Orchestrating AI for Secure Software Delivery

A Security Observatory case study in human authority, evidence gates and multi-model engineering

Luca Pacini

Field	Value
Publication date	11 August 2026
Review status	Owner-adjudicated evidence-boundary and terminology corrections incorporated. The role-separated adversarial challenge of this rebuilt candidate has not been completed.
Case study	Security Observatory
Method status	Derived from one bounded project; not independently replicated or validated as a general delivery standard.
Document licence	CC BY 4.0 (FortMilo-authored text and figures).

Evidence-basis notice. This paper proposes a practical delivery method derived from one small, security-sensitive software project and from the cited standards and research. It does not claim causal proof that the method improves productivity, quality or security across all projects. Project-specific implementation, test and release records are partly private and are therefore treated as self-attested unless a public artefact is cited. Public repository state, live-site observations, automated test evidence and external research are kept as distinct evidence classes rather than merged into a single assurance claim.

Disclosure of AI use and competing interests. This paper was produced under the method it describes. ChatGPT/GPT contributed synthesis, research, requirements shaping and document production; Codex contributed bounded implementation work in the case-study repositories; Claude and Claude Code were used in role-separated adversarial-challenge roles during technical, customer-facing and publication review. Named tools are described as the concrete implementation of delivery roles, not as an endorsement or comparative product claim. Model agreement was never treated as proof. The author defined the scope and structure, adjudicated findings, owns every claim and retains sole release authority. The author is the founder and sole proprietor of FortMilo; the case-study product is a FortMilo product. No external funding was received.

Core proposition. AI may extend delivery capacity; it does not acquire delivery authority.

Contents

Abstract

Small software projects face an uncomfortable asymmetry. They must make architectural, security, quality and release decisions comparable in consequence to those made by larger teams, but they cannot reproduce the same division of labour. Generative artificial intelligence can reduce some of that capacity constraint by accelerating research, drafting, implementation, test construction, documentation and review. Used without an explicit control model, however, the same capability can amplify error: an agent may invent requirements, expand scope, produce insecure code, validate its own assumptions, conceal uncertainty behind fluent prose, or declare completion without evidence.

This paper presents evidence-gated AI-assisted delivery, a human-governed method for small security-sensitive software projects. The method separates delivery capacity from delivery authority. Humans retain ownership of intent, security boundaries, public claims and release decisions. AI systems operate within bounded task contracts. Automated checks establish reproducible evidence. Role-separated adversarial challenge is used to expose assumptions, but model or provider separation is not represented as independent third-party assurance. Completion is permitted only when the relevant evidence gate has been satisfied and a human release authority has adjudicated residual uncertainty.

The method consists of nine linked states: human-owned intent and constraints; a bounded issue contract; AI-assisted implementation; automated proof; role-separated adversarial challenge; human adjudication; a release gate; live verification; and governance close-out. Each transition requires explicit evidence. The paper defines roles, task-contract requirements, an evidence ladder, data-handling boundaries, release criteria and anti-patterns.

The case study documents how Security Observatory actually used this method across three evidence domains: a private Salesforce product repository, a public website repository, and private governance. It records the concrete allocation of ChatGPT/GPT, Codex, Claude/Claude Code and human authority; multi-repository and multi-worktree controls; primary-source web research; product and website release gates; and non-confirming cases in which the method detected defects or publication errors after earlier controls had failed to prevent them.

The principal claim is deliberately narrow. AI-assisted delivery can be made more governable when authority, evidence, baseline identity and uncertainty are represented explicitly. The method does not remove the need for engineering judgement, professional independent security review where required, or representative runtime validation. It provides a disciplined way for a small project to use AI without allowing fluency, model consensus or successful automation to substitute for proof.

1. Introduction

1.1 The capacity problem in small software projects

A small software project is not merely a large project with fewer people. It has a different risk structure. The same individual may define requirements, design the architecture, implement the change, review the code, write the tests, update the documentation and decide whether to release. This concentration of responsibility produces speed, continuity and domain coherence, but it also removes many of the natural challenge mechanisms present in a larger engineering organisation.

Security-sensitive software sharpens the problem. A seemingly small change can alter authorisation, evidence retention, data exposure, deployment behaviour or the meaning of a customer-facing statement. The cost of an incorrect claim may exceed the cost of an incorrect line of code. A system that reports security posture must therefore protect not only execution integrity, but also epistemic integrity, defined here operationally as the maintained correspondence among four things: what the system observed, what it retained, what the project has verified, and what any public artefact asserts. A project preserves epistemic integrity when no statement in the fourth category exceeds the evidence available in the first three.

AI assistance is attractive because it appears to restore missing capacity. A coding agent can implement a bounded change. A language model can review a design, compare documentation, draft tests or challenge a release statement. Yet capacity is not equivalent to accountability. The model does not bear the consequence of an insecure release, a misleading product claim or a corrupted repository. It cannot inherit the owner’s duty to decide what is acceptable.

The central design question is therefore not whether a small project should use AI. It is:

> **How can a small project use AI intensively while keeping intent, evidence, accountability and release authority human-owned?**

1.2 The failure of informal AI assistance

Informal AI use usually begins with an apparently harmless pattern: describe the desired outcome, accept a plausible implementation, run a few tests and move on. That pattern is insufficient for security-sensitive work because it obscures five separate questions:

1. Was the requested change actually the authorised change?

2. Did the implementation remain within the permitted security and data boundaries?

3. What evidence demonstrates that the implementation behaves as claimed?

4. Was the evidence subjected to role-separated adversarial challenge, or merely restated by the same reasoning process?

5. Who accepted the residual risk and authorised release?

Without explicit answers, an AI-assisted project can produce a polished but weakly governed result. The danger is not limited to hallucinated code. More common failures include scope laundering, where an unrequested feature is introduced as a tidy-up; evidence inflation, where a passing unit test becomes a universal support claim; review convergence, where two models repeat the same unsupported assumption; and false completion, where a task is called done despite an unverified deployment or customer-facing state.

1.3 Purpose and contribution

This paper contributes a practical governance method for a small project rather than a new model architecture or a universal software-development lifecycle. It combines four established ideas:

- risk management for organisations that develop, deploy or use AI;
- secure software-development practices;
- human oversight and assurance;
- evidence-bounded release governance.

Its distinctive contribution is deliberately narrow: a lightweight operating loop for governing AI contributions to ordinary software delivery, combined with explicit baseline identity, evidence classes, exclusions, role-separated challenge and human release authority. Among the instruments reviewed in Section 15, these elements are not assembled into the same project-level operating procedure for a small team using generative models as delivery contributors.

The method is intentionally compatible with ordinary issue trackers, Git repositories, automated tests, static validation, web research and human review. It does not require a dedicated governance platform or a large assurance function. The case study also documents where the process itself failed or had to be hardened, rather than presenting the method as having emerged fully formed.

2. Scope, definitions and methodological stance

2.1 Scope

The method is intended for small software products and reference implementations where:

- one person or a small team owns architecture and delivery;
- AI systems assist with research, drafting, coding, testing or review;
- security, privacy, compliance-adjacent or customer-trust claims matter;
- repository history and release artefacts are available;
- the project can define explicit human release authority.

It is not a substitute for formal safety engineering, regulated validation, professional penetration testing, legal advice or independent certification. It may support those activities by improving traceability, but it cannot confer assurance that has not been performed.

2.2 Definitions

Term	Meaning in this paper
AI-assisted delivery	The use of a generative model or coding agent to contribute to research, design, implementation, test, documentation or review.
Human authority	The retained human power to define scope, approve exceptions, accept risk, make public claims and authorise release.
Task contract	A bounded statement of authorised work, exclusions, starting state, evidence requirements and stop conditions.
Evidence gate	A condition that must be satisfied by inspectable evidence before work may advance to the next delivery state.
Role-separated adversarial challenge	Review performed from a deliberately distinct delivery role, model/provider perspective, prompt or evidence set to expose assumptions. It is a challenge mechanism, not third-party assurance.
Baseline identity	The repository, worktree, branch, expected commit and clean/dirty state that define which software or publication candidate is actually under review.
Adjudication	Human resolution of disagreement, uncertainty or conflicting evidence.
Release evidence	Evidence tied to the exact candidate or deployed artefact, not merely to a similar earlier state.
Governance close-out	The update of requirements, issue status and retained evidence after live verification.

2.3 Methodological stance

The method adopts three forms of restraint.

First, it is evidence-bounded. A statement may be no stronger than the evidence that supports it. Source inspection, an automated test, a runtime observation and a live deployment check are distinct evidence classes. None silently substitutes for another. The same discipline applies to baseline identity: evidence about one branch, worktree or commit does not silently transfer to another.

Second, it is human-governed. Human oversight is not represented by a ceremonial approval at the end. Human authority is embedded at the points where intent, security boundaries, public claims and release risk are decided.

Third, it is proportionate. A small project cannot reproduce the bureaucracy of a large regulated organisation. The method therefore concentrates control at the highest-leverage boundaries: task authorisation, sensitive data, role-separated adversarial challenge, release evidence and public claims.

3. Risk model for AI-assisted delivery

3.1 Risk is introduced at the interaction boundary

An AI model does not act in isolation. Risk emerges from the interaction between the model, the prompt, available tools, repository state, external services and human expectations. The same model can be low-risk when drafting non-sensitive prose and high-risk when given write access to a production-connected repository.

The relevant unit of control is therefore not “the model”. It is the authorised interaction: what the model may see, what it may change, what evidence it must produce and what decisions remain outside its authority.

3.2 Principal failure modes

Requirement invention

The model supplies missing requirements from statistical expectation rather than project authority. The output may be plausible and still be wrong for the product.

Scope laundering

A change is introduced as a refactor, tidy-up or best practice although it alters product behaviour, claims, dependencies or security posture. The problem is particularly acute when a vacated design slot is automatically filled rather than deliberately removed.

False completion

The model reports “done” after local implementation while deployment, visual acceptance, package installation or live verification remains outstanding.

Evidence inflation

A source-level observation is described as runtime proof. A single runtime execution is treated as edition-wide support. A test fixture is treated as customer-environment validation.

Review correlation

A second model agrees because it received the same framing, copied the same assumption or drew from the same incomplete evidence. Apparent consensus may therefore be correlated error rather than separate confirmation.

Self-review limitation

Language models can often identify superficial problems in their own output, but research has shown that reliable self-correction of reasoning generally requires external feedback. A model's revised answer is not, by itself, independent evidence.[10]

Security weakness propagation

AI-generated code can reproduce insecure patterns, mishandle validation, misunderstand platform-specific authorisation or pass weak tests. Empirical work on generated code found in public repositories has identified a high prevalence of security weaknesses across many weakness categories.[9]

Sensitive-data disclosure

Prompts, logs, tool calls or retained transcripts may expose secrets, production evidence, personal data or attacker-useful diagnostics. The risk exists even when the final code is safe.

Context collision and operator fatigue

Multiple model conversations, browser tabs, terminals and worktrees can each hold a coherent but different version of the project state. Fatigue makes it easier to paste a challenge into an implementation window, send an implementation prompt to the reviewer, execute a command from the wrong tree or act on a stale baseline. These are ordinary human errors, but in a multi-model workflow they can redirect a correct instruction into the wrong role, repository or authority boundary.

Automation bias

A fluent recommendation can receive more weight than its evidence warrants. Experimental work shows that domain experts presented with automated suggestions can accept improper output and take less initiative of their own,[11] and that explanations do not reliably reduce this bias and can increase it.[12]

3.3 Risk is not reduced by model plurality alone

Using several models can improve challenge coverage, but model plurality is not an assurance mechanism unless roles and evidence are separated. Three agents repeating the same repository claim do not create three independent observations. Different providers may still share public training material, common reasoning patterns or the same mistaken project framing.

In this method, provider diversity is therefore treated as role separation, not as an independence guarantee. A useful challenger should have a materially different objective, evidence view or access path from the implementer. The human adjudicator still resolves the result against source, tests, runtime evidence or authoritative external material.

4. Governance principles

4.1 Human-owned intent

The human owner defines the problem, the permitted outcome and the non-negotiable boundaries. AI may propose clarifications, but it does not convert ambiguity into authority. Where a missing decision affects security, legal wording, public claims or release scope, the correct outcome is a recorded blocker or owner decision.

4.2 Bounded authority

Every AI task should state what may be read, what may be changed, what must remain untouched and when the agent must stop. Tool access should be the minimum needed for the task. A model that can inspect a repository does not automatically need permission to push, deploy or alter connected services.

4.3 Source hierarchy

Conflicting statements are resolved against an explicit hierarchy. A typical hierarchy for this case study is:

1. current owner decision;

2. canonical requirements and decisions;

3. current source and tests;

4. runtime or deployed evidence;

5. historical documentation;

6. model inference or conversational memory.

The exact order may vary, but model inference or chat memory should never outrank current project authority or direct evidence. Where the answer depends on current repository or live-site state, the relevant source is re-read rather than recalled.

4.4 Evidence before acceptance

The model's narrative is not proof. Each material claim must be supported by an inspectable artefact: a source path, diff, test result, deployment result, HTTP response, rendered page, checksum or owner acceptance record.

4.5 Role separation

Implementation, challenge and release authority should be separated even when the project has one human owner. AI systems can supply additional roles, but the human remains the adjudicator. The agent that implemented the change should not be the sole authority on whether it is correct.

4.6 Data minimisation

The model receives only the information required to perform the task. Secrets, access tokens, session identifiers, private keys, certificate bodies, production customer evidence and raw sensitive diagnostics are excluded unless a specifically authorised secure process exists.

4.7 Traceable state transitions

A task does not move from "not done" to "done" because work occurred. It moves because the evidence required for that state exists. The issue, commit and release records should show the transition and the evidence supporting it.

4.8 Reproducibility

A reviewer should be able to identify the exact repository, worktree, branch, starting commit, clean/dirty state, changed files, validation commands, resulting commit and deployed artefact. Reproducibility need not mean complete public source, but it requires unambiguous internal provenance. Correct reasoning about the wrong baseline is still incorrect review.

4.9 Consensus is not proof

Agreement between AI systems is a review signal, not a truth criterion. When models disagree, the disagreement is adjudicated against source, requirements, tests or runtime evidence. When models agree, the evidence is still checked.

4.10 Public claims are release artefacts

Customer-facing wording, diagrams and whitepapers are part of the release surface. A false hardening claim can be more damaging than a missing marketing claim. Public statements therefore receive the same evidence discipline as code.

4.11 Operator state is part of the control boundary

Human authority does not make the human infallible. Attention, fatigue and context switching affect the reliability of adjudication and command execution. The method therefore uses visible role and tree labels, one write-capable task per tree, mandatory preflight checks and a simple stop rule: when the operator cannot state which role, repository, worktree and baseline is active without guessing, the task returns to read-only or stops.

5. The evidence-gated delivery cycle

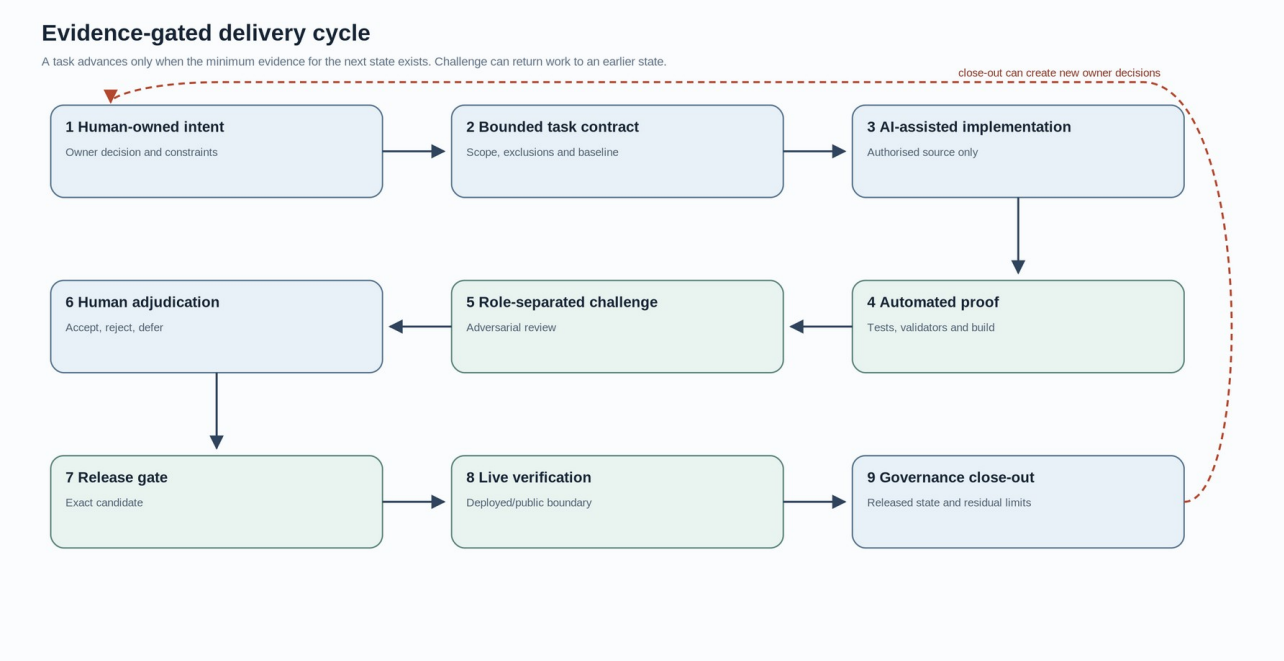

Figure 1. The evidence-gated delivery cycle. Every transition is gated by the minimum evidence for the target state; challenge and adjudication may return work to an earlier state.

The method uses nine states. The sequence is not a rigid waterfall; feedback may return work to an earlier state. What matters is that each transition is explicit and evidence-gated.

5.1 Human-owned intent and constraints

The owner defines the outcome, security posture, customer boundary and exclusions. For a security product, this may include read-only operation, no automatic remediation, no secret retention and no unsupported compliance claim.

Minimum evidence: owner decision or canonical requirement.

5.2 Bounded issue contract

The requirement is converted into a task contract with scope, exclusions, repository baseline, permitted tools, validation, stop conditions and expected reporting.

Minimum evidence: issue or task specification tied to the current baseline.

5.3 AI-assisted implementation

The implementation agent edits only the authorised source. It records blockers instead of working around them through unrelated changes. The agent may create tests and validators, but it may not weaken existing controls merely to obtain a pass.

Minimum evidence: source diff and changed-file inventory.

5.4 Automated proof

Builds, unit tests, static checks, security validators, content validators and integrity checks are executed. Automated proof should include negative tests for prohibited behaviour, not only positive tests for the desired path.

Minimum evidence: command output tied to the exact candidate.

5.5 Role-separated adversarial challenge

A challenger reviews the result from an adversarial, customer or assurance perspective. It should search for omissions, scope drift, insecure assumptions, misleading wording and untested failure paths. The challenger does not gain authority to expand the task automatically.

Minimum evidence: review findings with source references and explicit uncertainty.

5.6 Human adjudication

The owner accepts, rejects or modifies review findings. Decisions are recorded, particularly where a review recommendation conflicts with project boundaries or introduces a new claim.

Minimum evidence: owner decision and updated task boundary where necessary.

5.7 Release gate

The exact candidate is checked against release criteria: clean repository state, correct branch, required tests, visual acceptance, package or deployment evidence, protected secrets and approved public wording.

Minimum evidence: release checklist tied to the candidate SHA or immutable artefact.

5.8 Live verification

The deployed result is verified separately from the local build. Examples include HTTP status, live content checks, package installation, runtime execution, checksum comparison or customer-facing visual inspection.

Minimum evidence: live observations and artefact identity.

5.9 Governance close-out

The issue and canonical requirements are updated with the released state, evidence and remaining limitations. Deferred work is preserved without being represented as complete.

Minimum evidence: closed issue, updated status and retained release record.

6. Roles and separation of responsibility

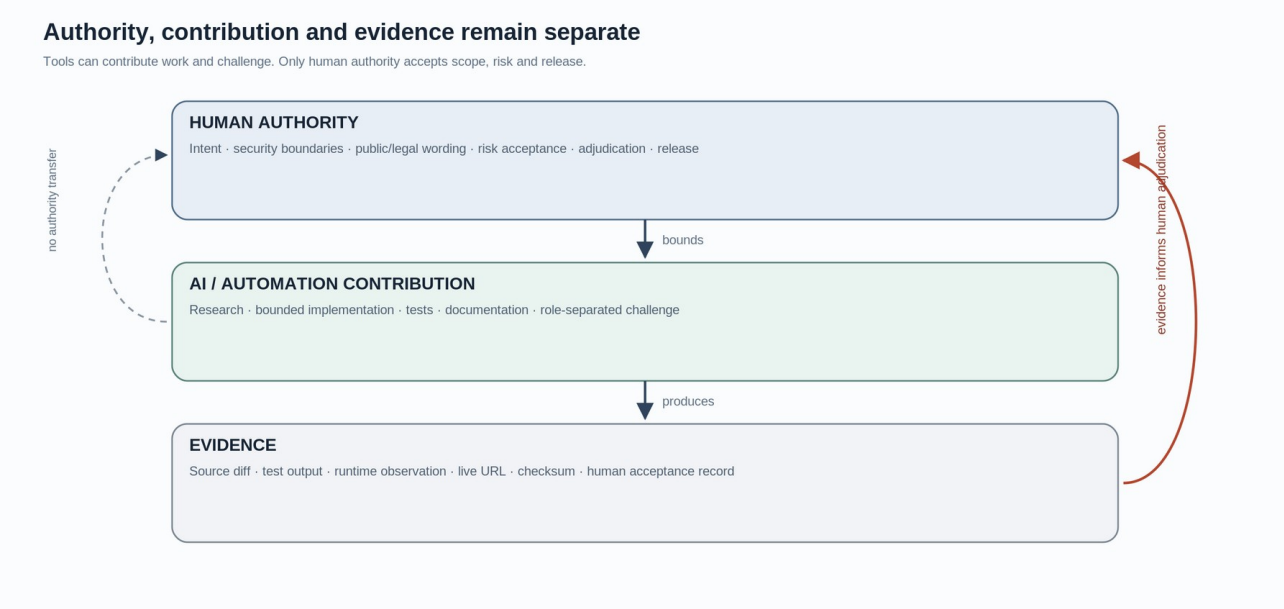

Figure 2. Human authority, AI contribution and evidence remain separate. Tools may contribute work or challenge, but neither contribution nor model agreement transfers release authority.

6.1 Owner, architect and release authority

The human owner controls intent, architecture, security boundaries, risk acceptance, legal and public wording, visual acceptance and release. In the case study this role was held by Luca Pacini. The same person may perform several human activities, but the authority boundary remains explicit: implementation convenience, reviewer preference or model confidence cannot silently become product policy.

6.2 Orchestration, synthesis and research role

A general-purpose model can translate owner decisions into bounded tasks, compare competing findings, perform primary-source research and maintain a coherent view across repositories. In the case study this role was performed primarily by ChatGPT/GPT.

The orchestration role was not an authoritative project database. When current state mattered, it re-read the canonical requirements, GitHub state or live web surface. Web research was used to verify external claims; it was not allowed to stand in for evidence about the private implementation.

6.3 Implementation role

The implementation role converts a bounded requirement into source changes, tests and validation evidence. In the case study this role was performed primarily by Codex. Codex worked against explicitly identified repository/worktree/branch baselines

and was used in both the private Salesforce product tree and the public website tree.

For Salesforce product work, implementation and repository inspection were separated from human-controlled deployment and release decisions. A successful Codex report did not constitute deployment proof.

6.4 Role-separated adversarial-challenge role

The challenger examines the candidate for defects, unsupported claims, scope drift and missing evidence. In the case study Claude and Claude Code were used in this role. Claude Code could challenge repository and source state; web-facing Claude review could challenge the public/customer-facing surface and publication artefacts.

This separation increased the chance of finding a different class of error, but it did not create third-party assurance. Findings remained recommendations until the human owner adjudicated them against evidence and scope.

6.5 Automated evidence role

Automated tests, validators, package/build checks, static checks, checksum calculations and HTTP/live checks supplied reproducible observations. Automation could prove only the condition actually exercised. It could not decide whether a requirement was authorised or whether residual risk was acceptable.

6.6 External-source and live-boundary verification

Official documentation, standards bodies, research publishers and the public website supplied external evidence. These sources were deliberately kept separate from private repository evidence. An official Salesforce or NIST page can support a claim about platform guidance or a standard; it cannot prove the current state of private code. Conversely, source inspection cannot prove that a public page is live or indexed.

6.7 Role allocation is replaceable; authority rules are not

The method is vendor-neutral even though the case study names the tools actually used. Another project could allocate the same roles to different products or to human reviewers. What must survive a tool change is the separation among authority, contribution, challenge and evidence.

7. The task contract

A strong task contract reduces ambiguity before the model begins work. It also makes review more objective because the candidate can be compared with an authorised specification rather than an inferred intention.

7.1 Required elements

Element	Required content
Requirement anchor	Issue, requirement or decision identifier.
Repository identity	Exact repository, worktree and relevant source boundary.
Starting baseline	Branch, expected commit and required clean/dirty-state condition.
Objective	The smallest complete outcome.
In scope	Exact files, behaviours or artefacts authorised.
Out of scope	Explicitly prohibited changes and deferred work.
Security boundaries	Secrets, data, authorisation and remediation restrictions.
Evidence requirements	Tests, validators, runtime checks and artefact integrity.
Stop conditions	Wrong repository, dirty unrelated work, missing source, divergent baseline or absent authority.
Commit and publication rules	Commit message, push policy, review gate and live verification.
Final report	Evidence required to close the issue.

7.2 Why exclusions matter

An inclusion list defines the intended result; an exclusion list protects the product from plausible but unauthorised elaboration. In AI-assisted work, exclusions are particularly important because models tend to complete patterns. A removed card may be replaced because the layout appears incomplete. A newly available slot may acquire an unverified security claim. Explicit exclusions prevent design tidiness from becoming product scope.

7.3 Stop conditions as controls

A stop condition is not a failure of productivity. It is a control against operating on the wrong state. Typical stop conditions include:

- expected commit not present;
- wrong repository or worktree;
- unrelated dirty changes;
- missing authoritative source;
- inability to verify a security-sensitive claim;
- required owner decision absent;
- deployment target unavailable.

The agent should report the precise blocker and preserve the repository state. A permitted exception – for example, reviewing committed HEAD while unrelated work-in-progress remains excluded – must be explicit and evidence-bound, not inferred.

8. Evidence classes and permitted claims

8.1 Evidence is a ladder, not a binary label

Evidence class	Meaning	Permitted claim
Proposed	A design or change has been suggested.	May be considered; not implemented.
Source-established	The behaviour is visible in reviewed source.	Implemented in the reviewed source; runtime not implied.
Test-observed	A controlled automated test exercised the behaviour.	Observed under the stated fixture.
Runtime-observed	The behaviour occurred in a reviewed execution.	Observed in that execution and environment.
Release-validated	The exact candidate passed the defined release gates.	Validated for the stated candidate and prerequisites.
Live-verified	The deployed artefact was checked after publication.	Present and functioning at the checked live boundary.
Supported	The project accepts the behaviour as part of its product contract for stated environments.	Supported only within the declared scope and prerequisites.
Expected	The behaviour is required by design but not yet evidenced at any higher class.	May be stated only as an expectation or release-gate requirement, never as implemented or validated behaviour.
No conclusion available	The project does not hold reliable evidence for a project-level conclusion.	No positive or negative implementation, validation or support claim.

The case-study product's evidence-confidence model is defined in the companion technical paper, Evidence Semantics and Scanner Orchestration.[18] This delivery ladder reuses that progression for project claims, adds Proposed before implementation evidence and Live-verified after release validation, and deliberately names the no-conclusion project state No conclusion available. It does not reuse the subscriber-facing label Not assessed at that layer. The distinction is intentional: the companion paper governs product evidence semantics; this paper governs what the delivery project may claim about its own work. Delivery claims are classified on this ladder and retained in the private delivery evidence register; no entry from that register is published here.

8.2 Evidence cannot be silently promoted

Source-established behaviour does not become release-validated because the code appears straightforward. A successful local build does not prove a hosted page is live. A package version does not prove clean-org installation. A runtime result in one organisation does not establish edition-wide compatibility.

8.3 Claim classes are not product vocabulary

The ladder in this section classifies claims made by the project about its own work. It is not the vocabulary the product renders to subscribers. One concrete pair is enough to show the boundary: project-level No conclusion available means the delivery project cannot support a conclusion about its own work; product-level Not assessed means the question was outside the assessed run or scope. The full product presentation model is defined in the companion technical paper.[18] The two layers share one invariant: absence of evidence is never promoted into a favourable conclusion.

9. Security and data boundaries

9.1 Least information to the model

AI assistance should operate on sanitised and task-relevant context. The default prohibited set should include:

- access and refresh tokens;
- session identifiers;
- client or consumer secrets;
- passwords and private keys;
- certificate bodies;
- raw sensitive headers;
- production customer evidence;
- personal data not required for the task;
- raw security diagnostics that increase attacker usefulness.

9.2 Repository and context boundaries

The agent should receive access only to the repository, worktree and evidence set required for the task. In the case study, product source, website source and governance were distinct boundaries. Cross-tree reasoning was allowed only when the task required it; cross-tree editing was not assumed.

A model was not deliberately supplied with access or refresh tokens, session identifiers, private keys, certificate bodies, customer evidence, raw attacker-useful diagnostics or autonomous remediation authority. Where a task needed platform behaviour or research context, sanitised summaries or authoritative external sources were preferred to raw operational data.

9.3 No autonomous remediation

Read-only operation is a product-scope decision, not a property the delivery method can confer. What the method requires is narrower and absolute: an AI-assisted task may not introduce write-back actions, token revocation, permission removal, API blocking or endpoint modification into a product whose contract excludes them, however useful the addition appears. Such changes are owner decisions requiring explicit authorisation, threat modelling and their own release evidence.

9.4 Prompt and transcript retention

Project governance should assume that prompts and tool outputs may persist. Sensitive values should therefore be excluded before interaction rather than relying on deletion afterwards. When raw diagnostics are needed, they should remain in a controlled engineering channel and be replaced with bounded categories in retained product or publication artefacts.

9.5 External research and live-web boundary

External research should use authoritative sources for platform, security, standards and legal-adjacent claims. Search snippets and secondary commentary can identify leads, but load-bearing statements should be checked against primary documentation or research.

The case study also used web access for a different purpose: checking the deployed public website and publication URLs. These observations prove only the public boundary checked at that time. They do not prove the repository from which the site was built, just as repository evidence does not prove that a cache, search engine or hosted page has updated.

10. Role-separated adversarial challenge and human adjudication

10.1 Challenger objective

The challenger is not asked whether the implementation “looks good”. It is asked to falsify specific claims:

- Does the diff remain within the authorised issue?
- Is the repository/worktree/commit actually the one named by the task?
- Can a prohibited state still occur?
- Does the test prove the claim it is cited for?
- Has a local observation been described as live or universal?

- Has customer wording exceeded the product evidence?
- Has a security boundary been weakened for convenience?

10.2 Challenge without scope capture

A role-separated adversarial challenge can identify valuable defects and still recommend unauthorised work. Each finding should therefore be classified:

- accepted defect;
- valid but already tracked;
- deferred improvement;
- unsupported assertion;
- regression risk;
- new owner decision required;
- leave alone.

This classification protects the project from replacing its roadmap with a reviewer’s imagination.

10.3 Model/provider separation is not third-party assurance

Using a different provider or model for challenge can reduce direct self-review correlation, but it does not make the review independent in the professional assurance sense. The challenger may share public sources, assumptions or framing with the implementer. The project therefore describes the control as role-separated adversarial challenge and reserves “independent” for genuinely separate external assurance or replication.

10.4 Resolving model disagreement

When models disagree, the human should not choose the more confident answer. The disagreement should be decomposed into factual propositions and checked against current owner decisions, source, tests, runtime evidence or authoritative external material. Where evidence remains incomplete, the result stays unresolved rather than being forced into consensus.

10.5 Human visual and professional judgement

Some acceptance criteria are not reducible to automated checks. Layout balance, public tone, diagram readability and whether a page appears credible to a customer require human judgement. A manual visual gate is legitimate evidence when it is explicit, bounded to defined surfaces and recorded separately from automated proof.

11. Case study: Security Observatory

11.1 Project context

Security Observatory is a Salesforce-native security evidence application operating under the bounded-write product contract defined in the companion technical paper, Evidence Semantics and Scanner Orchestration.[18] Its architecture combines Lightning Web Components, Apex orchestration, scanner families, Salesforce data and setup sources, selected Tooling API evidence and sanitised product-owned records. The product deliberately excludes automatic security remediation and the retention of secrets or attacker-useful raw values.

The delivery context concentrates multiple high-consequence responsibilities in a small project: Salesforce architecture, security design, package engineering, evidence semantics, documentation, a public website, licensing and release governance. That concentration made the project a useful bounded environment in which to develop and challenge the method described here.

11.2 Actual tool and authority allocation

The methodology is role-based, but the case study used the following concrete allocation:

Delivery role	Case-study implementation	Authority boundary
Product owner, architect, adjudicator and release authority	Luca Pacini	Owns scope, security boundaries, public/legal wording, risk acceptance, visual acceptance and release.
Orchestration, synthesis and primary-source research	ChatGPT/GPT	May structure tasks, compare evidence and research external claims; does not become repository truth or release authority.

Bounded implementation and source inspection	Codex	Edits authorised source, creates tests/validators and reports evidence against the named baseline; no implicit product-policy or release authority.
Role-separated adversarial challenge	Claude / Claude Code	Challenges technical, customer-facing and publication claims; findings require human adjudication and are not third-party assurance.
Automated evidence	Salesforce tests, package/build checks, repository validators, static/site builds, checksums and HTTP checks	Proves only the exercised condition on the exact candidate or checked surface.
External/live evidence	Official web sources, public site, hosted documents and owner-controlled search/indexing tools	Verifies external claims or live state; cannot prove private implementation state.

11.3 Requirements and governance as the control plane

The project moved deliberately away from conversational intent as the authoritative state. Numbered requirements, bounded GitHub issues, explicit statuses and retained owner decisions became the control plane. Typical states included Done, Partial, Not done, Blocked and Deferred, each with a meaning tied to evidence rather than optimism.

This matters because chat history and model memory are convenient but lossy. Before implementation guidance, review or status reporting, the relevant canonical requirement and current repository/live state were re-read. When they disagreed with a remembered narrative, the current evidence won.

The resulting delivery contract was approximately:

Owner decision → canonical requirement/governance → bounded issue → implementation → automated proof → role-separated adversarial challenge → human adjudication → correction if required → release gate → deployment/publication → live verification → governance close-out.

11.4 Multi-repository and multi-worktree orchestration

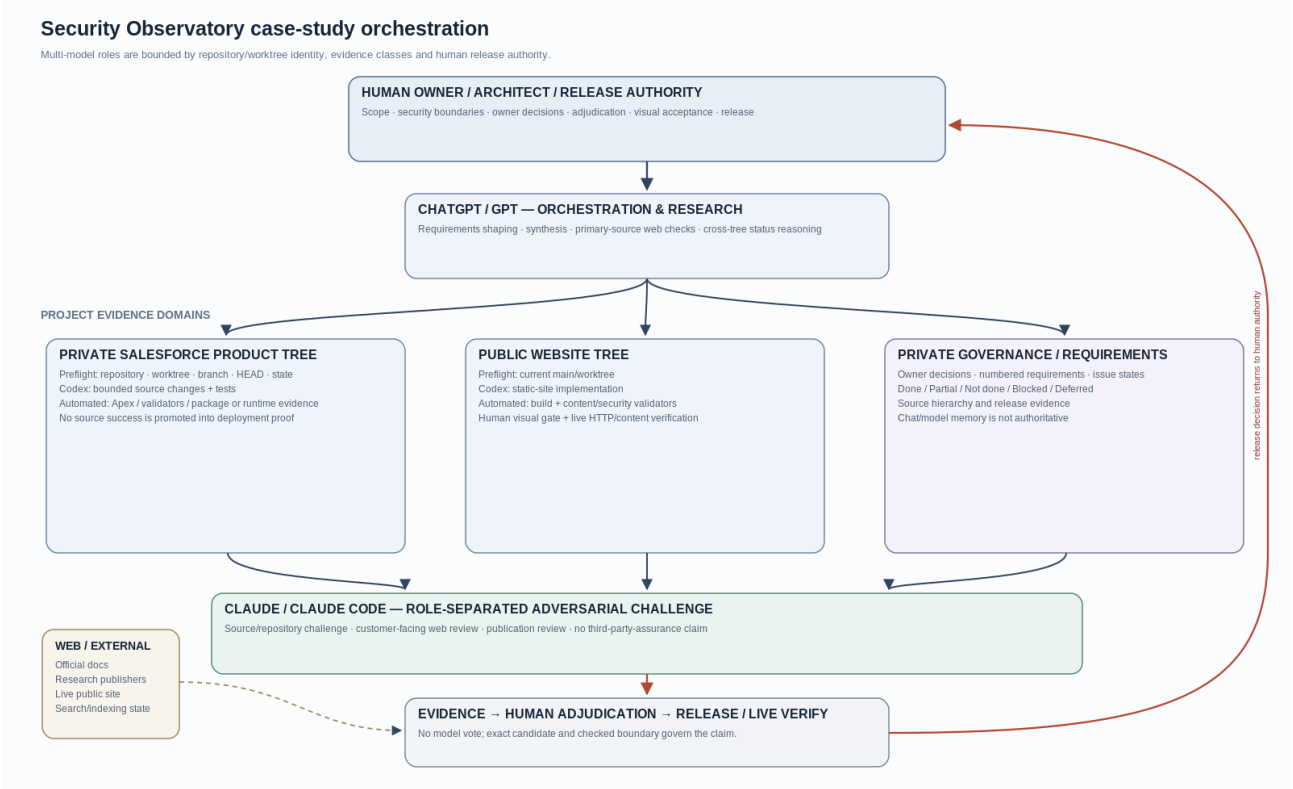

Figure 3. The case-study orchestration model. Human authority sits above three project evidence domains. ChatGPT/GPT supports orchestration and research; Codex implements against explicitly identified product and website baselines; Claude/Claude Code challenges source and public surfaces; automated and web evidence return to human adjudication before release.

I treated repository identity as part of the evidence boundary, not as incidental Git hygiene. Before a material task or review, I required the expected repository, worktree, branch, HEAD and clean/dirty state to be established. A model can reason perfectly about the wrong branch and still give me a useless answer.

One review showed why this matters. My preflight stopped me because the named product worktree was on a different branch from the expected integration branch. The committed HEAD did resolve to the intended baseline, but I only knew that after I checked. The match was not evidence until I verified the commit. I deliberately restricted the audit to committed HEAD so

unrelated work-in-progress could not leak into the review. On a separate website task, I initially treated a parent directory as if it were the site repository. It was not. I resolved the real worktree before I accepted any source conclusion.

That became a hard rule for me: baseline provenance is a delivery control. If I depart from the expected tree, I must say exactly why, identify the commit I am reviewing, and state what is included and excluded. A correct answer about the wrong software is still wrong.

11.5 Salesforce product delivery path

For product changes, the typical path was:

1. the owner defined the security and customer outcome;

2. the requirement was registered in canonical governance with explicit exclusions;

3. Codex inspected and modified only the authorised product baseline;

4. Apex tests, repository validators, package/build checks or runtime checks supplied the required evidence class;

5. Claude/Claude Code challenged implementation, failure handling and public claims;

6. ChatGPT/GPT helped decompose disagreements and cross-check external Salesforce documentation where necessary;

7. the owner adjudicated findings and retained final release authority;

8. deployment, package or runtime proof remained a separate gate and was not inferred from source success.

The process was deliberately conservative about security evidence. A unit test could justify a test-observed claim; it could not establish clean-org installation, edition-wide compatibility or a production runtime state unless those boundaries were actually exercised.

11.6 Public website and publication delivery path

The same method was applied to the public FortMilo website rather than treating the site as “just content”. A typical website/publication sequence was:

1. owner decision and governance issue;

2. verification of the current public-site baseline;

3. Codex implementation against the website repository;

4. local static build and content/security validators;

5. role-separated challenge from source and, where useful, the public/customer-facing web surface;

6. human adjudication and visual acceptance;

7. publication;

8. live HTTP/content verification and artefact identity checks;

9. owner-controlled search/indexing steps where relevant;

10. governance close-out only after the live boundary matched the approved candidate.

This surface produced useful counterexamples. A source tree can be correct while the live page is stale; a file can exist in GitHub while the landing page still points to stale content; and a search engine can remain out of date after both source and hosted site are correct. Each is a different evidence boundary.

11.7 Web research and external-source verification

Web access served two distinct roles in the case study.

First, ChatGPT/GPT and Claude used authoritative web sources to verify current platform behaviour, standards, research papers and publication details. Primary sources were preferred for load-bearing claims. This prevented a stale citation or remembered product behaviour from silently becoming technical fact.

Second, web access was used to inspect the public site and hosted documents after publication. That observation could establish that a URL returned content or that customer-facing wording was present. It could not establish the private source history behind it.

Evidence domain	Can support	Cannot by itself prove
Private repository/source	Current reviewed implementation, tests and baseline identity	Live publication, search indexing or customer runtime behaviour

Automated validation	Behaviour exercised by the test/build/check on the candidate	Unexercised requirements, visual quality or universal support
Official external source	Standards, vendor documentation and published research	Private implementation state
Public/live web	What a reader can retrieve at the checked URL and time	Which private source or local build produced it
Human adjudication	Accepted scope, residual risk and release decision	Technical behaviour not otherwise evidenced

11.8 Complete trace: evidence-first website release

A completed publication-surface trace demonstrates the whole loop without relying on private product details.

Human-owned intent. The owner approved a specific evidence-first homepage outcome, exact trust boundaries and prohibited overclaiming.

Task contract. Governance captured the permitted copy, layout, source tree and validation rules.

Implementation. Codex reapplied the design on the current website baseline rather than forcing a stale earlier change set onto newer source.

Automated proof. The static build and validators checked the required copy, navigation, metadata, images and prohibited language.

Role-separated challenge. Claude reviewed the public/customer-facing result and source evidence for credibility, scope and claim strength.

Human adjudication. Reviewer preferences that changed scope were separated from genuine defects; the owner accepted the final visual and public wording.

Release and live verification. The site was published and checked at the live boundary. Search/indexing work remained a distinct owner-controlled activity rather than being inferred from publication.

Close-out. Governance was updated only after the replacement path was accepted and the superseded work was explicitly closed.

11.9 Non-confirming product cases: evidence failures found by challenge

I do not trust a governance method whose case study contains only successes, so I am keeping the four product failures recorded against EV-01 through EV-04 in the paper.

EV-01 and EV-02 were false-zero defects. In the Connected App and Sites/Experience paths, a failed source acquisition could still reach zero-shaped evidence. My tests were green because I had not asserted those failure paths. I had therefore built false-clean paths and proved them green.

EV-03 exposed a different problem: scanner exceptions could lose a useful bounded cause before reaching the operator. The fix preserved only approved failure categories and safe next actions; unknown conditions still fail closed rather than retaining raw exception text.

EV-04 exposed a retained-evidence read problem. A bounded read failure could collapse family state into an unexplained Unknown presentation. The correction separated read layers, preserved readable retained evidence where possible and added a bounded read-limitation warning. The forced degraded branch is test-observed; I did not manufacture a live failure merely to strengthen the evidence label.

The method did not prevent these defects. The challenge stage caught them. I treated the findings as correctness blockers and froze favourable evidence-class upgrades while the affected paths remained unresolved. After remediation and named validation, the companion technical paper recorded the transition against EV-01 through EV-04. The value of the control was not that I avoided the mistakes; it was that the process stopped the claim until the evidence changed.

11.10 Publication-process failures and hardening

The documentation and website work forced me to harden the process again. I have kept these failures and near misses in the first person because they are the clearest evidence of what the controls are for:

What I got wrong or nearly got wrong	Consequence	Control I added or strengthened
I targeted a wrong or stale worktree assumption	I could have described the wrong software correctly	Repository/worktree/branch/HEAD preflight and explicit committed-HEAD exception rules
I lost the authoritative editable source for a publication artefact	Reconstructing it from extracted text could have introduced silent drift	Controlled reconstruction and a delta ledger tied to the public artefact

I wrote export-safety wording broader than the tests supported	Public wording implied more protection than the enumerated evidence established	I narrowed the claim to the reviewed trigger prefixes and control-character handling
I allowed a reviewer recommendation to reach beyond authorised scope	Review could have become roadmap capture	I classify findings as defect, deferred work, new owner decision or leave alone
I could not prove visual quality with automation	Passing validators could still leave a visibly poor publication	I retained an explicit human visual-acceptance gate
I changed a publication artefact and its landing page out of sequence	The artefact, landing-page link and live URL temporarily diverged	I split artefact integrity, landing-page linkage and live-URL verification into separate release checks
I found that publication wording no longer matched current product vocabulary	Carrying it forward unchanged would have misrepresented the current implementation	I compare publication wording with the current implementation and record material wording changes explicitly

I do not present these episodes as proof that the method was mature from the outset. They show the opposite: I changed the controls when reality contradicted my assumptions. A method that cannot absorb its own failures is theatre.

11.11 Human factors: too many windows, one tired operator

The least glamorous component in the orchestration model was also the most privileged: me. At several points, the most sophisticated failure mode in the stack was a tired human with too many nearly identical browser tabs, chat windows and terminals open.

I occasionally pasted a Claude challenge into the Codex window, or a Codex implementation report into the Claude window. Nothing dramatic or mysterious happened. Each system responded sensibly to the context I had given it; I had simply given the right material to the wrong role. I also ran correct diagnostic or validation commands in the wrong worktree before noticing that the path, branch or HEAD did not match the task I thought I was performing. The commands were not necessarily wrong. Their evidential context was.

The immediate cost was mostly small delays, repeated checks and a few unglamorous minutes working out which window I had just instructed. The professional lesson was more important: human context switching is part of the threat model. A role-separated process is not truly separated if the operator can silently route one role’s material into another role’s session or attach a result to the wrong repository baseline.

I therefore strengthened the operating discipline in five ways:

1. every task begins by naming the role, repository, worktree, branch, expected HEAD and permitted write state;

2. pwd, repository-root, branch, HEAD and clean/dirty-state checks precede material work;

3. only one tree is treated as write-enabled at a time, while review sessions remain explicitly read-only unless authorised otherwise;

4. copied material is labelled by destination and purpose rather than pasted as an unmarked block; and

5. when fatigue makes the active context uncertain, publication and write operations stop rather than relying on memory.

The rule that emerged is simple enough to remember late at night: when I have to ask myself which window is which, the release gate is already closed. The controls converted operator slips into visible rework rather than accepted evidence or a silent cross-tree release.

11.12 Human-only decisions

Several decisions remained deliberately outside model authority even when models supplied analysis:

- scope freezes and backlog priority;
- security exceptions or any change to the bounded-write product contract;
- acceptance or rejection of reviewer recommendations;
- legal and public wording;
- visual acceptance;
- package/release readiness;
- publication of public artefacts;
- residual-risk acceptance and final release.

11.13 Context deliberately withheld from AI systems

The project did not deliberately provide AI systems with secrets, access or refresh tokens, session identifiers, private keys, certificate bodies, raw customer evidence, raw attacker-useful diagnostics or autonomous remediation authority. Sanitised summaries, safe reason categories, repository source and public/official documentation were preferred where they were sufficient for the task.

11.14 What the case study does not prove

The case study does not prove that the method is faster than conventional delivery, that it generalises to larger teams, that model diversity provides independence, or that a human gate is sufficient assurance. It shows a narrower result: one technically experienced owner used AI intensively across implementation, synthesis, research and challenge while

keeping authority, evidence classes and release gates explicit – and the controls sometimes stopped the project's own favourable claims.

12. Measuring value without productivity theatre

12.1 Productivity evidence is context-dependent

Published evidence on AI-assisted programming is heterogeneous in design, population and result, and later evidence can change the interpretation of earlier studies.

Cui and colleagues report approximately 26 per cent more completed tasks across three randomised field experiments covering 4,867 developers, with the authors reporting a standard error of 10.3 percentage points.[14] METR, by contrast, studied sixteen experienced open-source maintainers working on their own mature repositories and found that early-2025 AI tools made the sampled tasks take approximately 19 per cent longer.[15]

Repository-level work by Song, Agarwal and Wen reports a 5.9 per cent increase in project-level code contributions associated with GitHub Copilot use.[16] Separate mechanism analyses estimate a 3.4 per cent increase in developer coding participation and a 2.1 per cent increase in individual productivity.[16] Those estimates describe different margins and are not arithmetic components that should be expected to reproduce the 5.9 per cent project-level estimate exactly. The same study reports that

coordination time increased by 8 per cent because of more code discussion; a cost direction, not a benefit.[16] Its normalised

issue and bug measures were statistically unchanged, which the authors interpret cautiously rather than as universal proof of unchanged code quality.

METR's February 2026 follow-up is an important caution against freezing the 2025 result into a timeless conclusion. The organisation reported that participants were likely obtaining more benefit from newer tools, but severe selection effects and measurement problems made the new experiment only very weak evidence for the size of any speed-up. Raw estimates suggested faster completion for some later-study groups, but the confidence intervals crossed zero and the study authors explicitly warned against treating those estimates as reliable current effect sizes.[24]

These studies are not commensurable. They differ in population, task type, tooling generation, outcome measure and identification strategy. The correct inference is not an average of the headline figures but a recognition that context and date are part of the evidence.

A small project should therefore avoid a headline claim such as "AI made delivery 30 per cent faster". The more defensible question is whether the method improved the project's effective assurance-adjusted capacity, defined here as the volume of work that passes the project's evidence gates and release authority within a period, at a residual-risk level the owner has explicitly accepted. Generated output that fails a gate, requires rework after challenge, or cannot be released without weakening a control contributes nothing to that measure.

12.2 Recommended measures

The table below is proposed instrumentation, not applied measurement: no values are reported in this paper, and Section 16.3 records that absence as a limitation.

Measure	What it tests	Misuse to avoid
Lead time from authorised issue to live verification	Delivery throughput	Ignoring increased rework or review debt.
Rework after role-separated adversarial challenge	First-pass quality and task clarity	Treating all challenge-driven change as model failure.
Scope deviation count	Task-contract effectiveness	Rewarding under-delivery merely because scope stayed narrow.

Measure	What it tests	Misuse to avoid
Escaped defect count	Release effectiveness	Comparing releases of very different risk.

Evidence-gate failure count	Whether controls are active	Treating blockers as wasted time.
Human review time	Governance cost	Assuming lower review time is always better.
Unsupported-claim corrections	Epistemic quality	Hiding corrected claims to improve the metric.
Sensitive-data incidents	Security boundary effectiveness	Counting only confirmed disclosure and ignoring near misses.
Live verification failures	Local-to-live fidelity	Omitting infrastructure or external-service factors.

12.3 Quality-adjusted capacity

The useful outcome is not maximum generated code. It is the amount of correctly scoped, reviewable and releasable work that a small project can complete without increasing residual risk beyond its accepted boundary.

13. Anti-patterns

13.1 The giant prompt

A single prompt combines product design, implementation, refactoring, documentation, deployment and review. The model loses task boundaries, and the human cannot attribute evidence to a specific decision.

13.2 The self-certifying agent

The same agent implements the change, writes the tests, interprets the results and declares release readiness without role-separated adversarial challenge.

13.3 Model voting

Several models are asked the same question, and the majority answer is treated as truth. Correlated training data and shared framing make this a weak assurance mechanism.

13.4 Test-green absolutism

Passing tests are treated as proof that the requirement, architecture and public wording are correct. Tests only prove the behaviours they actually exercise. Benchmark research illustrates the point at scale: resolution rates on suites built from real repository issues depend on the tests those repositories happened to contain, and a change that satisfies the available tests has not thereby satisfied the requirement.[17] The false-zero cases in Section 11.9 are the same failure mode in miniature.

13.5 Validator weakening

An existing guard is relaxed because a new implementation fails it. The correct sequence is to determine whether the guard is obsolete, whether the requirement changed, and whether both source and validation must change together.

13.6 Unbounded tidy-up

The agent performs unrelated refactors, dependency upgrades or design additions while implementing a narrow issue. Review becomes larger and causal attribution weaker.

13.7 Secret-rich context

The model receives production logs, credentials or raw customer evidence because it is convenient. Sanitisation is attempted after disclosure rather than before interaction.

13.8 Publication before evidence

A product page, whitepaper or diagram claims support, safety or release status before the exact candidate has been verified.

13.9 Artificial freshness

Dates, statuses or search metadata are updated merely to appear current, without a substantive change. This damages trust and makes retained evidence less meaningful.

13.10 Endless review churn

The project continuously changes sound work to satisfy every reviewer preference. Review becomes a source of scope instability rather than assurance.

13.11 The six-window cockpit

The owner keeps several near-identical model chats, browser tabs and terminals open and relies on memory to know which role and tree each one represents. A correct instruction, review or command then lands in the wrong context. This is not a model hallucination; it is operator-induced context collision. The output is unusable as evidence even when its content is technically sound because provenance is part of the evidence. The remedy is not a promise to concentrate harder. It is visible role and tree labelling, one write-capable context at a time, a baseline preflight before every command sequence, and a stop rule when fatigue makes provenance uncertain.

14. Proportionate adoption for a small project

14.1 Minimum viable control set

A small project can adopt the method with six artefacts:

1. a canonical requirements file;

2. bounded issues with exclusions and stop conditions;

3. a protected source repository with identifiable baselines;

4. automated build, test and validation commands;

5. a role-separated adversarial challenge record;

6. a release and live-verification record.

14.2 Risk-based tailoring

Low-risk documentation edits may require source diff, link checks and owner review. A change to authorisation, sensitive data, package installation or public security claims requires stronger evidence, role-separated adversarial challenge and exact-candidate validation.

14.3 When the method should become heavier

The governance model should be expanded when:

- customer or production data is processed;
- the product can change customer security configuration;
- regulatory obligations apply;
- multiple teams or suppliers contribute;
- the model can deploy autonomously;
- the cost of failure materially exceeds the owner’s ability to absorb it.

At that point, formal threat modelling, segregation of duties, independent security testing, change-management controls and organisational AI governance may be required.

15. Relationship to established guidance

15.1 What the cited instruments govern

The method is consistent with, but does not replace, established guidance – and it is important not to narrow those instruments incorrectly.

The NIST AI Risk Management Framework addresses organisational risk management for organisations that create, provide, acquire, deploy or operate AI systems across their lifecycle.[1] Its Generative AI Profile applies that framework to risks specific to generative AI products, services and uses.[2] That makes the NIST material relevant to a project

that uses generative AI as an engineering contributor, even when the delivered software is not itself an AI product. NIST does not, however, prescribe the small-project task-by-task operating loop defined in this paper.

The NIST Secure Software Development Framework organises secure-development practice around organisational readiness, protection of development environments and artefacts, production of secure releases, and vulnerability response.[3] NIST SP 800-218A extends secure-development guidance to AI model development and systems that incorporate models, with responsibilities spanning model producers, system producers and acquirers.[4]

The United Kingdom’s Guidelines for Secure AI System Development treat security as a lifecycle concern from early design decisions through implementation, release and ongoing operation of AI systems.[5] The AI Cyber Security Code of Practice sets

baseline security expectations across the AI lifecycle for organisations involved in building, deploying and operating AI.[6] The Software Security Code of Practice sets fourteen voluntary principles for organisations that develop or sell software, aimed at establishing a baseline of software security and resilience.[8] UK AI-assurance guidance frames assurance as a structured way to gather and communicate evidence about whether AI-related claims are trustworthy.[7]

15.2 Adjacent bodies of practice

Adjacent traditions supply components of the present method without assembling the same operating loop. Goal Structuring Notation provides a structured graphical notation for claims, arguments and supporting evidence.[19] The NCSC's Assurance Principles and Claims for the Software Security Code of Practice uses claims-argument-evidence structures to decompose software-security principles into evidence-backed claims.[25]

SLSA v1.2 is the closest supply-chain neighbour to the baseline-provenance control in this paper because its Source Track covers the creation, review and management of source revisions and provides source provenance that consumers can inspect.[20] SLSA attests properties of source revisions and build artefacts to consumers; the control here establishes which repository, worktree, branch and commit a task or review is actually operating against before work begins. The two controls are complementary rather than interchangeable. The in-toto framework likewise binds supply-chain steps and artefacts to verifiable metadata.[21] ISO/IEC 42001 defines requirements for an organisational Artificial Intelligence Management System.[22]

The OWASP GenAI Security Project publishes security guidance for systems that embed or use generative AI; its current 2025 Top 10 resource is titled OWASP Top 10 for LLM Applications 2025.[23] This paper addresses a different but adjacent problem: governing a generative model as a contributor to conventional software delivery, whether or not the delivered system contains AI.

15.3 The gap this method occupies

The instruments reviewed here govern AI risk at organisational/system level, secure software development, AI-system cyber security, assurance arguments, supply-chain provenance or GenAI application threats. They provide many of the principles used by this paper.

Within the reviewed set, however, they do not specify the same lightweight project-level procedure for a small team in which generative models contribute research, coding and review while repository baseline identity, explicit exclusions, evidence classes, role-separated adversarial challenge, human adjudication and live/publication verification are combined in one delivery loop.

That is the bounded gap this method attempts to address. It is not a claim that no comparable practice exists anywhere, nor that the method is a new standard.

16. Limitations and research agenda

16.1 Single-project derivation

The method is derived from one project with a technically experienced human owner. It may depend on architectural skill, security judgement and the ability to challenge model output. Results may differ where the human lacks domain expertise.

16.2 The assessor is not independent

I designed this method. I am its only practitioner to date, and I adjudicated every disputed finding in this case study. That is a real limitation, and I am not going to dress model or provider separation up as independence. Every favourable project-specific statement in Section 11 should therefore be read as self-attested unless I identify a public artefact or external source that supports it. Stronger standing requires independent replication: another project, another owner and an assessor with no stake in the method. None has yet occurred.

16.3 No controlled productivity comparison

The case study does not contain a controlled non-AI baseline, and none of the measures proposed in Section 12.2 has yet been instrumented with reported values. The paper cannot quantify speed, cost or defect reduction attributable to AI assistance.

16.4 Model and tool evolution

Coding-agent capabilities, context windows, tool controls and data-governance terms change rapidly. Controls must therefore be expressed at the role and evidence level rather than tied permanently to one product.

16.5 Human oversight is itself fallible

The method vests final authority in human adjudication, but empirical work on algorithm-in-the-loop decision-making shows that human overseers frequently fail to correct model error, calibrate poorly to model accuracy, and can introduce their own

biases;[13] related work shows that explanations can increase rather than reduce automation bias, [12] and that different models or providers may still share public data, reasoning patterns, project framing or common failure modes. The case study exposed a more mundane version of the same limitation: fatigue, context switching, wrong-window pastes and wrong-tree commands (Section 11.11). Those episodes caused small delays rather than material changes because the surrounding controls stopped them, but they show that operator state belongs in the threat model. Placing a human at the gate is therefore a necessary control, not a sufficient one. The quality of adjudication is bounded by the adjudicator's expertise, attention and incentives, and a project whose owner cannot technically evaluate a challenged claim retains the ceremony of oversight without its substance.

16.6 Private evidence

Some case-study evidence is retained in private repositories. Public readers cannot independently reproduce every source-established statement. The paper therefore distinguishes self-attested project evidence from public standards and published research.

16.7 Future research

Useful future work includes:

- controlled comparison of evidence-gated and informal AI-assisted delivery;
- measurement of review correlation across models;
- task-contract patterns for different risk classes;
- methods for proving context sanitisation;
- reliable machine-readable evidence registers;
- assurance of autonomous agent tool use;
- the relationship between human expertise and effective challenge quality.

17. Practical adoption checklist

Before the task

- Is the owner decision explicit?
- Are the repository, worktree, branch, expected HEAD and clean/dirty-state conditions known?
- Are the model, browser and terminal windows visibly labelled by role and tree?
- Am I sufficiently alert for a write-capable task; if not, should this remain read-only or stop?
- Are inclusions and exclusions stated?
- Are secrets and sensitive data excluded?
- Are stop conditions defined?
- Is the evidence required for completion known?

During implementation

- Is the agent changing only authorised source?
- Before each command sequence, does the terminal still show the expected root, branch, HEAD and status?
- Am I sending this instruction to the intended role, rather than merely the nearest open window?
- Are new claims tied to source or evidence?
- Are validators being strengthened rather than bypassed?
- Are blockers reported without speculative workarounds?
- Is the working tree state preserved?

Before release

- Has automated proof passed on the exact candidate?
- Has role-separated adversarial challenge occurred against the correct baseline?
- Has the owner adjudicated findings?

- Has visual or customer-facing acceptance occurred where needed?
- Are public statements no stronger than the evidence?
- Are artefact identity and checksums recorded where relevant?

After release

- Was the live boundary checked?
- Does the deployed artefact match the committed candidate?
- Were requirements and issue status updated?
- Are residual limitations and deferred work recorded?
- Were stale or abandoned model and terminal sessions closed so they cannot be reused accidentally?
- Has monitoring been assigned without pretending it is completed implementation?

18. Conclusion

AI can materially expand the range of work a small software project can attempt. That expansion is useful only when delivery authority remains distinct from delivery capacity. A model may generate code, tests, prose, research and critique. It cannot inherit responsibility for scope, security, truth or release.

Evidence-gated AI-assisted delivery treats every transition as a claim requiring proof. The task contract defines authority and exclusions. Baseline identity establishes which software is actually being changed or reviewed. Automated checks provide reproducible evidence. Role-separated adversarial challenge exposes assumptions without being misrepresented as third-party assurance. Human adjudication resolves uncertainty. Release and live verification establish the state of the exact artefact. Governance close-out preserves what was done, what remains unproven and what must not be claimed.

The Security Observatory case study adds a practical observation: multi-model orchestration is useful only when multi-tree provenance is equally explicit. ChatGPT/GPT, Codex and Claude/Claude Code could contribute different forms of capacity, but repository/worktree identity, primary-source verification and human release authority were the controls that kept those contributions attached to the correct evidence boundary.

The method is intentionally conservative. It does not promise that AI will make every project faster, nor that several models will eliminate error. It offers a more defensible proposition:

A small project can use AI intensively without allowing AI fluency, model consensus or automation success to replace human authority and evidence.

Appendix A. Reusable bounded task contract

Requirement anchor

State the authoritative issue, requirement or owner decision.

Repository and baseline

- Repository:
- Worktree:
- Branch:
- Expected starting commit / HEAD:
- Active role and window label:
- Write-enabled or read-only:
- Required clean/dirty-state condition:
- Permitted committed-HEAD-only exception, if any:

Objective

State the smallest complete outcome in customer or system terms.

In scope

List authorised behaviours, files and artefacts.

Out of scope

List prohibited changes, deferred items and adjacent work that must not be pulled into the task.

Security and data boundaries

State prohibited secrets, data classes, authorisation changes, remediation actions and external egress.

Validation

List exact build, test, static, security, visual, package, runtime and live-verification requirements.

Stop conditions

State the conditions under which the agent must stop without modification.

Commit and publication

State commit rules, push authority, release gate and live-verification sequence.

Final report

Require starting and ending state, changed-file tree, validation evidence, unresolved risks and final repository status.

Appendix B. Minimal delivery evidence register

Template only. No entries are published in this paper; the completed delivery evidence register is retained privately.

Fields: Claim; Scope; Evidence class; Artefact; Limitation; Next gate; Owner decision.

References

1. National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023. https://doi.org/10.6028/NIST.AI.100-1

2. National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024. https://doi.org/10.6028/NIST.AI.600-1

3. National Institute of Standards and Technology, Secure Software Development Framework (SSDF) Version 1.1, NIST SP 800-218, 2022. https://doi.org/10.6028/NIST.SP.800-218

4. National Institute of Standards and Technology, Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile, NIST SP 800-218A, 2024. https://doi.org/10.6028/NIST.SP.800-218A

5. National Cyber Security Centre, Guidelines for Secure AI System Development, 2023. https://www.ncsc.gov.uk/collection/guidelines-secure-ai-system-development

6. Department for Science, Innovation and Technology, AI Cyber Security Code of Practice, 2025. https://www.gov.uk/government/publications/ai-cyber-security-code-of-practice

7. Department for Science, Innovation and Technology, Introduction to AI Assurance, 2024. https://www.gov.uk/government/publications/introduction-to-ai-assurance

8. Department for Science, Innovation and Technology and National Cyber Security Centre, Software Security Code of Practice, published 7 May 2025; GOV.UK publication page last updated 15 January 2026. https://www.gov.uk/government/publications/software-security-code-of-practice

9. Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J. and Chen, J., “Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study”, ACM Transactions on Software Engineering and Methodology, 34(8), Article 218, 2025. https://doi.org/10.1145/3716848

10. Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X. and Zhou, D., “Large Language Models Cannot Self-Correct Reasoning Yet”, International Conference on Learning Representations (ICLR), 2024. https://openreview.net/forum?id=IkmD3fKBPQ

11. Levy, A., Agrawal, M., Satyanarayan, A. and Sontag, D., “Assessing the Impact of Automated Suggestions on Decision Making: Domain Experts Mediate Model Errors but Take Less Initiative”, Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, ACM, 2021. https://doi.org/10.1145/3411764.3445522

12. Vered, M., Livni, T., Howe, P. D. L., Miller, T. and Sonenberg, L., “The Effects of Explanations on Automation Bias”, Artificial Intelligence, 322, 103952, 2023. https://doi.org/10.1016/j.artint.2023.103952

13. Green, B. and Chen, Y., “The Principles and Limits of Algorithm-in-the-Loop Decision Making”, Proceedings of the ACM on Human–Computer Interaction, 3 (CSCW), Article 50, 2019. https://doi.org/10.1145/3359152

14. Cui, Z. K., Demirer, M., Jaffe, S., Musolff, L., Peng, S. and Salz, T., “The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers”, Management Science, 2026. https://doi.org/10.1287/mnsc.2025.00535

15. METR, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity, 2025. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

16. Song, F., Agarwal, A. and Wen, W., “The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot”, arXiv:2410.02091, submitted 2024, version 3 revised 14 May 2026. https://arxiv.org/abs/2410.02091

17. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O. and Narasimhan, K., “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?”, The Twelfth International Conference on Learning Representations (ICLR), 2024. https://arxiv.org/abs/2310.06770

18. FortMilo, Evidence Semantics and Scanner Orchestration, 2026. https://fortmilo.co.uk/documents/evidence-semantics-and-scanner-orchestration.pdf

19. Assurance Case Working Group, Goal Structuring Notation Community Standard, Version 3, Safety-Critical Systems Club, 2021. https://scsc.uk/gsn-standard

20. Open Source Security Foundation, Supply-chain Levels for Software Artifacts (SLSA), Version 1.2, 2025. https://slsa.dev/spec/v1.2/

21. Torres-Arias, S., Afzali, H., Kuppusamy, T. K., Curtmola, R. and Cappos, J., “in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes”, Proceedings of the 28th USENIX Security Symposium, 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

22. International Organization for Standardization and International Electrotechnical Commission, ISO/IEC 42001:2023 – Information technology – Artificial intelligence – Management system, 2023. https://www.iso.org/standard/42001

23. OWASP GenAI Security Project, OWASP Top 10 for LLM Applications 2025, 2025 edition. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

24. METR, We are Changing our Developer Productivity Experiment Design, 24 February 2026. https://metr.org/blog/2026-02-24-uplift-update/

25. National Cyber Security Centre, Software Security Code of Practice – Assurance Principles and Claims (APCs), 7 May 2025. https://www.ncsc.gov.uk/guidance/software-security-code-of-practice-assurance-principles-claims

Each URL is printed in full so the references remain usable in printed copies.

Publication and licensing

Author: Luca Pacini, trading as FortMilo

Public location: United Kingdom

Case-study product: Security Observatory

Competing interests. The author is the founder and sole proprietor of FortMilo. Security Observatory is the case-study product, and this paper forms part of the FortMilo publication surface. No external funding was received for this work.

AI-use disclosure. Generative AI systems contributed research, synthesis, implementation work in the case-study repositories, document production and role-separated adversarial challenge under the authority model described in this paper. The author adjudicated their outputs and retains sole responsibility for the published claims.

Source/package availability for the case-study product is defined in the companion technical paper, Evidence Semantics and Scanner Orchestration.[18]

Copyright 2026 Luca Pacini. The FortMilo-authored text and figures of this paper are licensed under the Creative Commons Attribution 4.0 International licence (CC BY 4.0). This licence does not relicense third-party trademarks, standards, publications or cited material, which remain subject to their own rights and licences.

Salesforce is a trademark of Salesforce, Inc. Security Observatory is independently developed by FortMilo and is not endorsed by Salesforce, Inc. Product and company names used to describe ChatGPT/GPT, Codex, Claude and Claude Code remain the property of their respective owners; their inclusion identifies the case-study tooling only.

This paper is not a certification, industry standard, third-party assurance report or universal productivity claim. It is a proposed method derived from one bounded case study and the cited guidance and research.