Governed Human-AI Collaboration Across the Software Development Lifecycle: A Process Framework and Its Reference Implementation

Tomás dos Santos Taborda
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
tomassantostaborda@tecnico.ulisboa.pt

Abstract

Generative Artificial Intelligence and Large Language Models have moved from autocomplete assistants to active contributors across the Software Development Life Cycle, yet adoption has outpaced method. A majority of functionally correct AI-generated code has been reported to carry security defects, and measured delivery bottlenecks shift toward review rather than disappearing. A Systematic Literature Review conducted for this work, following Kitchenham's guidelines over 646 retrieved articles, confirms this pattern and identifies the absence of structured, repeatable guidance for integrating AI tools into professional software workflows as the field's central unresolved problem. This dissertation addresses that gap through the Design Science Research Methodology. It proposes a process-oriented framework for governed Human-AI collaboration that treats AI systems as first-class collaborators whose contributions must be explicitly bounded, validated and traced: six lifecycle phases with phase-specific quality gates, responsibilities expressed as functions rather than job titles, three governance metrics, and a persistent, versioned specification that anchors every generated artefact to its originating requirement. The framework is instantiated as Apex, a working web application that operationalises each phase, gate and artefact against a real project management backend. Evaluation combines a demonstration of the framework, standardised instruments assessing perceived usability and cognitive workload, semi-structured interviews with practitioners and an Agile expert, and an analytical assessment against defined success criteria. Ten of twelve mapped governance constructs are realised in the reference implementation as specified, one as a proxy and one partially; across thirteen task-based sessions, perceived cognitive workload is low (a Raw NASA-TLX aggregate of 21.5 out of 100) while perceived usability sits below the published corpus norm (a System Usability Scale mean of 56.73, grade D); the framework was further demonstrated end to end on a separate product, delivered over nine days and growing from six to nine epics and fifty-four stories, surfacing a Testing Gate passable by a suite that had stopped measuring the system it tested, while its demonstration within an independent partner organisation did not occur.

Keywords

Artificial Intelligence, Large Language Models, Software Development Life Cycle, Design Science Research, Human-AI Collaboration, Software Process, Governance

1 Motivation and Problem Statement

Adoption of AI coding assistance is close to universal while trust in it is not, and the two are moving in opposite directions. The 2025 Stack Overflow Developer Survey reports 84% of developers using or planning to use AI tools, up from 76% the previous year, while the share trusting the accuracy of that output fell from 40% to 29%, with more respondents actively distrusting AI output (46%) than trusting it (33%) [28]; industry telemetry places daily use higher still, at around 90% of developers [8]. That distrust is empirically warranted. Benchmarking a leading coding agent on real-world tasks, Zhao et al. found 61% of solutions functionally correct but only 10.5% secure [32], and an assessment across more than one hundred models found 45% of generated samples introduced an OWASP Top 10 vulnerability, rising to 72% for Java, with newer and larger models improving on functional correctness while security performance stayed flat [30]. Productivity gains are similarly conditional rather than automatic. An early controlled experiment reported a 55.8% speed-up, though it remains unpublished [24], while a randomised controlled trial of sixteen experienced developers on their own mature repositories found they took 19% *longer* with AI assistance while estimating afterwards that it had made them 20% faster [5]; at organisational scale, telemetry from 1,255 teams associates high AI adoption with 98% more pull requests merged but 91% longer review times and no significant correlation with delivery outcomes [10].

> *The value of AI assistance in software engineering is conditional on the process that surrounds it, rather than being a property of the model alone. Capability without governance redistributes effort rather than reducing it, and can degrade quality attributes that are expensive to recover later.*

A Systematic Literature Review conducted for this work (Section 2) confirms this reading directly. While the corpus consistently reports meaningful benefits from AI and LLM use, it equally consistently reports that organisations lack established, clear methodologies to shape the responsible use of these tools within professional software development [4, 9, 13, 16]. Existing methodologies do not close this gap. Waterfall, Scrum, Kanban and the Agile Manifesto were designed for development environments in which every contributor is human, and lack explicit structures for governing AI participation, bounding the context a model may act upon, or allocating responsibility over generated artefacts [31].

The research problem addressed in this work is the absence of structured, repeatable guidance and standardised evaluation practices for integrating AI and LLM-based tools across the

SDLC while preserving human oversight, software quality and security, and organisational accountability.

The problem has four dimensions that any solution must satisfy: *placement*, fixing for every SDLC phase where AI participates, for what purpose and where its participation ends; *governance*, the quality gates and human-in-the-loop practices that keep AI contributions traceable, validated and accountable; *realisation*, establishing which of those constructs a working system can enforce and which resist it; and *practitioner support*, establishing whether the result is usable and cognitively sustainable in a real setting. This dissertation designs, instantiates, demonstrates and evaluates a framework addressing all four.

2 Research Methodology

Two complementary methodologies structure this work. A Systematic Literature Review, following Kitchenham's three-phase protocol (planning, conducting, reporting), retrieved 646 articles and synthesised the corpus around five research questions covering benefits, consequences, integration methodologies, stakeholder perspectives, and adopting industry sectors (RQ1-RQ5, reported in full in the dissertation's Systematic Literature Review chapter). The Design Science Research Methodology proposed by Peffers et al. structures the design, construction, demonstration and evaluation of the artefact across its six activities: problem identification, definition of objectives, design and development, demonstration, evaluation, and communication. The two methodologies are sequential rather than parallel. The review's findings are the evidence base the problem statement and the framework's design principles are grounded against, and DSRM's iteration loop is not strictly linear, since feedback gathered while constructing the reference implementation fed back into the design of the framework itself.

3 Related Work: Findings of the Literature Review

The review's findings on methodology (RQ3) are the most direct grounding for this dissertation's proposal. Current industry practice is not built around full automation but around control. The most prevalent documented methods, In-Context Learning, Prompt Engineering and Chain-of-Thought Prompting, all aim to shape and constrain AI behaviour before code is generated, and the Human-In-The-Loop paradigm keeps developers directly responsible for reviewing, validating and approving all outputs. Multiple emerging frameworks, including Agile-based human-AI collaboration models, indicate the field is actively searching for structured and repeatable integration patterns, but no universally accepted approach has yet emerged [1]. Findings on benefits (RQ1) and consequences (RQ2) sharpen the tension the framework must resolve. Reported benefits extend across the entire SDLC rather than code generation alone, but they are inseparable from risks that directly undermine the same benefit, chiefly reduced code quality, security exposure, and eroded traceability. Findings on stakeholder opinion (RQ4) converge across developers and organisations on the same trade-off: enthusiasm for productivity gains alongside concern about reliability and hallucination, and a shared insistence that AI augment expertise under a controlled level of trust rather than substitute for it.

The closest published methodological precedent is Amazon Web Services' AI-Driven Development Lifecycle, which organises work into Inception, Construction and Operations, replaces sprints with short cycles called Bolts, and replaces epics with Units of Work [2]; its vendor-reported productivity outcomes are unverified and carry no evidential weight here, but its process vocabulary is adopted directly. The closest engineering practice is spec-driven development, in which a persistent, versioned specification rather than the source code itself is the artefact both developers and AI agents work against, tracked by Thoughtworks' Technology Radar as an emerging technique [29] and recently framed in a not-yet-peer-reviewed preprint as a response to a productivity-reliability paradox in AI-augmented development [11].

4 Proposed Framework

The framework is a method in the Design Science Research sense: a set of phases, hats, artefacts and decision rules prescribing how a team organises AI participation across the SDLC, stated at the level of process rather than tooling so it can be assessed and adopted independently of any single tool stack. It deliberately excludes model training or fine-tuning, model architecture, and infrastructure-level MLOps, since these govern the lifecycle of a model as a deployed asset, whereas this framework governs the lifecycle of software a model helped write.

4.1 Design Principles

Seven principles, each grounded in the evidence of Sections 1 and 3 rather than asserted as preference, govern every design decision: Human-in-the-Loop by Design; AI as Augmentation, Not Substitution; Process-Level Governance of AI Usage; SDLC-Wide and Phase-Aware Integration; Functional, Not Positional, Accountability; Risk-Proportional Validation and Control; and Spec-Anchored Continuity. The last of these instantiates spec-driven development (Section 3) rather than inventing it, and is the mechanism argued in Section 4.4.

4.2 Lifecycle Phases

The framework organises work into six phases, preceded by a Strategic Alignment stage that decides whether a Unit of Work should exist at all. Figure 1 shows the phase sequence and its three governed loop-back routes into Maintenance; Table 1 summarises, for each phase, the permitted role of AI, the accountable hat, and the control that must pass before exit; each phase consumes the output of the one before it. The model departs from AI-DLC's three-phase Inception/Construction/Operations split at two points, both argued from the evidence in Section 1 rather than announced: a decoupled Testing phase, because tests derived by the same model from the same specification in the same working context confirm internal consistency rather than external correctness, and a distinct Maintenance phase with a diagnostic firewall, because a post-deployment signal may be a genuine defect, an unspecified request, or evidence the specification itself was wrong, and treating all three as defects is what erodes a specification over time. Maintenance's exit is always a governed loop back into a named earlier phase rather than a patch applied in place, which is the structural difference from a Waterfall lifecycle with a maintenance tail appended to it.

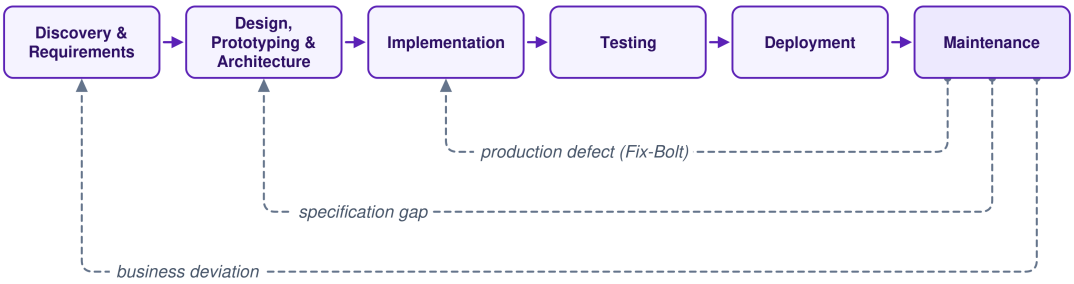


Figure 1: The framework's six lifecycle phases and the three governed loop-back routes into Maintenance: a production defect, a specification gap, and a business deviation.

Table 1: The six lifecycle phases: condensed from the full phase-mapping table of the framework chapter.

Phase	Role of AI	Exit control
Discovery & Requirements	Drafts stories, compiles approved scenarios into Gherkin	Discovery Gate (the Trio)
Design, Prototyping & Architecture	Proposes UI structure and a technical specification	Design Gate (Design Lead, Tech Lead)
Implementation	Generates and refactors code inside a Bolt	Implementation Gate (Developer)
Testing	Drafts end-to-end BDD scripts from locked Gherkin	Testing Gate (QA)
Deployment	Generates infrastructure delta only when required	Deployment Gate (Deploying hat)
Maintenance	Reconstructs a narrow context to diagnose one signal	Diagnostic firewall, governed loop-back

4.3 Roles as Hats

Every role named is a hat, a function the framework requires to be performed rather than a person who must be hired; one individual may wear several, and on a small team an unfilled hat's responsibility falls to whoever performs the adjacent work. The Trio (Product Owner, Tech Lead and Design Lead hats) is inspired by the Squad concept popularised in the Spotify Model [18], cited alongside its own documented limits: it was a description of one company at one point in time rather than a blueprint [19], and a first-hand account names matrix ambiguity and autonomy without accountability as failure modes it never fully resolved even internally [20]. The framework answers both directly, the first through the Tech Lead hat's pre-Bolt architectural lock, the second by separating logistics authority (Bolt Master hat) from technical authority (Tech Lead hat) and attaching every gate to a named function; the treatment of team cognitive load as an explicit design constraint [27] is the more rigorous vocabulary for the same argument, and the Product Manager hat sits outside the Squad closer to the single-threaded-leader pattern Amazon adopted once its earlier structure did not survive scale [7].

4.4 Governance Mechanisms and Quality Gates

A Bolt is the smallest execution iteration the framework recognises, the implementation of one task inside one User Story, measured in hours; the term is adopted from AI-DLC [2], and what the framework adds is the obligation sequence around it, since a short cycle without a fixed reference is simply a fast way to accumulate divergence. Spec-Anchored Continuity requires every AI-generated artefact to carry forward the context of previously validated phases through a persistent, versioned specification split into a global layer (architecture, technology-stack decisions, security policy) and a local layer (a story's locked Gherkin and technical specification); the distinction argued for this mechanism is that narrowing a model's working context is a control rather than a convenience, since an unbounded context is the specific condition under which a model produces a confident and substantively wrong change [32]. Five quality gates, each with a named owning hat, a specific body of required evidence, and a defined consequence on failure, govern Discovery, Design, Implementation, Testing and Deployment (Figure 2 shows the Deployment Gate as implemented in Apex); failure never discards work, it returns it to a named earlier activity with the evidence attached. Three governance metrics, Bolt Cycle Time, Context Traceability Rate, and AI Defect Escape Rate, are expressed in vocabulary continuous with the NIST AI Risk Management Framework's Govern-Map-Measure-Manage loop [23] and ISO/IEC 42001's plan-do-check-act vocabulary [17], a claim about shared language only; no conformity assessment against either standard has been performed. Grouping stories at the Epic level rather than requirement by requirement is independently supported by evidence of higher expert-rated correctness and completeness than requirement-level decomposition [26]; unlike AI-DLC, the framework therefore leaves the Epic, a team's optional sub-grouping of a Unit of Work's stories, in place rather than replacing it.

4.5 Replacing Sprint Cadence, and Positioning Against Alternatives

The framework replaces sprints and sprint backlogs with continuous flow, leaving the Epic an optional grouping rather than replacing it, on the ground that a fixed-length iteration is a mechanism for a world in which implementation is the binding constraint, and the evidence in Section 1 shows that premise no longer holds

where AI assistance is heavily used. The constraint moves to review rather than disappearing [5, 10]. Four alternatives are engaged directly rather than by omission: no framework at all, correct for a short-lived prototype but contradicted by the flat security-versus-capability trend already cited; AI-DLC adopted unmodified, whose two gaps in build/test coupling and post-deployment classification motivate the divergence rather than a rejection of it; unmodified Scrum or SAFe cadence, undermined specifically by developers misjudging their own speed under assistance [5]; and metrics-only governance, the strongest of the four, answered by the distinction that metrics measure whether governance is working while containing no mechanism themselves for what a model may act upon or where a post-deployment signal is routed [12].

5 Reference Implementation: Apex

Apex is not positioned against a coding agent. A coding agent accelerates the writing of code, whereas Apex governs whether the code that results still answers the requirement that prompted it, by carrying a stable specification identifier from every generated artefact back to its origin. It is a split full-stack application, a Next.js/TypeScript frontend and a Python/FastAPI backend over a typed REST API, organised around six architectural properties that carry framework constructs rather than merely support them:

- **Spec-anchored storage** keeps all workflow state as versioned Markdown context files plus a machine-readable story index, under a path keyed first by tenant instance and then by project, so a hosted account and a private deployment of the same tool remain isolated tenants for one user, with access mediated by a single service that enforces per-request isolation.
- **Repository and design grounding.** Two further context files ground Phase 2-6 generation in the actual project: a `repomix` pack of the connected repository, rebuilt after every push, and the connected Figma files, fetched through the guarded proxy.
- **Single-writer concurrency.** A deliberate model pins the backend to one replica and one worker by default, since the story index and configuration live on shared storage guarded only by a process-local lock, with an optional coordination layer promoting the locks to cross-replica equivalents once configured.
- **Proxy-first external integration** routes every outbound call to a user-configurable host through a server-side proxy that validates HTTPS and non-private resolution before forwarding, closing server-side request forgery, with one deliberate browser-direct exception whose credential never reaches the backend.
- **Project management adapter.** Built against one tool first and extended behind the same interface to a second, with a third started but left incomplete, this is where the framework's tool-independence claim (Section 4) is tested directly: it holds at the level of operations, generalising authentication and identifier shape, while failing at the level of guarantees, since neither concurrency semantics nor identifier type can be made uniform and each gap is mitigated at application level instead.
- **Provider abstraction** detects the model vendor from the model identifier and maps every provider error to a typed exception at one boundary, so no other part of the codebase branches on vendor.

Of twelve framework constructs mapped construct by construct against Apex's mechanisms, ten are realised as specified (Table 2). One, AI Defect Escape Rate, is realised only as a *proxy*: the system cannot observe a production defect directly or attribute one to generated logic, so what is measured is the proportion of deployed stories carrying a project-management issue confirmed as a bug, linked to the story and dated after deployment. One, spec-drift detection, is only *partially realised*: built for the whole specification, it cascaded because the technical specification was one project-wide file in which an edit could not be attributed to a story, one real edit flagging 44 stories at once, and was removed; it was revived for the functional specification alone, where each scenario already carries a stable identifier naming its story, and remains open for the technical specification, which has no equivalent per-item ownership.

Table 2: The twelve mapped framework constructs and the realisation outcome each reached in Apex.

Construct	Outcome
Six lifecycle phases	Realised
Quality gates	Realised
Spec-Anchored Continuity	Realised
Stable specification identifiers	Realised
Interactive Bridge	Realised
Consistency Factor	Realised
Bolts and Fix-Bolt	Realised
Bolt Cycle Time	Realised
Server-side reconciliation of generated output	Realised
Context Traceability Rate	Realised
AI Defect Escape Rate	Proxy
Spec-drift detection	Partial

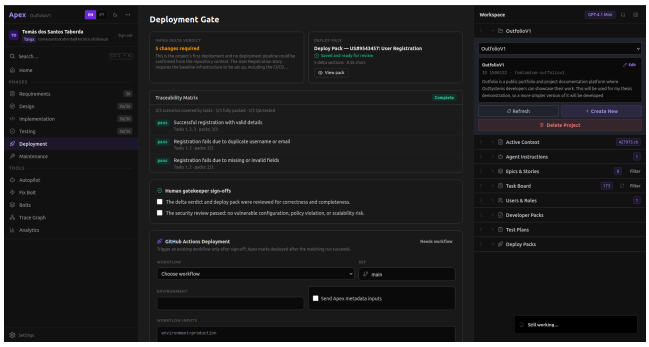


Figure 2: Apex's Deployment Gate: the delta verdict and deploy pack, the traceability matrix reconciled server-side against the parsed specification, and the two human gatekeeper sign-offs required before deployment proceeds.

One of the ten realised constructs is a finding rather than a clean success. Bolt Cycle Time was first implemented and displayed at story level under the framework's task-level name, a proxy mistaken for the construct and caught only by comparing the implementation against its written definition.

Spec-Anchored Continuity is where the translation from construct to code is least one to one. The framework mandates only the two-layer split by lifetime and leaves the number and granularity of

files unspecified; Apex resolves that latitude with a fixed, named taxonomy, three global files (technology stack, decisions, constraints) and five local ones (concept, functional specification, technical specification, design bundle, runtime specification), maintained as cumulative, project-wide documents rather than one artefact locked per story. That is coarser than the framework requires, and it is an implementation liberty rather than an obligation the framework imposes; it is also the same granularity gap that spec-drift detection exposed, since a per-project technical specification cannot support the per-story attribution the framework's own assumption presumes.

Feedback gathered while building Apex fed back into the framework itself, not only the tool. A reverted change is recorded for the same reason a change kept is, since it is evidence about the design.

- **DevOps Alliance role removed** (framework-level). This earlier role, external to the team and holding deployment approval and infrastructure-security review, was removed once feedback observed that a role defined only by what it supervises can be assumed by the process and never staffed; its two responsibilities were reassigned to whoever performs the deployment.
- **Design/technical sign-off merged** (framework-level). Phase 2's two separate sign-offs were merged into one evidence-based gate, still owned jointly by the Design Lead and Tech Lead hats, once the same observation showed that role-gating a gate on a small team either blocks the work or is satisfied twice by the same person.
- **Design-conflict detector** (tool-level). Narrowed, then removed entirely.
- **Spec-drift flagging** (tool-level). Narrowed after the whole-specification cascade described above.
- **Authentication for a design integration** (tool-level). An authentication flow requiring an operator-registered application and backend-held secret was reverted for conflicting with Apex's zero-configuration deployment model.

This record came from five practitioners consulted informally during construction, with no fixed guide, recording or demographic data, so it is evidence that the method was revised in response to use rather than evidence about the distribution or strength of practitioner opinion.

6 Demonstration, Evaluation Design, and Results

The evaluation strategy is fixed independently of its outcome. The framework is demonstrated in two settings that fail in different directions, Apex's own construction and a separate product built against the framework end to end, with a third setting, a partner organisation, pursued and abandoned before it produced a deployed increment. The System Usability Scale [6] and the Raw NASA Task Load Index [14, 15] evaluate the instantiation (Apex, practitioner support), while semi-structured interviews with practitioners and with an Agile and Scrum expert, together with an analytical assessment against stated success criteria, evaluate the framework itself (placement, governance). A favourable usability score for the tool is deliberately not treated as evidence that the framework is sound; Table 3 states which instrument evaluates which artefact, and the two artefacts are evaluated by different means for that reason.

Table 3: Evaluation instruments, artefact evaluated, and problem dimension addressed.

Instrument	Artefact	Dimension
Demonstration in a real project	Framework + instantiation	Realisation, practitioner support
System Usability Scale	Instantiation	Practitioner support
Raw NASA-TLX	Instantiation	Practitioner support
Apex UX questionnaire	Instantiation (2 items score the framework)	Practitioner support
Practitioner/expert interviews	Framework	Placement, governance
Analytical assessment	Framework	Placement, governance

Thirteen sessions were completed (P0 to P12), all returning all four instruments with no withdrawal. Participants were recruited through the author's own professional and personal connections rather than an open call, and the sample is junior and mixed:

- **Experience.** Three students with no professional experience, three under a year, seven with one to three years, nobody exceeding three.
- **Current role.** Six students, four developers, one AI engineer, two entirely outside software development.
- **Prior AI exposure.** Spanned the full scale, from daily use to none.
- **Session language.** Nine chose English, four Portuguese, with two administration mismatches between declared language and completed instrument reported as recorded rather than corrected.

The System Usability Scale mean across all thirteen is 56.73, a grade D on Sauro and Lewis's curved scale [25] and between *OK* and *Good* on Bangor et al.'s adjective scale [3], below the published corpus mean of approximately 68; the Portuguese subgroup is reported as indicative only given the validated Portuguese instrument's own weak inter-rater agreement [22]. Table 4 summarises both instruments.

Table 4: Perceived usability (SUS) and workload (NASA-TLX), summary statistics.

Metric	Value
SUS mean, all 13 (SD 18.50, median 60)	56.73, grade D
SUS mean, English (n=10)	60.50, grade D
SUS mean, Portuguese (n=3)	44.17, grade F
Usability / Learnability subscale	59.86 / 44.23
NASA-TLX grand aggregate (78 admin.)	21.5 / 100
NASA-TLX highest subscale	Mental demand, 29.1
NASA-TLX Frustration subscale	16.7

The Usability and Learnability sub-scales [21] separate sharply, and free-text responses converge on the same reading. Difficulty concentrates at first contact with a dense interface rather than distributing through use. The Raw NASA-TLX profile is not flat.

The two heaviest tasks by aggregate are the design step and the implementation task-breakdown step, the two carrying the largest volume of generated content to review, and the lightest is an export step that generates nothing. Cognitive sustainability therefore reads favourably while usability does not, and the two do not contradict. The tool asks little of a practitioner who already knows where they are, and does too little to get them there.

The author-designed UX questionnaire's most diagnostic pair, whether generated content was trusted enough for real use and whether reviewing it took less effort than writing it, both return a median of 4 with no participant below 3, the calibrated-trust signature the pair was built to detect; because nothing was observed during the unmoderated sessions, the design cannot rule out acceptance without scrutiny in any individual case, and one participant's free text reports exactly that state directly. The study's one planned measurement of gate comprehension, a cross-read against a guaranteed deployment refusal, was lost when that refusal was redesigned into a normal deployment step before the study ran, so gate legibility, the framework's most distinctive mechanism, is the construct this evaluation has least evidence about.

Table 5 rates the framework against its own five design criteria, fixed before any result existed; no criterion is rated *not met*, and the information is in the three *partially met* rows, each naming a specific shortfall rather than a general reservation.

Table 5: Analytical assessment against the framework's five design criteria, fixed in advance.

Criterion	Rating	Shortfall named
Alignment with SDLC phases	Met	Instantiation required no phase merged, split or skipped
Human-in-the-loop support	Met	Every gate needs a specification write and human acceptance; generation alone cannot pass it
Clarity	Partial	Spec-drift detection exposed an unstated per-story specification-scope assumption
Completeness	Partial	AI Defect Escape Rate is a proxy; spec-drift detection is incomplete for the technical specification
Governance of AI activity	Partial	AI-attributable defects observable only through a proxy; a governance surface reported confidently from stale evidence

These results answer the instantiation half of practitioner support and, through the analytical assessment and the interviews, feed placement and governance. The separate demonstration product, Outfolio, was in fact carried from Strategic Alignment to a deployed increment over nine calendar days, growing from a pre-committed six epics to a delivered nine, fifty-four stories, none abandoned, with all three of the framework's named loop-back routes exercised more than once. Its complete implementation is published at https://github.com/thomastabs/OutfolioV1, with the deployed product itself available at https://outfolio-v1.vercel.app/. Its own governance exports return:

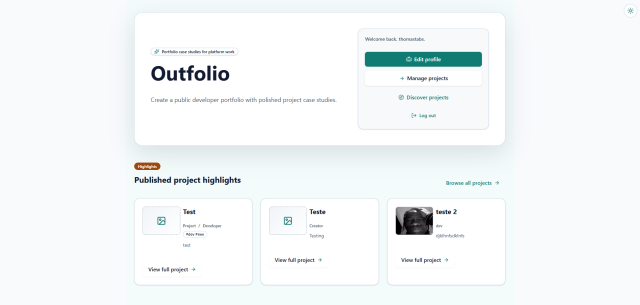


Figure 3: Outfolio's dashboard: the deployed product built for this setting's lifecycle demonstration.

- Context Traceability Rate: 92.6 per cent
- Spec Conformance mean: 98.65 per cent
- AI Defect Escape Rate: 1.85 per cent

alongside two further findings: a Testing Gate shown passable by a suite that had stopped measuring the system it tested, and the stale-context pre-flight failure named above recurring on two further occasions. What is reported above is therefore evaluation evidence now joined by an observed demonstration rather than standing apart from one, with the remaining boundary, that both are the work of a single operator in a single setting, carried forward explicitly in §7.

7 Discussion and Threats to Validity

Table 6 states the verdict on each dimension of the research problem, drawn directly from §5 and §6.

Table 6: Verdict on each dimension of the research problem.

Dimension	Verdict
Placement	Holds unambiguously at phase level: the six-phase decomposition survived instantiation with no phase merged, split or skipped. Qualified by one construct, spec-drift detection, carrying a per-story specification-granularity precondition the framework never states.
Governance	Separates by strength. Traceability and validation are strongest, since gates are enforced against specification content and generated coverage claims are reconciled server-side rather than trusted as reported. Human-in-the-loop support is met structurally. Accountability is weaker: AI Defect Escape Rate is instantiable only as a proxy, and a governance surface (the Phase 5 deployment pre-flight check) was observed reporting confidently from a stale repository-context file.
Realisation	Answered in full in §5: ten of twelve mapped constructs realised as specified, one as a named proxy, one partially realised with the resisting part identified.
Practitioner support	Splits without contradiction: cognitive sustainability is favourable (NASA-TLX aggregate 21.5/100, frustration among the lowest subscales), while usability is not (SUS grade D, below the published corpus mean; Portuguese subgroup at grade F).

Three qualifications refine the table's verdicts rather than sit inside it. For placement, an Agile expert who had not designed the framework read its hats and phase structure back in interview as a genuine departure from closed-role methodology, corroborating the

phase-level verdict independently; boundaries are unambiguous at phase level without yet being unambiguous at construct level. For governance, several participants' free text converged on the same stale-evidence surface independently of the recorded defect. For realisation, the finding is what building the framework revealed rather than the construct count alone.

These answers carry a boundary that must be stated plainly rather than allowed to blur. Placement, governance and realisation are grounded in Apex's own construction, in the completed Outfolio demonstration, and in the practitioner and expert interviews conducted for this evaluation; none of the three is the same as observing the framework operated, in the field, by a team that did not design it. The one setting that would have supplied that remaining evidence, an independent partner organisation that already managed its own projects in Taiga, the project-management tool Apex supported at the time, was pursued and abandoned. A mid-project migration to a different tool, Plane, delayed the attempt while Apex's adapter for it was built, and repeated outreach afterwards still did not lead to a scheduled engagement. Its absence is a different kind of gap than a small sample, since no amount of additional interviews, or of solo demonstration on a project of the author's own choosing, substitutes for an observed application by a team that did not design the framework. The threats below are accordingly threats to the evaluation together with this carried-forward demonstration limitation, not to the evaluation alone.

- **Construct validity.** Every instrument measures perception, not delivered software quality, and no completion rate, time on task or error count was collected, a deliberate consequence of an unmoderated, unrecorded design rather than an oversight. The sharpest gap is specific to the framework's core mechanism. The study's only planned cross-read on whether a phase gate is legible as a guardrail was lost when the task guaranteeing a gate refusal was redesigned before data collection, so gate comprehension is unmeasured rather than weakly measured.
- **Internal validity.** The author designed both artefacts, wrote the task script and the questionnaire, and is the sole participant in the separate demonstration product, so the same dual role of designer and evaluator runs through both the instantiation and the demonstration; two live defects, an interface-language leak between projects and the stale-context pre-flight check above, were found and interpreted by the author himself, the first only because he recognised its symptom in a participant's free text, and the system was not frozen across the collection window.
- **External validity.** The sample is a convenience sample from one professional network, skewed junior, and the task environment is a single seeded project on one project-management tool that ten of thirteen participants had never used, confounding unfamiliarity with the tool and with the board; the observation window, one session of about an hour, precludes any claim about long-term maintainability or organisational adoption.
- **Conclusion validity.** Thirteen participants across five levels of prior AI exposure support no inferential statistic and none is computed.

8 Conclusion

This dissertation contributes a systematic synthesis of 646 articles on AI and LLM use in software development; a process framework for governed Human-AI collaboration that treats AI as a first-class, explicitly bounded collaborator across six lifecycle phases; Apex, a working reference implementation showing ten of twelve mapped governance constructs realised as specified, one as a named proxy and one partially realised; and an empirical evaluation, standardised usability and workload measurement across thirteen participants, practitioner and expert interviews, and an analytical assessment against criteria fixed in advance, together answering realisation in full and substantial parts of placement, governance and practitioner support.

Read together, the four dimensions answer the problem stated in §1. Structured, repeatable guidance for governing AI participation across the SDLC is achievable, but achieving it in an artefact is not the same as making that artefact effortless to use. Placement holds unambiguously at phase level, qualified by at least one construct carrying a precondition the framework never stated. Governance separates by strength. Traceability and validation are strongest, human-in-the-loop support is met structurally, and accountability is weakest, since AI Defect Escape Rate is instantiable only as a proxy and a governance surface was found reporting confidently from stale evidence. Realisation is answered in full, ten of twelve constructs realised as specified, one as a named proxy, one partially realised; the completed Outfolio demonstration adds that realised does not yet mean reliably realised, since the Testing Gate was shown passable by a suite that had stopped measuring the system it tested, and a destructive command was once issued inside a phase no gate was positioned to catch. Practitioner support splits without contradiction. Cognitive sustainability is favourable while usability is not, a grade D System Usability Scale mean with density and orientation the difficulties participants named most. The framework and its instantiation ask little of a practitioner who already knows where they are, and do too little to get them there.

Three limitations already qualify those answers. All reported evidence concerns early adoption in a single setting: Outfolio is nine days of solo operation by the person who designed both artefacts, on one project and one technology stack; the task-based study is one roughly hour-long session per participant on one seeded project and one project-management tool, drawn from a junior-skewed convenience sample; and the third planned setting, an independent partner organisation, was pursued and abandoned before it produced a deployed increment, so nothing here speaks to sustained, multi-team use. The framework's constructs are also an original synthesis rather than drawn from one validated source, and several grounding sources are recent, vendor-reported, or not yet peer-reviewed.

Per the dissertation's Communication activity (§2), dissemination follows three vehicles: the complete Apex implementation and its documentation, published in full at https://github.com/thomastabs/apex, with the deployed instantiation itself available at https://apex-bolt.com (Outfolio, the separate demonstration product, likewise at https://github.com/thomastabs/OutfolioV1 and https://outfolio-v1.vercel.app/); this extended abstract itself, submitted and assessed alongside the dissertation; and the dissertation itself,

to be presented and defended before an academic examination committee, the structured adversarial examination no instrument in this evaluation supplies.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

During the preparation of this work, the author used AI to improve the wording and clarity of the manuscript. After using this tool, the author reviewed and edited the text as needed and takes full responsibility for the content of the publication. The final manuscript reflects the author's own analysis, results, and conclusions.

This declaration concerns the use of these technologies in the writing process only. The use of generative AI as the object of study of this research, and within the artefacts it produces, is described in Section 4 and Section 5.

References

[1] I. Ahmed, A. Aleti, H. Cai, A. Chatzigeorgiou, P. He, X. Hu, M. Pezzè, D. Poshyvanyk, and X. Xia. 2025. Artificial Intelligence for Software Engineering: The Journey So Far and the Road Ahead. *ACM Transactions on Software Engineering & Methodology* 34, 5 (2025), 1–27. doi:10.1145/3719006

[2] Amazon Web Services. 2025. AI-Driven Development Lifecycle (AI-DLC). Vendor methodology announced at AWS DevSphere 2025. Reported productivity figures are vendor-reported and not independently verified. https://aws.amazon.com/blogs/devops/ai-driven-development-life-cycle/

[3] A. Bangor, P. Kortum, and J. Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies* 4, 3 (2009), 114–123.

[4] L. Banh, F. Holldack, and G. Strobel. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (2025). doi:10.1016/j.infsof.2025.107751

[5] J. Becker, N. Rush, E. Barnes, and D. Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089, METR, preprint, not yet peer-reviewed. https://arxiv.org/abs/2507.09089

[6] J. Brooke. 1996. SUS: A 'quick and dirty' usability scale. In *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland (Eds.). Taylor and Francis, 189–194.

[7] C. Bryar and B. Carr. 2021. *Working Backwards: Insights, Stories, and Secrets from Inside Amazon.* St. Martin's Press, New York, NY.

[8] DORA and Google Cloud. 2025. State of AI-assisted Software Development 2025. Industry research report, not peer-reviewed. https://dora.dev/dora-report-2025/

[9] Mar'i Farhanna, Nerisafitra Paramitha, Marianingsih Susi, Hanjaya Surya, Aisyah Yanti, Primatama Dea, and Sutaji Deni. 2025. Exploring the Use of Generative AI in Software Development: A Preliminary Study. In *E3S Web of Conferences*, Vol. 645. 04002. doi:10.1051/e3sconf/202564504002

[10] Faros AI. 2025. The AI Productivity Paradox Report 2025. Industry telemetry report, not peer-reviewed. Sample of over 10,000 developers across 1,255 teams. https://www.faros.ai/blog/ai-software-engineering

[11] S. E. Farrag. 2026. The Productivity-Reliability Paradox: Specification-Driven Governance for AI-Augmented Software Development. arXiv:2605.01160, preprint, not yet peer-reviewed. https://arxiv.org/abs/2605.01160

[12] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler. 2021. The SPACE of Developer Productivity: There's More to it than You Think. *ACM Queue* 19, 1 (2021), 20–48. doi:10.1145/3454122.3454124

[13] C. Gao, X. Hu, S. Gao, X. Xia, and Z. Jin. 2025. The Current Challenges of Software Engineering in the Era of Large Language Models. *ACM Transactions on Software Engineering & Methodology* 34, 5 (2025), 1–30. doi:10.1145/3712005

[14] S. G. Hart. 2006. NASA-Task Load Index (NASA-TLX); 20 years later. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 50. 904–908.

[15] S. G. Hart and L. E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, P. A. Hancock and N. Meshkati (Eds.). North-Holland, 139–183.

[16] Arshia Hemmat, Mohammadreza Sharbaf, Shekoufeh Kolahdouz-Rahimi, Kevin Lano, and Sobhan Y. Tehrani. 2025. Research directions for using LLM in software requirement engineering: a systematic review. *Frontiers in Computer Science* 7 (2025). doi:10.3389/fcomp.2025.1519437

[17] ISO/IEC. 2023. *Information technology - Artificial intelligence - Management system.* International Standard ISO/IEC 42001:2023. International Organization for Standardization/International Electrotechnical Commission.

[18] H. Kniberg and A. Ivarsson. 2012. Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. Whitepaper, industry-reported, not peer-reviewed. A description of Spotify's organisation at the time of writing, not a prescriptive method. https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify

[19] Leading Complexity. 2023. Summary of Henrik Kniberg's session, about his thoughts on the Spotify Model, in the Leading Complexity programme 2022. Crisp's Blog. Third-party published summary of Henrik Kniberg's session in the Leading Complexity programme 2022, not written by Kniberg himself. https://blog.crisp.se/2023/02/06/leadingcomplexity/leading-complexity-series-summary-1-henrik-knibergs-thoughts-on-the-spotify-model

[20] J. Lee. 2020. Failed #SquadGoals. Practitioner account by a former Spotify employee, industry-reported, not peer-reviewed. https://www.jeremiahlee.com/posts/failed-squad-goals/

[21] J. R. Lewis and J. Sauro. 2009. The factor structure of the System Usability Scale. In *Human Centered Design, HCII 2009, Lecture Notes in Computer Science*, Vol. 5619. 94–103.

[22] A. I. Martins, A. F. Rosa, A. Queirós, A. Silva, and N. P. Rocha. 2015. European Portuguese validation of the System Usability Scale (SUS). *Procedia Computer Science* 67 (2015), 293–300. doi:10.1016/j.procs.2015.09.273

[23] National Institute of Standards and Technology. 2023. *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* Technical Report NIST AI 100-1. National Institute of Standards and Technology, U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

[24] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590, preprint, not published in a peer-reviewed venue. https://arxiv.org/abs/2302.06590

[25] J. Sauro and J. R. Lewis. 2016. *Quantifying the User Experience: Practical Statistics for User Research* (2nd ed.). Morgan Kaufmann.

[26] S. Siddeeq, M. Abbasi, J. Rasku, Z. Zhang, F. Christophe, T. Mikkonen, and P. Abrahamsson. 2026. Epic-Organized vs. Requirement-Aligned Gherkin: An Empirical Evaluation of LLM-Based Acceptance Criteria Generation. Accepted at the International Conference on Software Engineering of Emerging Technologies (SEET 2026), Springer Nature. arXiv:2607.01980. https://arxiv.org/abs/2607.01980

[27] M. Skelton and M. Pais. 2019. *Team Topologies: Organizing Business and Technology Teams for Fast Flow.* IT Revolution Press, Portland, OR.

[28] Stack Overflow. 2025. 2025 Developer Survey: AI. Annual practitioner survey, industry-reported, not peer-reviewed. https://survey.stackoverflow.co/2025/ai/

[29] Thoughtworks. 2025. Spec-driven development. Technology Radar, Vol. 33. https://www.thoughtworks.com/en-us/radar/techniques/spec-driven-development

[30] Veracode. 2025. 2025 GenAI Code Security Report: Assessing the Security of Using LLMs for Coding. Industry report, not peer-reviewed. Assessment of over 100 large language models across Java, Python, C# and JavaScript. https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf

[31] S. Zhang, Z. Xing, R. Guo, F. Xu, L. Chen, Z. Zhang, X. Zhang, Z. Feng, and Z. Zhuang. 2025. Empowering Agile-Based Generative Software Development through Human-AI Teamwork. *ACM Transactions on Software Engineering & Methodology* 34, 6 (2025), 1–46. doi:10.1145/3702987

[32] S. Zhao, D. Wang, K. Zhang, J. Luo, Z. Li, and L. Li. 2025. Is Vibe Coding Safe? Benchmarking Vulnerability of Agent-Generated Code in Real-World Tasks. arXiv:2512.03262, preprint, not yet peer-reviewed. https://arxiv.org/abs/2512.03262