

Behavior as the Source of Truth Across the Software Value Stream

Using executable behavioral promises to connect intent with delivery and operational evidence

The central proposition

A versioned behavioral promise gives every stage of the value stream a shared reference for generating artifacts, validating outcomes, governing change, and learning from production.

By Marc Hornbeek, a.k.a. DevOps_The_Gray

Originator of Simulation-Driven Software Engineering

September 2026

Executive Summary

Software changes form across the value stream. Intent becomes a specification, architecture, design, implementation, release candidate, deployment, and operating service. Each handoff can separate the delivered system from the outcome stakeholders expected. A shared, executable description of intended behavior gives those stages a persistent reference against which evidence can be reviewed.

This white paper follows one login-protection promise: for a single account under observation, deny failed logins; on the third failure, block further attempts, preserve evidence, and raise a security alert. Behavior Simulation Notation (BSN) expresses the states, triggers, effects, and scenario explicitly. Simulation challenges that candidate specification before implementation commitments are made. The approved behavior then guides architecture, code, tests, release decisions, deployment checks, monitoring, and incident response.

Simulation-Driven Software Engineering (SDSE) keeps the behavior model connected to the engineering lifecycle. Code, configuration, scan findings, system tests, and production telemetry remain necessary evidence. A passing simulation supports only its modeled conditions, and the login example deliberately leaves observation windows, successful-login resets, identity correlation, and unblock policy for further specification.

Core conclusion

Behavior is the source of truth for intended outcomes. Implementation and operational evidence establish whether the system actually keeps that promise.

1. Define and simulate the behavioral promise before downstream artifacts are accepted.
2. Connect the model version to implementation, candidate identity, tests, approvals, and deployment evidence.
3. Compare production behavior with the promise and feed incidents back into the model.
4. Keep ownership and consequential risk decisions with accountable people.

The current BSN Workbench supports modeling, structural validation, scenario simulation, and traces. Automated artifact derivation and production comparison require additional engineering; they are not assumed capabilities in this paper.

Contents

Executive Summary	2
1 A shared behavioral reference	4
2 The login protection promise	4
3 The BSN model and simulation scenario	5
4 Intent and a reviewable specification	6
5 Architecture and design	6
6 Implementation and AI assistance	6
7 Integration and DevSecOps validation	7
8 Release candidate evidence and approval	7
9 Deployment validation	8
10 Production monitoring and incident response	8
11 Governance of behavioral change	8
12 SDSE across the value stream	9
References and further information	9
About the author	9

1 A shared behavioral reference

A software product changes form many times before it reaches a customer. An intent becomes a specification, a design, code, a release candidate, a deployed service, and eventually an incident record. Each handoff creates a chance to lose the original meaning. The most durable reference is the behavior the system is expected to demonstrate under stated conditions. Keep that behavior visible and testable, and every stage can produce evidence against the same promise.

A login protection example will carry the argument from its first statement through production response. It is small enough for a product owner, engineer, security analyst, and operator to discuss together. Each can ask what the system promises, how the team will demonstrate it, and what they will do when evidence contradicts it. AI can help produce artifacts along the way, but people must own the promise and the decisions made from the evidence.

2 The login protection promise

Suppose a service owner says: “For one account under observation, deny each failed login. On the third failed login, block further attempts, preserve evidence, and raise a security alert.” The customer expects protection; the security operations team needs a dependable signal; the operator needs enough evidence to investigate. The promise includes a trigger, an ordered response, a resulting state, and observable outputs.

Here is a compact Behavior Simulation Notation model of that promise. Each account would need an independent instance of this state in a real implementation. This example assumes the three input events refer to the same account and deliberately leaves the observation window, successful-login reset, identity correlation, and eventual unblock policy for further specification.

BSN gives the team explicit states, triggers, effects, and a scenario that can be simulated before implementation. A passing scenario supports the stated case; it does not prove every possible security condition. Reviewers should inspect the trace as well as the result, add scenarios for blocked attempts and recovery, and challenge the assumptions that this small model exposes. The model’s ACTION is an observable marker of required evidence preservation; the design must later define what evidence is stored, where, for how long, and with what privacy controls.

Scope of the example

The model illustrates behavior for one account. Production validation must add scenarios for concurrent attempts, recovery, delivery failures, privacy, and other operational conditions.

3 The BSN model and simulation scenario

The following illustrative model and scenario are preserved from the source article. Review the trace and the assumptions together before accepting the candidate behavior.

LoginGuard behavior model

```
BOX LoginGuard
INTERFACE
  INPUT FailedLogin
  OUTPUT AccessDenied
  OUTPUT SecurityAlert
END
OPERATING END
LOGIC
  STATE Watching
  STATE FailedOnce
  STATE FailedTwice
  STATE Blocked
  RULE FirstFailure
    ON INPUT FailedLogin
    IN Watching
    THEN STATE FailedOnce
    AND OUTPUT AccessDenied
  END
  RULE SecondFailure
    ON INPUT FailedLogin
    IN FailedOnce
    THEN STATE FailedTwice
    AND OUTPUT AccessDenied
  END
  RULE ThirdFailure
    ON INPUT FailedLogin
    IN FailedTwice
    THEN STATE Blocked
    AND OUTPUT AccessDenied
    AND ACTION PreserveEvidence
    AND OUTPUT SecurityAlert
  END
  RULE RejectWhileBlocked
    ON INPUT FailedLogin
    IN Blocked
    THEN OUTPUT AccessDenied
  END
END
END
```

ThreeFailures scenario

```
SCENARIO ThreeFailures
INIT LoginGuard IN Watching
AT T0 INPUT FailedLogin TO LoginGuard
AT +1m INPUT FailedLogin TO LoginGuard
AT +1m INPUT FailedLogin TO LoginGuard
EXPECT LoginGuard STATE Blocked
EXPECT LoginGuard OUTPUT AccessDenied
EXPECT LoginGuard ACTION PreserveEvidence
EXPECT LoginGuard OUTPUT SecurityAlert
END
```

4 Intent and a reviewable specification

At inception, ask who needs this behavior and what harm follows if it fails. Product and security owners can agree on the blocked state and alert as outcomes, while operations can insist on usable evidence. Record the scope of “one account,” define the meaning of a failed attempt, and decide what should happen when a valid login occurs between failures. Those decisions belong in the behavioral specification, not in assumptions scattered across tickets and code.

The BSN model is the candidate specification. Run ThreeFailures and examine the ordered trace: first failure moves to FailedOnce, second to FailedTwice, third to Blocked, then evidence and alert effects occur. Add a fourth attempt scenario to demonstrate continued denial, and let a security analyst review what an actionable alert must contain. If the team changes the threshold or introduces a time window, revise the model and rerun the scenarios before downstream artifacts are accepted. Otherwise different teams will implement different meanings of the same sentence.

5 Architecture and design

The behavior implies responsibilities. Authentication must emit an event with a stable account identity; the guard must retain per-account state; an evidence store must accept a protected record; and the alert path must reach the security operations team. An architecture review can therefore ask whether those paths exist, whether a restart loses the count, and whether a failed alert delivery is visible. A box and arrows alone cannot answer whether the third attempt actually reaches a blocked state.

Design translates the model into interfaces, state storage, event schemas, access controls, alert routing, and recovery rules. The design should identify who may unblock an account and how a false positive is handled. It should specify correlation IDs and redaction so responders can investigate without placing credentials in logs. Each decision should refer back to the behavior it implements and identify any additional behavior the initial model omitted. Without that review, the alert may arrive promptly but leave the responder unable to tell which account needs attention.

6 Implementation and AI assistance

The model can guide human or AI assisted creation of API contracts, state transition tables, acceptance criteria, test cases, audit schemas, alert configuration, and operational runbooks. A coding assistant can be given the BSN contract and asked to implement its states and effects; a review assistant can compare a change with the contract and point out missing transitions. Generated material still needs engineering review. A plausible test that checks for an alert but forgets denial on the fourth attempt is insufficient.

Today the BSN Workbench validates model structure, executes scenarios, and produces traces. Its published manual describes automatic code, test suite, and documentation generation as future direction. Teams can derive those artifacts manually or build their own controlled integrations, recording model version, generation method, reviewer, and artifact version. Ask the reviewer to compare the AI output with the scenario and with any new behavior introduced by the implementation. Provenance and human judgment matter when an AI generated change enters the pipeline.

7 Integration and DevSecOps validation

Unit tests can exercise each state transition. Integration tests must cross the real authentication service, state store, evidence store, alert transport, and security console. Inject three failed attempts for one account, then a fourth; verify denial, state persistence, one actionable alert, and protected evidence. Test lost messages, retries, clock differences, concurrent attempts, and a restarted service. These tests expose gaps between the simulated model and the actual system.

DevSecOps controls still check code quality, dependencies, secrets, configuration, identity permissions, and supply chain integrity. Tie their findings to the behavioral outcome. A dependency scan may be clean while the evidence store is writable by the wrong principal, or while a logging change leaks passwords. Security review should examine both the promise and the mechanisms supporting it. The pipeline should retain the model version and scenario results beside conventional scan results so a green build has a clear behavioral meaning.

8 Release candidate evidence and approval

A release candidate is a specific binary, configuration, dependency set, and deployment manifest. Test that exact candidate in a representative environment. Compare actual event traces with the expected sequence, and retain the results under the candidate's identity. Packaging checks should verify that alert endpoints, credentials, state storage, and evidence retention settings accompany the build. A passed test from a different commit or configuration does not support approval of this candidate.

Delivery validation adds operational conditions: can the on-call team see the alert, find the evidence, and identify the affected account without exposing sensitive data? Release approval should examine the unresolved deviations and make a human decision about risk. The approval record should name the behavioral promise, the tested candidate, relevant scenarios, observed traces, security findings, and any accepted exception. A signature on a change ticket is useful only when the approver can see what behavior was actually demonstrated.

Evidence retained for the release decision

Evidence	Connection to the behavioral promise
Model and scenarios	Identify the model version, conditions exercised, expected effects, and reviewed traces.
Release candidate	Bind results to the exact binary, configuration, dependencies, and deployment manifest.
Security findings	Record conventional control results and deviations affecting denial, evidence, or alert delivery.
Human approval	Record the decision owner, unresolved deviations, and explicitly accepted exceptions.

9 Deployment validation

A progressive deployment can issue controlled failed logins in a safe test account or replay approved synthetic events. Before widening exposure, confirm the third attempt blocks that account, a fourth is denied, and the security alert reaches its destination. Stop rollout when the promised path fails, preserve diagnostic evidence, and apply the documented recovery procedure. Deployment automation needs explicit conditions for those actions; the BSN contract supplies their reason and their expected outcome.

The deployment record should connect candidate identity, environment configuration, model version, test result, operator decision, and rollback result. This gives the next team a coherent explanation of what went live. Check again after configuration changes, even if the binary is unchanged. An alert route changed at deployment time can invalidate the behavior demonstrated in preproduction.

10 Production monitoring and incident response

Production telemetry should let operators reconstruct the sequence of failed attempts, state changes, denial, evidence capture, and alert delivery. Monitor missing alerts, repeated blocked attempts, excessive false positives, and evidence collection failures. A dashboard that shows a healthy service is incomplete if the security behavior has stopped working. Use privacy preserving correlation so the observed trace can be compared with the promise without storing secrets.

When an incident occurs, the behavior model becomes an investigation aid. Responders can ask which effect failed: counting, block enforcement, evidence preservation, or alert delivery. The runbook should specify how to isolate affected accounts, preserve records, contain the attack, and restore service safely. After the incident, revise the model and scenarios if the original promise was incomplete, or repair the implementation if the system broke a sound promise. Either way, feed the finding back to inception and specification.

11 Governance of behavioral change

A useful behavioral baseline is versioned and owned. Changes to a threshold, exception, privacy rule, or response action should receive a reason, a review, and new simulation and system evidence. Teams can move quickly because the question is precise: which promised outcome changes, and which artifacts and controls must follow? This is especially valuable for AI generated code or configuration, where a small edit can alter behavior across several services.

Behavior is the best source of truth for intended outcomes, while code, configuration, tests, and telemetry show how those outcomes are pursued and whether they occur. It cannot replace empirical evidence or human judgment. A model may be incomplete, and a passing simulation may miss an environmental failure. Governance works when the model is tested against reality and corrected as the team learns.

12 SDSE across the value stream

Simulation-Driven Software Engineering offers one way to establish this persistent contract early. BSN makes the LoginGuard behavior explicit, and scenario simulation produces a trace that people can review before the service is built. The same declared behavior can then guide implementation, tests, gates, deployment checks, monitoring, and incident learning. The present Workbench supports modeling, validation, simulation, and traces; broader automated derivation and production comparison require additional engineering.

Return to the account in our example. The organization succeeds when the third failed attempt causes the intended block, the fourth is denied, the alert reaches a human who can act, and protected evidence explains what happened. It can show the evidence accumulated from specification to production and identify the stage where a promise ceased to hold. The security analyst has a usable signal, the operator has a recovery path, and the customer receives the protection that was intended. That is the practical value of keeping behavior at the center of the software value stream.

References and further information

Source article: Marc Hornbeek, Behavior as the Source of Truth Across the Software Value Stream. This white paper adapts the supplied article and preserves its illustrative BSN model and capability qualifications.

Technical reference cited in the source article: Marc Hornbeek, BSN Workbench User Manual v1.0, September 2026. The model is illustrative; production use requires additional scenarios and operational detail.

For more about SDSE and the BSN Workbench, visit www.PrimaModus.com.

About the author

Marc Hornbeek, known as DevOps-the-Gray, is an award-winning engineer, consultant, author, and instructor with five decades of experience in software and systems engineering. He is CEO of Engineering DevOps Consulting and Intelligent-Institute and a Senior Advisor to the DEVOPS INSTITUTE®.

Marc specializes in DevOps, DevSecOps, SRE, platform engineering, continuous testing, AI-native delivery, and governance. He has led more than 95 transformations across enterprises, manufacturers, service providers, and government organizations. Marc is an IEEE Outstanding Engineer and IEEE Life Member and has authored several books on secure, reliable, intelligent, and human-centered engineering.

LinkedIn: <https://www.linkedin.com/in/marchornbeek>

Developed by Marc Hornbeek with assistance from ChatGPT. Marc retains final editorial responsibility.