model behavior, run scenarios, interpret evidence, define SRE artifacts, and approve a baseline.
Leading evidence	Conflicts, missing expectations, unsafe paths, observability gaps, and recovery gaps found before implementation.
Decision quality	Material changes to scope, architecture, SLOs, controls, capacity policy, release criteria, or authority.

Value layer	Measures to consider
Delivery outcome	Reliability clarification after work begins, rework, test/alert churn, integration delay, and readiness exceptions.
Operational outcome	Incidents, detection time, understanding time, recovery time, repeat failure, toil, and behavioral drift.
Business outcome	Customer impact, failure cost, value loss, compliance exposure, trust, and acceptable autonomy.

Table 11. Measure decisions and outcomes improved—not the volume of artifacts produced.

Example value hypothesis

If a team simulates and validates a consequential cross-service reliability behavior before implementation, then late clarification, missing telemetry, recovery rework, and behavioral escapes should decline relative to a comparable baseline while the added modeling cost remains lower than the cost avoided or risk reduced.

20. Leadership Implications

Leaders should not frame SRE and SDSE as competing disciplines. The choice is about where uncertainty resides and when evidence can change the decision. When uncertainty concerns the behavior of an implemented service, SRE provides direct operational leverage. When uncertainty concerns the intended behavior of an interacting system, simulation can create evidence before implementation and exposure.

23. Treat reliability as an end-to-end behavioral property, not only a component metric.
24. Give SRE authority to influence intent, architecture, recovery, observability, and automation decisions early.
25. Require SLOs and error-budget policies to trace to validated stakeholder and system outcomes.
26. Connect BSN models, tests, telemetry, controls, runbooks, releases, and incident learning.
27. Preserve Human-in-Control authority over consequential AI and automation behavior.
28. Allow unfavorable evidence to revise scope, design, schedule, controls, release, and delegated authority.
29. Scale only after a bounded pilot demonstrates improved decisions, reliability outcomes, and reusable learning.

Leadership question

What important reliability decision would improve if the organization could execute and challenge the proposed behavior before committing to implementation?

Conclusion

SRE and SDSE reduce different forms of reliability uncertainty. SRE uses evidence from implemented and operating services to balance risk, protect users, respond to failure, and improve the system. SDSE uses executable models to determine whether the proposed reliability behavior is coherent, acceptable, observable, recoverable, and governable before production evidence exists.

The integration is especially important for distributed, regulated, safety- or security-sensitive, and AI-native systems. In these environments, local health can coexist with unreliable end-to-end behavior. An SLO can be met while the wrong outcome occurs. A recovery can complete while business state remains inconsistent. An agent can be available while acting outside acceptable authority.

The practical starting point is small but consequential: one reliability decision, one bounded behavior, one executable model, one validated scenario set, and one SRE evidence portfolio derived from the baseline. Operate the service, compare reality with the promise, and revise both. That creates the continuous reliability discipline modern systems require.

Closing statement

SDSE validates the reliability behavior before deployment. SRE measures and improves it in operation. Together they create confidence from intent through learning.

Appendix A. SDSE-SRE Reliability Readiness Checklist

Review area	Question
Outcome	Is the reliability promise expressed as a user or dependent-system outcome?
Boundary	Are consequential services, people, suppliers, policies, and dependencies included?
Conditions	Are normal, boundary, degraded, overload, failure, misuse, recovery, and intervention represented?
SLO meaning	Can each important SLI and SLO be traced to a validated behavioral expectation?
Observability	Can telemetry explain state, sequence, causality, authority, recovery, and outcome?
Response	Are detection, ownership, escalation, containment, communication, recovery, and handback explicit?
Conformance	Can production traces and outcomes be compared with the baseline?
Learning	Do incidents and drift trigger model, test, SLO, control, runbook, and implementation revision?

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

Appendix B. Reliability Scenario Template

Field	Working answer
Intended outcome	_____
Unacceptable outcome	_____
Participants and dependencies	_____
Initial state and conditions	_____
Initiating event	_____
Expected degraded behavior	_____
Detection and evidence	_____
Authority and intervention	_____
Recovery and reconciliation	_____
Residual uncertainty	_____

Table B1. Document the behavior and evidence—not only the initiating fault.

Appendix C. Mapping BSN Evidence to SRE Artifacts

BSN evidence	Derived SRE artifact
--------------	----------------------

BSN evidence	Derived SRE artifact
Participant and interface	Ownership, dependency map, service boundary, contract and health evidence
State transition	Metric, event, log, trace span, invariant, diagnostic and conformance check
Scenario expectation	SLI, SLO, acceptance criterion, alert or release condition
Failure behavior	Resilience test, chaos hypothesis, fault-containment and degraded-mode control
Recovery behavior	Runbook, recovery test, reconciliation check, RTO/RPO evidence
Authority decision	Permission control, approval, escalation, revocation and audit evidence
Model limitation	Residual risk, monitoring requirement, review date and resimulation trigger

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

Appendix D. SDSE-SRE Pilot Worksheet

Field	Working answer
Reliability decision and owner	_____
Proposed behavior and consequence	_____
Model boundary and exclusions	_____
Priority scenarios	_____
Validated expectations	_____
Derived SLO and error-budget changes	_____
Derived telemetry, test and runbook changes	_____
Baseline measures and pilot duration	_____
Continuation decision and date	_____

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

Appendix E. Leadership Review Checklist

- 30.The reliability promise has an accountable owner and explicit unacceptable outcomes.
- 31.SRE participates early enough to influence behavioral and architectural decisions.
- 32.SLOs and error-budget policies trace to validated customer and system behavior.
- 33.Failure, degradation, overload, recovery, and intervention behavior have been exercised.
- 34.Telemetry can explain state, causality, authority, recovery, and customer outcome.

35.The BSN model, tests, controls, runbooks, releases, and operational evidence remain connected.

36.Operational learning can revise the baseline and engineering commitments.

37.The pilot ends with an explicit decision to stop, revise, repeat, integrate, or scale.

References

The following sources support the description of SRE, SLOs, error budgets, lifecycle engagement, reliability, observability, incident response, postmortem learning, chaos engineering, and SDSE. The integrated SDSE-SRE framework is presented by the author.

- [1] Beyer, Betsy, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, eds. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media, 2016. [Official online book](#)
- [2] Beyer, Betsy, Niall Richard Murphy, David K. Rensin, Kent Kawahara, and Stephen Thorne, eds. The Site Reliability Workbook. O'Reilly Media, 2018. [Official online workbook](#)
- [3] Google. "SRE Engagement Model." The Site Reliability Workbook. [Official chapter](#)
- [4] Google. "Example Error Budget Policy." The Site Reliability Workbook. [Official example](#)
- [5] Amazon Web Services. AWS Well-Architected Framework: Reliability Pillar, updated 2024. [Official guidance](#)
- [6] OpenTelemetry. "Observability Primer" and "Signals." [Official documentation](#)
- [7] Google. "Incident Response." The Site Reliability Workbook. [Official chapter](#)
- [8] Google. "Postmortem Culture: Learning from Failure." The Site Reliability Workbook. [Official chapter](#)
- [9] Principles of Chaos Engineering. [Official principles](#)
- [10] Hornbeek, Marc, with Debashis Bhattacharyya and Jitin Dhanani. Simulation-Driven Software Engineering: A Behavior-First Discipline for Modern Systems, 2026. [Purchase on Amazon](#)

About the Author

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

Contact: mhornbeek@engineeringdevops.com