

The depth signature

A feature is not complete because its happy path renders. Its required depth depends on the system it changes. This paper defines a risk-shaped signature for testing security, data, concurrency, recovery, performance, accessibility and operations at the right level.

AUTHORED BY

Daniel Johnson & Gary McLean Hall

AI DRAFTING

Astra, Fable & the models of the moment

EVIDENCE CHECKED

15 September 2026. The date tells you when the cited material was reviewed, not when the future agreed to behave.

INVALIDATED BY

When new evidence materially changes the failure surfaces described here.

EXPIRY DATE

The reader's call. Before "it passed once" is promoted to an assurance strategy.

The founders own the argument; AI helped draft the prose. The sources and limits are here so you can challenge both.

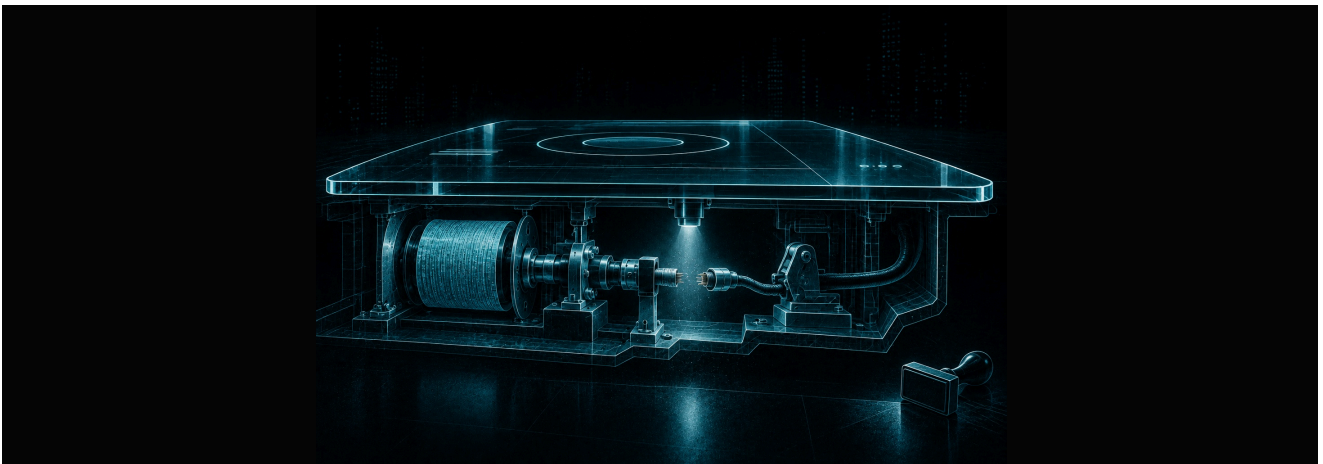
Contents

01. [The missing dimension of “done”](#)
 02. [Why AI makes shallow completion easier](#)
 03. [Deriving the signature](#)
 04. [From property to gate](#)
 05. [Coverage is not depth](#)
 06. [Domain depth](#)
 07. [System depth](#)
 08. [Human depth](#)
 09. [Operational depth](#)
 10. [Economic depth](#)
 11. [Choosing proportional depth](#)
 12. [An illustrative deadline-calculator signature](#)
 13. [Governance](#)
 14. [Conclusion](#)
 15. [References](#)
-

This paper proposes an assurance method. Its review date records an editorial and source check; it does not certify a product or constitute experimental validation of the method. The worked cases below are constructed examples. External research supports particular distinctions, while the proposed signature and its application remain engineering judgement.

The missing dimension of “done”

Software work is usually described by breadth: screens, endpoints, integrations and stories completed. Production failure arrives through depth: two requests at once, a retry after a timeout, an expired role, a partial migration, a keyboard-only user, a database restored from yesterday or an operator who has never seen the code.



A depth signature is the set of failure dimensions that a particular change must survive before its completion claim is credible. It replaces generic language such as “production-ready” with a risk-shaped verification contract.

The signature is not a universal checklist. A static paragraph and a payment ledger do not deserve the same tests. The discipline is to select depth from the obligation and consequences, then make each selected property observable.

Breadth and depth should be planned together. If a team estimates “build the form” and later treats permissions, duplicate submission and recovery as additional scope, the original completion claim was underspecified. Those properties are not universally required at the same level, but the decision about them belongs before acceptance. A depth signature makes the hidden portion of the obligation available for estimation and review.

The signature also separates absence of evidence from evidence of absence. A component may have no concurrency risk because it performs a pure local calculation over immutable inputs. That is a reasoned exclusion. A service that updates shared records does not become safe merely because the current tests contain no simultaneous requests. The first case can justify omitting a concurrency test; the second identifies an untested mechanism.

Why AI makes shallow completion easier

Generative tools are exceptionally good at producing visible breadth. A prompt can create routes, components, forms, validation messages and tests in minutes. The result looks finished because the artefacts associated with a feature are present.

A plausible screen hides this question

Happy path

One user submits one form successfully.

Missing challenge

Repeat the request after the commit succeeds but the response is lost.

Models also tend to follow the examples and success paths supplied in context. If the brief says “a user submits a form”, the generated implementation may handle a successful submission while inventing or omitting behaviour for duplicates, simultaneous edits and downstream failure. Asking for “edge cases” yields a plausible list; it does not establish that the relevant system property has been exercised.

The answer is not a longer adjective. It is a gate tied to a failure model.

Deriving the signature

Begin with consequences. What harm follows if this unit is wrong, unavailable, slow, inaccessible or irrecoverable? Then identify the mechanisms by which that harm could occur.

Derive depth from a harm

Possible harm

A retry creates a duplicate durable business effect.

Needed observation

Concurrent requests, their identities and the final persisted effects.

A depth signature may include:

- domain correctness: rules, rounding, calendars, state transitions;

- data integrity: identity, uniqueness, history, migration and retention;
- concurrency: races, retries, idempotency and ordering;
- security: authentication, authorisation, input handling and secrets;
- privacy: minimisation, purpose, access, erasure and audit;
- resilience: dependency failure, restart, recovery and rollback;
- performance: latency, throughput, resource bounds and degradation;
- accessibility: semantics, keyboard use, focus, contrast and alternatives;
- operability: logs, metrics, alerts, runbooks and ownership;
- economics: infrastructure cost under realistic use; and
- changeability: boundaries, tests and documentation for the next change.

Not every unit selects every dimension. Every omitted dimension should be explicitly assessed as irrelevant or shown to be covered elsewhere.

The derivation should produce a failure model, not just a list of quality nouns. For an illustrative booking operation, “data integrity” becomes the possibility that two callers reserve the same capacity. “Resilience” becomes uncertainty after the database commits but before the caller receives a response. “Human depth” becomes whether the caller can distinguish a confirmed booking from a request still being retried. These mechanisms imply different observations even though they arise in one feature.

For each mechanism, name the state that must remain true, the event that could violate it and the place where the violation can be observed. Existing evidence may already cover part of the claim. Reusing that evidence is sensible if the changed artefact does not invalidate its assumptions. The signature should make this dependency explicit so an old passing result is not carried forward after the state model or permission boundary has changed underneath it.

From property to gate

Each selected dimension needs a claim, a challenge and evidence.

The idempotency gate

Challenge

Repeat concurrent requests with the same key.

Observation

One intended durable effect and the specified response semantics.

“The endpoint is idempotent” is a claim. Repeating the same request concurrently with the same key is a challenge. One durable effect, stable response semantics and a recorded test run are evidence.

“The service is accessible” is too broad. “Every interactive path in this unit is keyboard-operable, preserves visible focus and exposes an accessible name” can be exercised by automated checks and manual traversal. The evidence is bounded; it does not certify all accessibility for all users.

“The migration is safe” becomes a rehearsal against a representative snapshot, invariant checks before and after, a measured duration, a rollback attempt and a record of data that cannot be reversed.

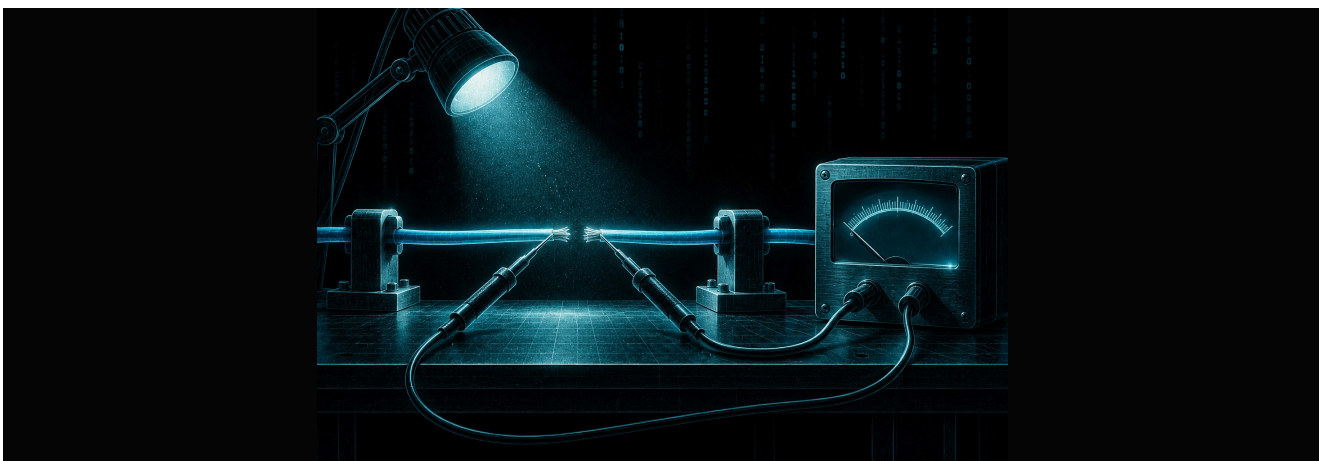
The gate must fail under a relevant mutation. If removing the permission check leaves the authorisation suite green, the suite does not support the claim regardless of coverage percentage.

A useful gate has sensitivity and specificity. It should reject the relevant defect, but it should not fail because of unrelated timing noise or unstable output. Repeated false failures train teams to rerun until green, weakening the authority of the whole suite. Diagnose whether the asserted property, test driver or environment is unstable; do not automatically widen tolerances until the failure disappears.

The evidence record should retain the challenge as well as the verdict. “Idempotency passed” is less useful than the request identity, number and arrangement of concurrent attempts, initial state, observed effects and code revision. This record lets a reviewer see whether the test actually exercised the disputed mechanism. It also gives future investigators a reproducible starting point when production reveals a case the original challenge did not cover.

Coverage is not depth

Line and branch coverage answer whether execution touched code. They do not answer whether assertions were capable of detecting wrong behaviour. Research on the oracle gap and mutation testing shows the distinction: a suite may execute a statement without checking its effect.



Google's Productive Coverage work makes a related point from industrial practice: uncovered code has different importance depending on similarity to tested or frequently executed code. Coverage becomes useful when connected to risk and action, not used as a single target.

The distinction has empirical support, but the research is not a universal conversion formula from coverage to safety. The study of coverage and mutation score investigates where executed code lacks effective assertions; Productive Coverage concerns making coverage information more actionable. Neither paper validates a particular percentage as an acceptable release threshold for a different organisation. Their relevance is the need to inspect what a metric actually observes. [Jain et al., 2023](#); [Petrovic et al., 2024](#).

A depth signature treats coverage as one signal. Mutation score, property tests, observed production paths, incident history and domain criticality can be stronger signals for particular claims.

Suppose a suite invokes a permission-sensitive endpoint across many cases but checks only that each response is well-formed JSON. It may achieve high branch coverage while failing to notice that another tenant's record is included. The missing depth is not more execution; it is an assertion about ownership and information exposure. A targeted mutation that removes the tenant filter should reveal whether the oracle can detect that failure.

This suggests a better use of coverage reports. Start with critical obligations and inspect where assertions fail to observe their effects. Then use uncovered paths to guide investigation rather than demanding an undifferentiated percentage. A rarely executed recovery branch can deserve more attention than a frequently used formatting helper. Conversely, a covered branch may deserve new tests if its effect is consequential and the current oracle is weak. The signature provides the context needed to make those choices.

Domain depth

Domain depth asks whether the system represents the work truthfully.

Do not flatten domain state

Unknown $\neq$ false

Missing knowledge is not evidence for a negative answer.

Absent $\neq$ zero

A missing amount must not silently become a valid calculated input.

Test ordinary examples, boundaries and counterexamples. Version applicable rules. Distinguish unknown from false, absent from zero, pending from rejected. Exercise time zones, calendars, rounding and historical rules where they matter. Verify that illegal states are difficult or impossible to represent.

For consequential decisions, preserve the inputs and rule version required to reproduce the result. A later explanation generated from prose is not the same as an audit record created at decision time.

Human domain review belongs here. Machines can exercise encoded rules at scale. They cannot decide that the encoded policy is legitimate.

System depth

System depth appears at boundaries between components.

Observe both sides of an interruption

Visible response

What the caller received when a dependency timed out.

Durable state

What committed, queued or remained pending after the same failure.

Dependencies time out. Messages duplicate. Webhooks arrive out of order. A transaction commits before a process crashes. Two users edit the same record. A cache holds old authorisation state. A provider changes a response field.

The signature should name which of these conditions matters and how the product behaves. Useful gates include contract tests, fault injection, concurrent test drivers, queue redelivery, clock control and restart tests. The evidence should cover both the response visible to a user and the resulting durable state.

Architecture helps when it creates places to test. Explicit domain boundaries and ports make dependencies replaceable with controlled doubles. Entangled systems hide failure behind end-to-end flakiness.

Fault injection should be placed at a meaningful boundary. A random process kill can demonstrate that a service restarts, yet miss the difficult interval between committing a record and acknowledging the request. The test should deliberately create uncertainty at that point, retry the operation and inspect both business state and downstream effects. A stable response alone cannot establish that no duplicate notification or charge occurred.

Test doubles require an evidence boundary of their own. A controlled dependency can make failure scenarios reproducible, but it only demonstrates behaviour under the contract encoded by that double. If the real provider has different retry or ordering semantics, the local test may be internally consistent and externally wrong. Use contract checks or representative integration observations to keep the double aligned with the boundary it represents. The purpose is controlled variation with known assumptions, not a miniature world in which the implementation always behaves well.

Human depth

Products fail people without throwing exceptions.

A human acceptance exercise

Reach

Keyboard users can reach the action and retain visible focus.

Understand

The person can distinguish saved, submitted and merely drafted state.

Accessibility is part of functional correctness. A control that cannot be reached by keyboard does not work for every intended user. Error text that disappears from assistive technology does not communicate the state. A session timeout that destroys unsaved statutory work is a reliability defect with a human consequence.

Human depth also covers comprehension and authority. Does the interface show which rule is being applied? Can a worker tell whether the system saved, submitted or merely drafted? Does a reviewer have the evidence needed for a genuine decision, or only a generated recommendation dressed as one?

Automated tools can find classes of problems. Representative human evaluation remains necessary for meaning, workflow and assistive-technology use.

Error recovery is a useful meeting point between accessibility and state correctness. After a failed submission, can the person find the error, understand which information remains saved and return to the relevant control without losing work? An automated scan may identify a missing label, but it will not establish that the explanation makes sense or that the recovery sequence matches the user's task. Record the intended interaction and test it as a workflow. This avoids treating accessibility as a cosmetic pass applied after the rest of the product's behaviour has already been decided.

Operational depth

Google's SRE guidance distinguishes internal system signals from observation of user-visible behaviour. Its practical lesson is that monitoring should support decisions and action. [Monitoring Distributed Systems](#).

Hand the runbook to someone else

Challenge

Deploy, diagnose and restore without the original build session.

Retain

The commands, restored state and ownership needed to repeat recovery.

An operational signature identifies service-level objectives, the signals that show whether users receive the service, the conditions that require human action and the runbook for recovery. It tests backup restoration, credential rotation, deployment rollback and ownership transfer.

"The team can operate it" should be challenged by asking somebody outside the build session to deploy, diagnose and restore it from the written material. If the original agent's conversational memory is required, the handover is incomplete.

A recovery exercise should begin from the written operating material, not from a builder narrating the missing steps. Give the operator a concrete symptom and the access expected in the real role. Observe whether they can identify the affected service, distinguish data loss from delayed processing and choose an authorised recovery action. Record undocumented dependencies and decisions that required the original author's memory.

This exercise also exposes the difference between a backup and a recoverable service. Restoring bytes is one part of recovery. The organisation may also need to re-establish credentials, replay events, reconcile effects and tell users which actions need repeating. Recovery evidence should identify the point in time restored and the treatment of work performed afterwards. A successful command with no account of the resulting business state is too narrow a claim to support an operational handover.

Economic depth

A technically correct product can still fail because its cost scales badly.

Define the cost-bearing unit

Denominator

A case, document, tenant or transaction with a stated workload.

Challenge

Realistic and adverse demand, including paid dependencies and resource bounds.

Record the expected unit of use: case, document, active user, inference, tenant or transaction. Model the main cost drivers. Load-test realistic and adverse patterns. Set budgets for compute, storage, transfer and paid dependencies. Observe them after release.

AI features need special attention because token, accelerator and retrieval costs can be variable. A deterministic cache or small local model may satisfy a bounded task more cheaply. A frontier model may be justified for rare high-value reasoning. The choice is an engineering decision, not an identity.

Choosing proportional depth

Use consequence, novelty, reversibility and reach.

Choose depth in proportion to consequence

Higher demand for evidence

Broad reach, high harm and difficult reversal.

Avoid a proxy

Blanket test volume that adds latency without observing the relevant failure.

A high-consequence, novel, hard-to-reverse change used by every tenant demands more evidence than a familiar, isolated, reversible change behind a flag. Existing incidents and production observations should raise the signature where reality has already revealed a failure mode.

Proportionality prevents two errors. Under-testing leaves important claims to hope. Blanket over-testing creates slow, brittle systems that teams bypass. Memon and colleagues reported that Google could not regression-test each code change individually despite substantial testing resources. Selection and feedback time are part of the design. [Memon et al., 2017](#).

An illustrative deadline-calculator signature

Consider a statutory-deadline calculator.

Two calendars. Two retained answers.

The issued answer stays attached to Calendar A. A new calculation may use B; changing today's calendar must not rewrite yesterday's decision.

Calendar A Count 2 → 3 → 4 March. Result: Wednesday 4 March.

Calendar B Exclude 3 March. Count 2 → 4 → 5. Result: Thursday 5 March.

Seeded fault Exclude the start date: both answers move a day late.

Invented policy example · not a statutory deadline or legal guidance.

Its signature includes domain correctness, data history, time, accessibility, performance and audit. Examples cover known dates, weekends, holidays and rule changes. Properties check ordering and reproducibility. The rule version is stored with the result. A mutation that changes an inclusive boundary must fail. The interface explains the start date, exclusions and result. A past answer can be reconstructed after the calendar changes.

It may not require a distributed load test if calculation is local and cheap. It does require a decision about authoritative holiday data and failure behaviour when that data is unavailable. The signature follows the obligation, not the fashion.

Take a constructed example in which an inclusive counting rule is implemented as an exclusive one. Ordinary dates may still look plausible, especially if the interface displays only the final result. The test corpus should include a boundary case where the conventions differ and assert the intermediate explanation as well as the date. A seeded change to that comparison should fail. This demonstrates sensitivity to the actual legal or policy interpretation the organisation has selected; it does not establish that the interpretation itself is legitimate.

Now change the holiday calendar after a decision has been issued. Historical reproduction should still use the recorded version, while a new calculation can use the current approved version. Test both paths and the distinction shown to the user. If required calendar data is missing, the specification must say whether calculation stops or produces a clearly provisional result. A test that fills the missing data with a convenient default would erase the very uncertainty the product needs to expose.

Finally, ask a representative user to explain the result from the interface. Can they identify the start date, rule, exclusions and status? Can they reach any correction or review action with the intended interaction method? This connects domain, data and human depth in one example. Separate green checks on the calculator and its controls are useful, but the assembled workflow must still communicate the consequential answer correctly.

A complete challenge, with an intentionally fictional rule

To make the mechanism inspectable without giving legal advice, define a fictional service rule: count three eligible working days from the recorded start date, including that date when eligible. Saturdays, Sundays and dates in the selected holiday-calendar version are excluded. This is an invented policy for the example, not a statement about any statutory period.

Use Monday 2 March 2026 as the start. Calendar A contains no exclusions in that week. The result is Wednesday 4 March: Monday, Tuesday and Wednesday count. Calendar B excludes Tuesday 3 March, producing Thursday 5 March. An exclusive-start mutation produces Thursday under Calendar A and Friday under Calendar B. The challenge therefore distinguishes the chosen interpretation using visible intermediate steps as well as a final date.

Dimension	Observation to retain	Deliberate fault that must be rejected
Counting rule	Ordered counted dates and final date	Exclude an eligible start date
Calendar history	Calendar identifier and retained contents	Recompute an old result using the current calendar
Input meaning	Recorded local date and timezone interpretation	Shift a date because a timestamp is converted incorrectly
Explanation	Start, exclusions, rule version and status	Display a correct date with a contradictory explanation
Recovery	Stored result and reproduction after restore	Restore the answer while losing its rule dependency

This small fixture establishes a known result. It does not establish completeness across all calendars. Add generated cases with a valid start, a bounded holiday set and the same rule version. Assert that every counted date is eligible, that the selected date is the third eligible date and that adding an exclusion cannot make the result earlier. The last property needs its preconditions: the counting rule and all other inputs remain unchanged, and the search domain still contains enough eligible dates. A generator that can create an endless absence of working days also needs a defined resource bound and failure result.

Now issue the Calendar A result, update the approved current calendar to B and retrieve the issued decision. Historical retrieval should show Wednesday under A. A new calculation should show Thursday under B. If the product supports a policy-authorised

correction, that action should produce a distinct record linking the old and new results. Updating the old row in place would erase the evidence that the original answer followed a different approved input.

The buyer's acceptance question is practical: can an operator explain both answers without contacting the developer? Supply the stored inputs and retained rule material to a reviewer who did not write the function. They should reproduce the result and identify why it differs from today's calculation. If only the current service can explain its past answer, the archive is dependent on continuing implementation behaviour. The signature should say whether that dependency is acceptable or whether independent reproduction is required.

There is a resource boundary too. A calendar lookup can fail, return malformed data or take longer than the interaction permits. The correct failure behaviour follows the product contract: refuse to issue a final answer, or present an explicitly provisional calculation if authorised. Silently substituting an empty holiday set would create a plausible but unsupported result. Fault injection should confirm the chosen behaviour and verify that a refused calculation does not leave a record labelled final.

Governance

Store the signature with the unit. Review it before implementation. Allow agents to propose additions, but require accountable acceptance for omissions. Record the commands, versions and results that satisfied each property.

Evidence has an expiry condition

Attached assumptions

Code revision, environment, dataset and interpretation.

Invalidation example

Calendar-storage changes require renewed evidence that old inputs remain available.

At release, publish a bounded claim: what was tested, against what data and under which conditions. Do not turn the signature into certification theatre. Unknowns and residual risks are part of the evidence.

Governance should preserve disagreement rather than force every dimension into a pass box. A reviewer may accept a known limitation because the affected path is isolated and reversible, while another limitation prevents release. Record the distinction, its owner and the condition for revisiting it. That makes residual risk actionable. A generic "accepted risk" label without a boundary or review trigger can become a permanent hiding place for unfinished obligations.

Signatures should also be maintained as the product learns. An incident can reveal a failure mechanism missing from several earlier units. Update the reusable challenge and identify where it applies, then distinguish newly tested releases from historical releases that never passed it. This is stronger than retrospectively claiming that an existing checklist already covered the problem. Depth improves when the organisation can admit exactly what its earlier evidence did not establish.

Evidence has an expiry condition

A gate result belongs to a code revision, environment, dataset and interpretation. Its usefulness can survive a later change only when the relevant assumptions survive. Editing unrelated help text need not force a complete migration rehearsal. Changing calendar storage invalidates evidence that historical calculation inputs remain available. The signature should identify that dependency rather than treating every old green result as permanently transferable.

This gives assurance a maintenance model. An owner can identify which claims a change may invalidate and commission the necessary rechecks. When an incident reveals a missing failure mechanism, record the affected claim and strengthen its challenge; do not merely add the incident's exact input to a growing regression list. The more general mechanism may apply across several products, while the fixture remains useful as a reproducible example.

For leadership, the useful report is a map of supported claims, unresolved failure mechanisms and decisions needed. A composite quality score can hide a critical weakness behind strong results elsewhere. Excellent response time does not compensate for a missing permission boundary. Depth is deliberately multidimensional because some properties are release conditions in their own right. Accepting a limitation should identify its consequence, owner and review trigger, with a clear distinction between an acceptable constraint and an unfulfilled promise.

Conclusion

Completeness is not the number of visible features. It is the degree to which the product's important obligations survive the ways reality can challenge them.

Define the depth signature from consequence. Turn properties into falsifiable gates. Prove the gates can fail. Preserve the evidence and its boundary. AI can help enumerate, implement and exercise far more of that surface than was previously economical, but the organisation must still decide which failures matter.

References

- NIST, [Secure Software Development Framework 1.1](#), 2022.
- Google, [Monitoring Distributed Systems](#), 2016.
- Sadowski et al., [Modern Code Review at Google](#), 2018.
- Petrovic et al., [Productive Coverage](#), 2024.
- Jain et al., [Mind the Gap: The Difference Between Coverage and Mutation Score Can Guide Testing Efforts](#), 2023.
- Memon et al., [Taming Google-scale continuous testing](#), 2017.