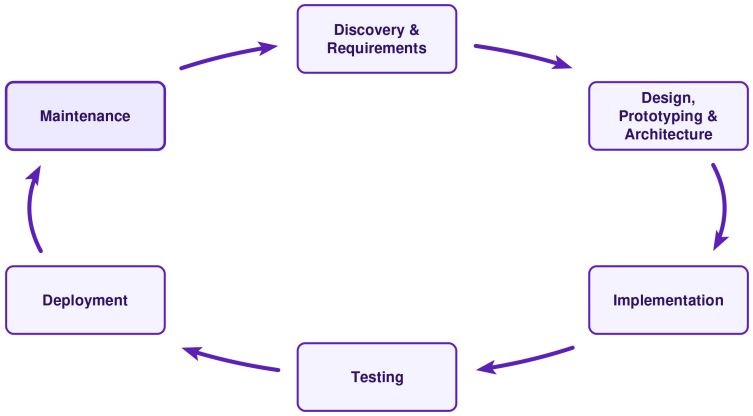
TÉCNICO
UNIVERSIDADE
DE LISBOA



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

Tomás dos Santos Taborda

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Miguel Leitão Bignolas Mira da Silva

Examination Committee

Chairperson: Prof. Helena Isabel de Jesus Galhardas
Supervisor: Prof. Miguel Leitão Bignolas Mira da Silva
Member of the Committee: Prof. Ademar Manuel Teixeira de Aguiar

October 2026

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa. I acknowledge the use of artificial intelligence tools to support the drafting and refinement of this thesis, with all content produced by these tools carefully reviewed, edited, and validated to ensure accuracy and originality.

Acknowledgments

This dissertation, and the experience of completing it over the past year, represents the culmination of my journey as a university student. Considerable dedication, effort, and perseverance were required throughout this process, and I am deeply grateful for the outcome and for everything I learned along the way. This work would not have been possible without the support, encouragement, and guidance of the many people who accompanied me throughout this journey. I would therefore like to express my sincere gratitude to everyone who contributed, directly or indirectly, to the completion of this dissertation.

I would first like to express my deepest gratitude to my dissertation supervisor, Professor Miguel Leitão Bignolas Mira da Silva, for his invaluable guidance, constant support, and availability throughout the entire process. Particularly in the early stages, when I was still uncertain about how to structure and approach the dissertation as a whole, his direction and insight were fundamental in helping me move forward with greater clarity and confidence. His encouragement and guidance played an essential role in the successful completion of this work.

A very special thank you goes to my family and especially to my parents, who have always been the strongest foundation I could have asked for. Their unwavering love, support, patience, and encouragement have guided me throughout my life and were especially important during this demanding final year. I am deeply grateful for the values they have instilled in me, the confidence they have always placed in me, and the constant care that has shaped the person I am today. I feel incredibly fortunate to be their son and will always be grateful for everything they have done for me.

I would also like to express my sincere appreciation to my friends and colleagues. Their encouragement, advice, availability, and light-hearted presence helped me maintain perspective and balance during the most challenging stages of this journey. Their support made this experience not only easier, but also far more enjoyable.

A particularly heartfelt thank you goes to my girlfriend, whom I had the privilege of meeting during this final year. Her support, patience, affection, and companionship had an immeasurable impact on my well-being throughout the completion of this dissertation. Sharing this period of my life with her made it all the more meaningful, and falling in love with her has been one of the most special experiences of my life.

Finally, I would like to thank Hugo de Sousa and his colleagues for generously sharing their real-world corporate perspective on AI-driven development and the use of AI throughout the Software Development Lifecycle. Their practical and theoretical insights into contemporary frameworks, methodologies, and professional practices were highly valuable to the development of this dissertation.

To each and every one of you, thank you.

Abstract

Generative Artificial Intelligence and Large Language Models have moved from autocomplete assistants to active contributors across the Software Development Life Cycle, enabling development in which practitioners express requirements in natural language and supervise the resulting artefacts. Adoption has outpaced method. A Systematic Literature Review over 646 articles confirms substantial benefits alongside recurring risks in reliability, security, traceability and accountability, and identifies the absence of structured, repeatable guidance for integrating these tools into professional workflows as the field's central unresolved problem. Therefore, value is conditional on process, not on model capability alone.

This dissertation addresses that gap through the Design Science Research Methodology. It proposes a process-oriented framework for governed Human-AI collaboration treating AI systems as first-class collaborators whose contributions must be bounded, validated and traced. The framework defines lifecycle phases with phase-specific quality gates, responsibilities expressed as functions rather than job titles, and governance metrics, anchored by a persistent versioned specification linking every generated artefact to its originating requirement. It is instantiated as Apex, a web application operationalising each phase, gate and artefact against a real project management backend. Evaluation combines application in a real project, standardised usability and workload instruments, interviews with practitioners and an Agile expert, and an analytical assessment against criteria set in advance. The contribution is an evidence-grounded process model allowing organisations to adopt AI assistance without surrendering oversight of software quality, security and accountability.

Keywords

Human-AI Collaboration; Software Development Life Cycle; Large Language Models; Process Framework; AI Governance; Design Science Research

Resumo

A Inteligência Artificial Generativa e os Modelos de Linguagem de Grande Dimensão passaram de assistentes de preenchimento automático a contribuidores activos no Ciclo de Vida de Desenvolvimento de Software, permitindo exprimir requisitos em linguagem natural e supervisionar os artefactos gerados. A adopção ultrapassou o método. Uma Revisão Sistemática da Literatura sobre 646 artigos confirma benefícios substanciais a par de riscos recorrentes de fiabilidade, segurança, rastreabilidade e responsabilização, e identifica a ausência de orientação estruturada e repetível para integrar estas ferramentas em contextos profissionais como o principal problema por resolver. Concluindo então que o valor depende do processo, não apenas da capacidade do modelo em questão.

Esta dissertação aborda essa lacuna através da Design Science Research Methodology, propondo uma framework orientada a processo para a colaboração Humano-IA governada, cujas contribuições de IA devem ser delimitadas, validadas e rastreadas. Define fases com portões de qualidade específicos, responsabilidades expressas como funções e não como cargos, e métricas de governação, ancoradas numa especificação versionada que liga cada artefacto ao seu requisito. É instanciada no Apex, aplicação web que operacionaliza cada fase, portão e artefacto sobre uma ferramenta real de gestão de projectos. A avaliação combina aplicação num projecto real, instrumentos padronizados de usabilidade e carga cognitiva, entrevistas, e avaliação analítica face a critérios previamente definidos. O contributo é um modelo de processo fundamentado em evidência que permite adoptar assistência por IA sem abdicar do controlo sobre qualidade, segurança e responsabilização.

Palavras Chave

Colaboração Humano-IA; Ciclo de Vida de Desenvolvimento de Software; Modelos de Linguagem de Grande Dimensão; Framework de Processo; Governação de IA; Design Science Research

Contents

1 Introduction **1**

2 Research Methodology **3**
2.1 Systematic Literature Review (SLR) . . . 3
2.2 Design Science Research Methodology (DSRM) . . . 4
2.3 Mapping the Methodology onto this Document . . . 5

3 Research Background **6**
3.1 Artificial Intelligence . . . 6
3.2 Software Development Life Cycle . . . 7
3.3 AI in Software Development . . . 7
3.4 Vibe Coding . . . 7
3.5 Spec-Driven Development . . . 7

4 Systematic Literature Review **8**
4.1 Planning the Review . . . 8
4.1.1 Motivation . . . 8
4.1.2 Research Questions . . . 9
4.1.3 Review Protocol . . . 9
4.2 Conducting the Review . . . 10
4.2.1 Filtering Process . . . 10
4.2.2 Data Extraction Analysis . . . 11
4.3 Reporting the Review . . . 11
4.3.1 Benefits of AI and LLMs usage in Software Development . . . 12
4.3.2 Consequences of AI and LLMs usage in Software Development . . . 13
4.3.3 Methodologies and management methods used to integrate AI and LLMs in Software Development . . . 14
4.3.4 Developers' and other stakeholders' opinions regarding the use of AI and LLMs in Software Development . . . 15
4.3.5 Industry sectors adopting AI and LLMs for Software Applications Development . . . 15
4.4 Discussion . . . 16

5 Research Problem **17**

6 Research Proposal **19**
6.1 Objectives . . . 20
6.2 Purpose and Scope . . . 20
6.3 Design Principles . . . 21
6.4 Framework Overview . . . 22
6.5 Roles and Responsibilities . . . 23
6.6 Lifecycle Phases and the AI Interaction Mapping . . . 24
6.7 Governance Mechanisms and Quality Gates . . . 26

6.7.1 Bolts as the Unit of Execution . . . 26
6.7.2 Spec-Anchored Continuity . . . 27
6.7.3 Quality Gates . . . 28
6.7.4 Governance Metrics . . . 28
6.8 Operational Playbooks . . . 29
6.9 Continuous Flow in Place of Sprint Cadence . . . 30
6.10 Positioning Against Alternatives . . . 31
6.11 Summary . . . 31

7 Implementation **32**
7.1 Purpose and Positioning . . . 32
7.2 Architecture . . . 33
7.3 Operationalising the Framework . . . 35
7.4 The Governance Analytics Surface . . . 39
7.5 Design History and Practitioner Feedback . . . 40
7.6 Quality Assurance of the Implementation . . . 41
7.7 Limitations of the Implementation . . . 41

8 Demonstration **42**
8.1 Demonstration Design . . . 42
8.2 Context . . . 45
8.3 Application of the Framework . . . 46
8.4 Observations . . . 50
8.5 Summary . . . 53

9 Evaluation **54**
9.1 Evaluation Strategy . . . 54
9.2 Participants . . . 55
9.3 Instruments and Procedure . . . 56
9.3.1 System Usability Scale . . . 57
9.3.2 NASA Task Load Index . . . 57
9.3.3 Apex UX Questionnaire . . . 58
9.3.4 Semi-Structured Interviews . . . 59
9.3.5 Analytical Assessment . . . 60
9.4 Results . . . 61
9.4.1 System Usability Scale . . . 61
9.4.2 Raw NASA Task Load Index . . . 61
9.4.3 Apex UX Questionnaire . . . 62
9.4.4 Analytical Assessment . . . 63
9.4.5 Demonstration in a Real Project . . . 64
9.5 Threats to Validity . . . 65
9.6 Discussion . . . 66
9.6.1 Addressing the Research Problem . . . 67

10 Conclusion **69**
10.1 Summary and Contributions . . . 69
10.2 Communication . . . 71
10.3 Limitations . . . 71
10.4 Future Work . . . 72
Declaration of Generative AI and AI-Assisted Technologies in the Writing Process . . . 73

Bibliography **73**

A Extended Results for the SLR **79**
A.1 Stakeholder opinions reported in the literature 79
A.2 Industry sectors adopting AI and LLMs 81
A.3 Final set of studies retained by the review 82

B Evaluation Instruments **83**
B.1 Instrument Access 83
B.2 Informed Consent and Demographics 84
B.3 Raw NASA Task Load Index 84
B.4 System Usability Scale 85
B.5 Apex UX Questionnaire 85
B.6 Interview Guides 87
B.7 Per-Participant Raw Scores 88

List of Figures

2.1 Three phases of the Systematic Literature Review 4
2.2 Phases of the Design Science Research Methodology 4

4.1 Articles Filtering Process 10
4.2 Academic Journal Publications 11

6.1 The six lifecycle phases and the three loop-backs out of Maintenance 25

7.1 Apex's interface, Phase 1 Requirements 33
7.2 Apex architecture, part one 34
7.3 Apex architecture, part two 34
7.4 A representative walk through Apex, Phases 1 to 3 37
7.5 A representative walk through Apex, Phases 4 to 6 38

8.1 Outfolio's dashboard 45
8.2 A real scenario failure caught at the Testing Gate 48
8.3 Outfolio's first production deployment 49
8.4 Maintenance feedback routed to Discovery 50

B.1 Informed consent form 84

List of Tables

2.1 Mapping between DSRM activities and the chapters of this dissertation 5

4.1 Research Questions and Focus Areas 9
4.2 Research Elements and Details 9
4.3 Inclusion and Exclusion Criteria 10
4.4 Benefits of using AI and LLMs in Software Development 12
4.5 Consequences of using AI and LLMs in Software Development 13
4.6 Methodologies and management methods for integrating AI and LLMs 14
4.7 Impact of AI and LLMs across Software Development Lifecycle Phases 16

6.1 Research questions, key findings, problem aspects and framework objectives 20
6.2 The seven design principles and the evidence grounding each 21
6.3 The hats: the function each discharges and the gate each owns 23
6.4 The six lifecycle phases and the preceding Strategic Alignment stage 25
6.5 The five quality gates: owner, required evidence, and the consequence of failure 28
6.6 The six operational playbooks: trigger, procedure, and termination 29

7.1 Framework constructs against the mechanisms implementing them in Apex 36

9.1 Evaluation instruments, target artefact, and problem dimension addressed 55
9.2 Raw NASA-TLX subscale means by task 61
9.3 Apex UX questionnaire, selected items 62
9.4 Analytical assessment of the framework against criteria fixed in advance 64

A.1 Public / Researchers Opinions on AI and LLMs in Software Development 79
A.2 Developers / Practitioners Opinions on AI and LLMs in Software Development 80
A.3 Organizations / Management Opinions on AI and LLMs in Software Development 80
A.4 Industry sectors adopting AI and LLMs 81
A.5 Studies retained by the systematic literature review 82

B.1 Demographics items collected on the consent form 84
B.2 Raw NASA Task Load Index subscales, in the order presented 85
B.3 System Usability Scale items, as administered 85
B.4 Apex UX questionnaire items, by construct 86
B.5 Apex UX questionnaire responses 87
B.6 Semi-structured interview guides 87
B.7 SUS score and subscales, per participant 88

Acronyms

AI	Artificial Intelligence
AI-DLC	AI-Driven Development Lifecycle
API	Application Programming Interface
BDD	Behaviour-Driven Development
CI	Continuous Integration
DSRM	Design Science Research Methodology
GenAI	Generative Artificial Intelligence
IaC	Infrastructure as Code
KPI	Key Performance Indicator
LLM	Large Language Model
MLOps	Machine Learning Operations
NASA-TLX	NASA Task Load Index
NIST	National Institute of Standards and Technology
OKR	Objective and Key Result
PM	Product Manager
PO	Product Owner
QA	Quality Assurance
SDLC	Software Development Life Cycle
SLR	Systematic Literature Review
SSRF	Server-Side Request Forgery
SUS	System Usability Scale
UI	User Interface
UX	User Experience

1

Introduction

Generative Artificial Intelligence (GenAI) and Large Language Models (LLMs) have extended machine assistance towards generating coherent, context-aware artefacts, introducing intent-driven programming, colloquially *Vibe Coding*, in which developers describe desired behaviour in natural language and refine what a model produces [1, 2]. The shift is not confined to code: LLMs now participate in requirements elicitation, architectural exploration, test generation, documentation and debugging, touching every Software Development Life Cycle (SDLC) phase [3–5]. The practitioner's work moves from constructing software towards specifying intent, supervising generated artefacts and remaining accountable for them.

The evidence on outcomes, however, is more nuanced than early enthusiasm suggested, and that gap motivates this work. Adoption is close to universal while trust is not: 84% of developers report using or planning to use AI tools, up from 76%, while the proportion trusting their accuracy fell from 40% to 29% [6], with daily use around 90% [7]. That distrust is warranted, and the decisive point is that functional correctness and security do not move together: 61% of a leading coding agent's solutions were functionally correct but only 10.5% secure [8], and across more than one hundred models 45% of samples introduced an OWASP Top 10 vulnerability, with newer models improving at correctness while security stayed flat [9].

Productivity gains are equally conditional. Early evidence reported a task completed 55.8% faster with an assistant, though unpublished in a peer-reviewed venue [10], whereas a randomised controlled trial found 16 experienced developers on their own mature repositories took 19% longer while estimating afterwards that they had been 20% faster [11]; the favourable result came from a self-contained task with no codebase to respect. At organisational scale, telemetry across 1,255 teams associates high adoption with 98% more pull requests merged but 91% longer review times and no significant correlation with delivery outcomes [12]. The bottleneck moves downstream to validation rather than disappearing. Several

sources here are industry-reported rather than peer-reviewed and are identified as such throughout, so the argument rests on consistency of direction rather than any single figure: the value of AI assistance is conditional on the process surrounding it rather than a property of the model alone.

To characterise practice rigorously, an Systematic Literature Review (SLR) following Kitchenham's guidelines retrieved 646 articles, reported in Chapter 4. Its central finding is methodological: organisations lack established methodologies for the responsible use of Artificial Intelligence (AI) in professional software development [13–16], and proposed approaches remain fragmented across isolated tools, sharing no common foundation on lifecycle integration, oversight or evaluation [3]. Established methodologies do not close the gap: Waterfall, Scrum, Kanban and the Agile Manifesto were designed for environments in which every contributor is human [4], so they lack structures for governing AI participation, bounding the context a model may act upon, and allocating responsibility over generated artefacts, while their cadence assumes implementation is the binding constraint, which the evidence contradicts. **The research problem is therefore the absence of structured, repeatable guidance and standardised evaluation practices for integrating AI across the SDLC while preserving human oversight, software quality and security, and accountability.**

The objective is to design, instantiate, demonstrate and evaluate a process-oriented framework that addresses each of the four dimensions of that problem formalised in Chapter 5: *placement*, fixing for every SDLC phase where AI participates, for what purpose and where its participation ends; *governance*, providing 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. Chapter 6 addresses the first two by design, Chapters 7 and 8 the third, including the constructs weakened, proxied or left unimplemented, and Chapter 9 assesses all four.

The contributions are four: the synthesis above; a process framework treating AI as a first-class collaborator, with phases, gates, function-based responsibilities, metrics and a versioned specification anchoring artefacts to requirements; Apex, a reference implementation showing those constructs implementable rather than describable; and an empirical evaluation of its usability and workload.

2

Research Methodology

Contents

2.1 Systematic Literature Review (SLR) 3
2.2 Design Science Research Methodology (DSRM) 4
2.3 Mapping the Methodology onto this Document 5

This chapter presents the two research methodologies used in this dissertation, the SLR and the Design Science Research Methodology (DSRM), and maps the activities of the second onto the chapters of this document.

2.1 Systematic Literature Review (SLR)

The review follows Kitchenham's guidelines, which describe a systematic review as "a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest" conducted through "a trustworthy, rigorous, and auditable methodology." The guidelines prescribe three phases: **planning**, which defines the motivation, research questions, selection criteria and protocol; **conducting**, which applies the protocol to identify and select studies; and **reporting**, which synthesises the findings.

Figure 2.1 illustrates those phases and their activities; the filtering that separates them is conducted in Chapter 4, where the successive passes are given together with their attrition numbers.

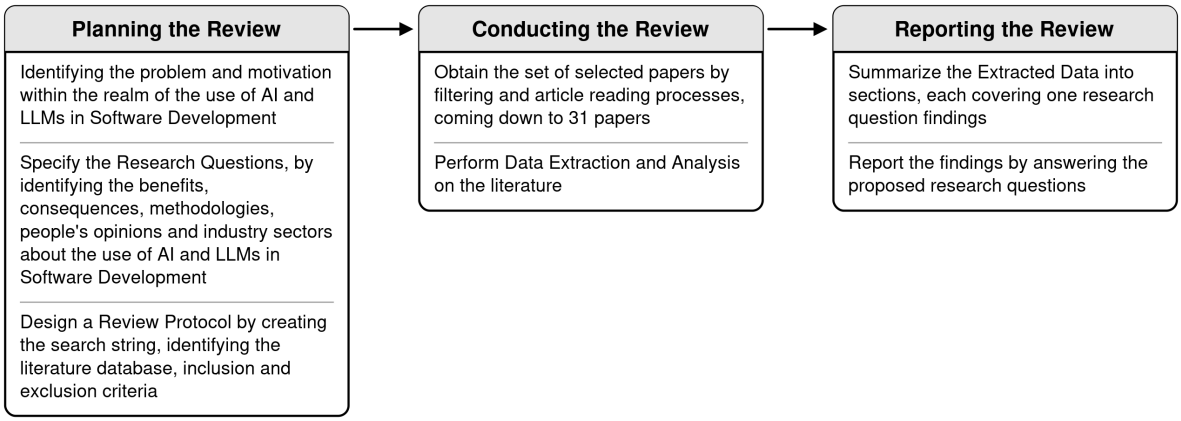


Figure 2.1: Three phases of the Systematic Literature Review

2.2 Design Science Research Methodology (DSRM)

The DSRM concerns the development of IT artefacts, such as models, methods or frameworks, addressing identified organisational or technological problems. Unlike purely observational research it emphasises constructing solutions and evaluating their usefulness. Proposed by Peffers et al., it follows a six-step process supporting systematic design, clear contributions and transferable knowledge, which makes it suitable for research proposing process-oriented solutions. The steps are illustrated in Figure 2.2 and described below.

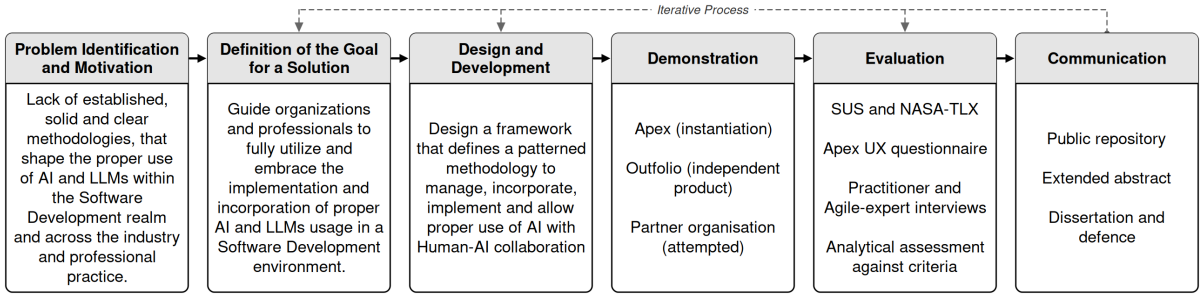


Figure 2.2: Phases of the Design Science Research Methodology

Problem identification and motivation identifies, through the SLR, the lack of structured guidance for integrating AI across the SDLC while preserving quality, accountability and human oversight. **Definition of objectives** converts this gap, along its four dimensions, into objectives for secure and responsible adoption. **Design and development** produces the framework and Apex as its reference implementation. **Demonstration** applies the artefact through Apex and Outfolio, with a third practitioner setting attempted but not completed. **Evaluation** assesses the framework through interviews and analytical criteria, and Apex through usability, workload and User Experience (UX) measures, keeping framework soundness distinct from tool usability. **Communication** disseminates the work through the Apex repository, an extended abstract, and the dissertation and defence.

2.3 Mapping the Methodology onto this Document

Because DSRM prescribes activities rather than document sections, Table 2.1 states where each is carried out. The mapping is not strictly linear: as Chapter 7 describes, feedback gathered while building the reference implementation fed back into the framework's design, which is the iteration path DSRM explicitly allows.

Table 2.1: Mapping between DSRM activities and the chapters of this dissertation

DSRM Activity	Chapter	Output
Problem Identification and Motivation	Chapters 4 and 5	Evidence base from the SLR; formalised research problem
Definition of Objectives	Chapter 6	Objectives derived from the review findings
Design and Development	Chapters 6 and 7	Framework and Apex reference implementation
Demonstration	Chapter 8	Apex and Outfolio applications, with a third setting attempted
Evaluation	Chapter 9	Usability and workload measures, UX questionnaire, interviews, and analytical assessment
Communication	Chapter 10	Public repository, extended abstract, dissertation, and defence

Realisation, the third dimension of the problem, spans both Design and Development and Demonstration because it is established in two halves: which constructs are realisable at all is shown in Chapter 7's construct-by-construct mapping, and whether that realisation survives real use is confirmed in Chapter 8.

3

Research Background

Contents

3.1 Artificial Intelligence 6
3.2 Software Development Life Cycle 7
3.3 AI in Software Development 7
3.4 Vibe Coding 7
3.5 Spec-Driven Development 7

This chapter presents the theoretical background relevant to the research: Artificial Intelligence, the Software Development Life Cycle, AI in Software Development, Vibe Coding, and Spec-Driven Development.

3.1 Artificial Intelligence

Artificial Intelligence refers here to systems performing tasks associated with human cognition, while Generative AI denotes systems that produce new content such as text, images, audio or code. Large Language Models (LLMs) are a class of Generative AI trained on large text and source-code corpora, typically using transformer architectures.

In software engineering, LLMs can interpret natural-language intent and generate, explain, refactor, document and test code across multiple programming languages. These capabilities make them useful as collaborative tools throughout software development, supporting both novice and experienced developers while reducing manual effort in repetitive or knowledge-intensive tasks [1, 3–5, 13, 17–25].

3.2 Software Development Life Cycle

The Software Development Life Cycle (SDLC) structures software development into requirements, design, implementation, testing, deployment and maintenance phases [3]. It is central to this research because AI participates differently across these stages. Although Agile and DevOps have increased iteration and automation, many activities remain manual and time-consuming, including debugging, documentation, repository consistency and routine implementation. As systems become more complex, pressure to improve productivity without compromising maintainability or security has driven interest in AI-assisted workflows [3, 4, 22, 24].

3.3 AI in Software Development

LLMs are increasingly used to accelerate prototyping, explain unfamiliar code, support documentation and assist with implementation. Their role is generally collaborative rather than fully autonomous: the model generates, summarises or proposes, while developers remain responsible for architectural decisions, validation, integration and alignment with requirements.

Research therefore increasingly focuses on how workflows adapt to this division of labour, how trust in AI-generated output is formed, and how responsibility is distributed between human judgement and automated generation. These issues are examined in detail in Chapter 4 [3, 4, 13, 14, 20, 22, 24, 26].

3.4 Vibe Coding

Vibe Coding describes a development style in which developers express intent, requirements or constraints in natural language and a generative model produces corresponding code or related artefacts. Development proceeds conversationally, with the developer evaluating and refining generated output rather than beginning from manually written code. This shifts part of the programming effort from direct implementation towards specification and review, affecting developer workflow, cognitive effort and responsibility across the SDLC [1, 2].

3.5 Spec-Driven Development

Spec-driven development is an emerging practice in which a persistent, versioned specification becomes the primary reference for both developers and AI agents. Rather than generating code directly from loosely defined prompts, the model works against explicit requirements, design decisions and constraints that are maintained alongside the codebase. Industry examples include AWS Kiro, GitHub Spec-Kit and the Tessl Framework, all of which place stronger emphasis on persistent specification artefacts [27]. Thoughtworks identifies the approach as an emerging AI-assisted engineering practice, while Farrag describes it as a form of specification-driven governance, although that work remains a preprint [27, 28].

The relevance to this dissertation is that an explicit specification gives AI-generated output a stable reference against which it can be constrained and validated. This principle directly motivates the Spec-Anchored Continuity mechanism introduced in Chapter 6.

4

Systematic Literature Review

Contents

4.1 Planning the Review 8
4.2 Conducting the Review 10
4.3 Reporting the Review 11
4.4 Discussion 16

This chapter describes the execution of the systematic literature review, following the three SLR stages: Planning, Conducting, and Reporting the results.

4.1 Planning the Review

4.1.1 Motivation

AI, particularly LLMs and GenAI, is increasingly adopted across the SDLC to support code generation and completion, documentation, requirements engineering, automated bug fixing, and developer learning, and its use has been associated with productivity gains, improved efficiency, and support for complex development tasks [3, 4, 13, 14, 20, 22, 24, 26].

Adoption has nevertheless outpaced the resolution of the challenges it raises. The literature reports persistent concerns over the quality, reliability and trustworthiness of AI-generated artifacts, methodological barriers such as limited explainability and the black-box nature of advanced models, a heavy dependence on prompt formulation, and a concentration on isolated tasks that leaves several SDLC phases, particularly requirements engineering and design, with limited practical guidance.

4.1.2 Research Questions

Five research questions structure the review:

Table 4.1: Research Questions and Focus Areas

RQ	Research Question	Focus
RQ1	What are the benefits of using and incorporating Vibe Coding, AI, and Large Language Models in software development?	Identify and categorize improvements in productivity, creativity, code quality, automation, collaboration, and innovation enabled by AI-driven tools.
RQ2	What are the consequences that may arise from using and incorporating Vibe Coding, AI, and LLMs in software development, both in code generation and other sections?	Analyze risks such as technical debt, bias, reduced developer learning, code ownership, ethical issues, and dependency on AI tools.
RQ3	What methodologies and management methods are being used to integrate Vibe Coding, AI, and LLMs into software engineering processes and workflows?	Explore frameworks, best practices, and process models (e.g., AI-augmented Agile, human-in-the-loop development, prompt engineering practices).
RQ4	What are developers' and other stakeholders' opinions regarding the use of AI and LLMs in software development?	Assess perceptions, trust, satisfaction, and acceptance of AI-assisted coding among developers and organizations.
RQ5	Which industry sectors are adopting AI for application development, and how do they benefit from its integration?	Map adoption trends across industries (e.g., finance, healthcare, IT, education), highlighting sector-specific motivations and outcomes.

4.1.3 Review Protocol

The criteria used for the literature search, stated in Table 4.2, returned 646 articles.

Table 4.2: Research Elements and Details

Element	Research Details
Source	EBSCO
Final Search String	(“Vibe Cod*” OR “AI models” OR “AI agen*” OR “large language model*” OR “generative AI” OR openai OR “chatgpt” OR LLM* OR “claude” OR “gemini” OR copilot OR co-pilot) AND (“software development” OR “application development” OR “code generation”)
Search Method	Search string found in the abstract of articles in English academic journals or conference materials, with no date range limit
Results	646

"Vibe Coding" is the most recent term for the practice, but the practice predates the term, and querying it alone returned few relevant articles because the same work is published under different expres-

sions. Synonymous keywords were added accordingly: it is the combination of "code generation" and "application development" with "generative AI" and "AI models" that retrieves the intended body of work, not any single term. The string was executed against the EBSCO Online Digital Library through its Advanced Search option, over the AB abstract field and the title, restricted to full-text-available results.

4.2 Conducting the Review

The second stage applies the filtering criteria to the 646 articles retrieved in the first; the attrition at each step is shown in Figure 4.1.

4.2.1 Filtering Process

Extracting all 646 articles was not possible: EBSCO's page allows the download of 50 articles at a time, and after reloading and extending the page to its maximum the site displayed, in relevance order, only 467 of the 646.

The AI-powered systematic review management platform "Rayyan" was used to organise the articles. Of the 467 extracted, 117 duplicates were identified and removed, leaving 350 for abstract reading; that pass rejected 286 as belonging to a theme other than developing applications using AI, leaving 64 articles.

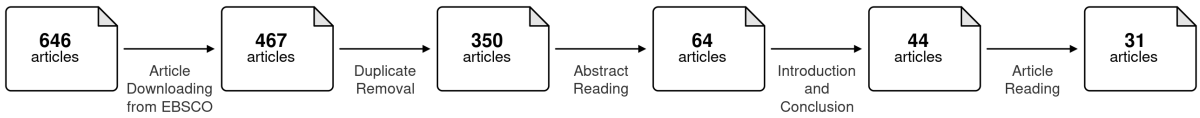


Figure 4.1: Articles Filtering Process

The content of those 64 articles was then read and each placed in one of four categories: Included, Maybe, Excluded and Skipped. Articles were "Skipped" when the introduction and conclusion showed them to be off-theme, outdated, or silent on all five research questions; 20 were skipped and excluded on those grounds. "Maybe" was used while it was unclear whether an article bore on the theme, and most were ultimately excluded in favour of more recent studies.

An article that answered any of the five research questions clearly and explicitly was included. Where the information was unclear, insufficient, or too thin to confirm that an answer was given, the predefined criteria in Table 4.3 decided eligibility. Of the 44 articles, 31 met the criteria and were retained for analysis.

Table 4.3: Inclusion and Exclusion Criteria

Type	Criterion
Inclusion	Relevance regarding AI and LLM usage in the development of software applications.
Inclusion	Relevance regarding benefits and consequences of incorporating and using AI and LLMs in the development of software applications.

Continued on next page

Type	Criterion
Inclusion	Relevance regarding methodologies and management methods used to integrate AI and LLMs into software engineering processes and workflows.
Inclusion	Relevance regarding people's opinions with AI and LLM use in software development.
Inclusion	Relevance regarding which industry sectors are adopting AI for application development and its benefits.
Exclusion	Not written in English.
Exclusion	Does not clarify any of the Inclusion Criteria.
Exclusion	Outdated due to the highly dynamic nature of the field (anything older than 2024).

4.2.2 Data Extraction Analysis

None is categorised as conference material: every one is a peer-reviewed academic journal article as classified by EBSCO, distributed over time as shown in Figure 4.2.

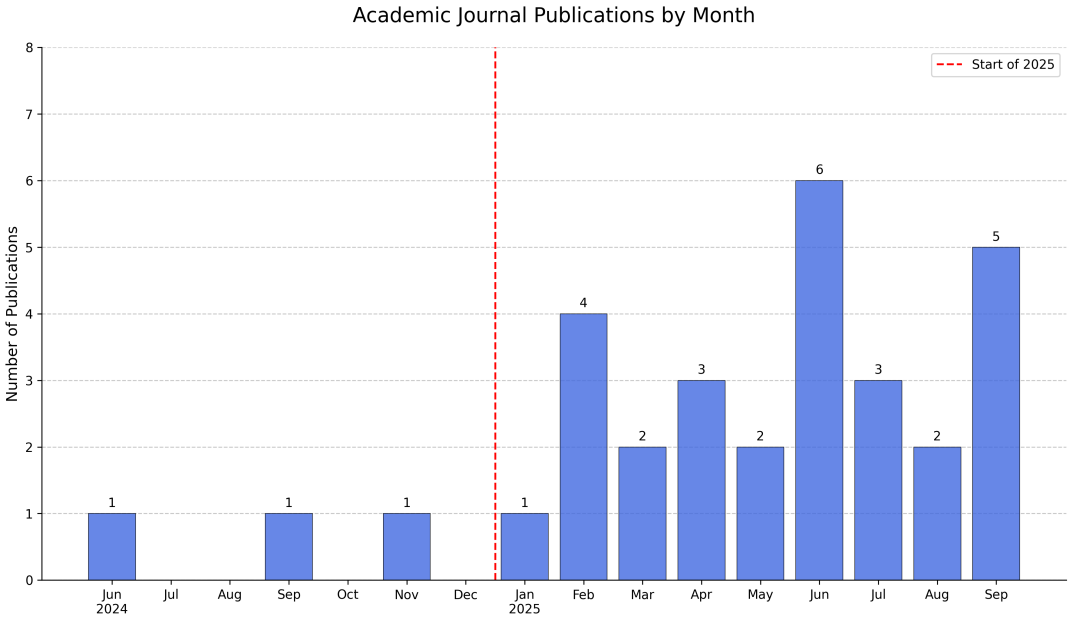


Figure 4.2: Academic Journal Publications

4.3 Reporting the Review

The final stage reports the retained studies by research question. The venue distribution is given in Section A.3: ACM Transactions on Software Engineering and Methodology contributes ten studies, Information and Software Technology three, and all remaining venues one each, indicating a concentrated but cross-field evidence base.

4.3.1 Benefits of AI and LLMs usage in Software Development

The first research question examines the benefits of AI and LLM adoption for developers, organisations and other stakeholders. The reported benefits are summarised in Table 4.4.

Table 4.4: Benefits of using AI and LLMs in Software Development

Benefits	Description	#	References
Improved Code Quality, Accuracy, and Traceability	"improving code quality and optimizing work with faster times" [14].	20	[3, 4, 13–16, 18, 20–23, 29–37]
Increased Productivity and Efficiency	"converts natural language or abstract intent into functioning software modules, dramatically shortening development cycles" [1].	18	[1–4, 13, 14, 18, 19, 21, 23–25, 29, 32–34, 36, 38]
Automation of Testing, Debugging, and Repair	AI and LLMs support "automating coding, bug detection, debugging, and personalizing software according to user needs" [14].	12	[1, 3, 16, 17, 19, 23–25, 29, 31, 33, 36]
Optimization of Code Performance and Efficiency	"The proportionally small number of studies in SE subfields like Build, Release, Deploy, Operate and Monitor indicates a broad untapped area for cost savings and novelty" [26].	10	[1, 2, 4, 13, 14, 17, 21, 24, 26, 33]
Support for Early Software Development Phases (Requirements & Design)	AI and LLMs have effectiveness in "...enhancing stakeholder communication and requirement documentation" while noting challenges with bias and accuracy in requirements and design elicitation [23].	8	[4, 15, 16, 20, 23, 24, 32, 33]
Automated Documentation and Summarization	"LLMs can [...] automatically generate documentation for codebases [...] ensuring that documentation stays current and can be updated as code evolves" [21].	8	[4, 13, 17, 20–22, 24, 34]
Enhanced Security and Vulnerability Management	AI supports SDLC security through "vulnerability detection, threat modeling, secure coding practices, and incident response" [17].	6	[17, 22, 24, 33, 35, 38]
Knowledge Acquisition and Understanding	"Reduce technical barriers and transform the programming process into a collaborative, conversational, and more accessible act" [2].	5	[2, 13, 19, 32, 33]
Management and Organizational Support	Assist in managing and "curating organizational information about day-to-day organizational practices" [32].	5	[3,4,16,17,20]
Developer Well-being and Satisfaction	"prioritize interesting and fun tasks through automating menial, boring tasks" [32].	4	[4, 18, 32, 33]
Democratization and Accessibility	"shift the programming paradigm from a traditional code-centric approach to one that involves verbal interactions between the programmer and the artificial intelligence agent" [2].	2	[1, 2]

The most frequently reported benefits are improved code quality and productivity. The literature generally treats them as complementary, AI can accelerate development while supporting accuracy, traceability and adherence to quality standards [29]. Automation of testing, debugging and repair reinforces this effect, particularly where LLMs use compiler or execution feedback to iteratively correct generated code. Benefits also extend to documentation, knowledge acquisition, developer satisfaction

and accessibility to programming.

4.3.2 Consequences of AI and LLMs usage in Software Development

The second research question concerns the technical, operational and human consequences of incorporating AI and LLMs into software development. These are summarised in Table 4.5.

Table 4.5: Consequences of using AI and LLMs in Software Development

Consequences	Description	#	References
Increased Operational Costs and Resource Constraints	"a substantial financial burden [...] when LLMs are deployed to handle a large number of tasks" [5].	20	[3, 5, 19–22, 24, 26, 37, 39]
Reliability, Accuracy, and Hallucination Risks	Particularly problematic in complex or niche scenarios: "this AI can simply make things up or sometimes spit out things that don't even exist" [13].	15	[2, 4, 15, 16, 18–20, 24, 25, 29, 30, 34–37]
Security Vulnerabilities in Generated Code	"AI-generated code may harbor unique security risks in areas such as cryptographic implementation, memory management, and error handling" [38]. Existing tools do not close the gap: they detect only "46.7% of vulnerabilities in AI-generated code, a significantly lower rate than for human-written code vulnerabilities (66.4%)" [38].	15	[3, 13, 14, 16–19, 21, 22, 25, 26, 33–35, 38]
Ethical, Legal, and Intellectual Property Issues	"questions about the ownership and licensing of that code will arise" when models are trained on publicly available, open-source projects [21].	8	[2, 3, 13, 17, 18, 20, 21, 24]
Homogeneity and Bias Propagation	LLMs increasingly train on LLM-generated code, tending to "reaffirm patterns seen in their training data (e.g., public GitHub repositories)" [24].	8	[3, 13, 14, 17, 18, 21, 24, 36]
Evaluation and Standardization Challenges	"It is urgent to establish an effective, objective, and complete evaluation system for large language models for code" [3].	7	[3, 14, 15, 22, 37, 38, 40]
Risk of Over-reliance and Cognitive Decline	"if AI is constantly available, people will tend to become lazy and stop thinking for themselves" [13].	4	[2, 13, 18, 32]
New Overhead and Workflow Inefficiencies (Human Review)	LLMs still "require human oversight and review of their outputs before implementation" [23].	4	[2, 13, 20, 23]

The most frequently reported concerns are operational cost, reliability and security. Training, fine-tuning and deploying large models can offset productivity gains through substantial resource costs, while hallucinated or otherwise plausible-but-incorrect output remains a persistent reliability risk. Security is similarly unresolved, with lower reported detection rates for vulnerabilities in AI-generated code than in human-written code [38].

Adoption also introduces human and governance concerns, including over-reliance and skill erosion, propagation of training-data bias, and unresolved ownership and licensing questions [21, 24]. The reported productivity benefits therefore introduce corresponding requirements for review, governance and evaluation.

4.3.3 Methodologies and management methods used to integrate AI and LLMs in Software Development

The third research question examines the methods used to integrate AI and LLMs into software-development workflows. The approaches identified in the corpus are listed in Table 4.6.

Table 4.6: Methodologies and Management Methods used to integrate AI and LLMs in Software Development

Methodologies	Description	#	References
In-Context Learning (ICL) / Few-shot Learning	"integrating context-specific information, in the form of an input prompt or instruction template, during the inference and thus without the need to perform gradient-based training" [39].	9	[15, 16, 19, 20, 22, 24, 33, 36, 39]
Prompt Engineering (PE) / Prompt-Driven Development	"The effectiveness of prompt engineering critically determines the performance of leveraging the capabilities of LLMs". [22]	5	[3, 19, 22, 33, 40]
Chain-of-Thought (CoT) Prompting	"A CoT is several intermediate natural language reasoning steps that lead to the final answer" [30].	5	[3, 16, 19, 22, 30]
Human-In-The-Loop (HITL) Paradigm	"The importance of the human contribution in identifying and resolving problems to maintain software quality is widely acknowledged by many authors" [23]. LLMs still "require human oversight and review of their outputs before implementation" [23, 24].	6	[2, 15, 21, 23–25]
Structured Prompting / Structured Chain-of-Thought (SCoT)	"SCoT denotes several intermediate reasoning steps constructed by programming structures" [30].	4	[22, 29, 30, 37]
Multi-Agent Systems / Self-Collaboration Approach	"a self-collaboration framework with role instruction, which allows LLM agents to collaborate with each other to generate code for complex requirements" [25, 31].	4	[13, 25, 31, 35]
Integration with External Tools and Domain Knowledge	"Combining the linguistic capability of LLMs with specialized domain knowledge to enhance output quality, functionality, or security" [13, 25].	3	[3, 4, 13, 25]
Other Single-Study Techniques	Five techniques appeared once each: Retrieval-Augmented Generation as a lower-computation alternative to parameter updates, though weaker than LoRA [19, 39]; AgileGen and Gherkin-based testing for structured human decision-making and clearer acceptance criteria [4]; Vibe Coding for translating natural-language intent into software [1]; and LLM-assisted testing, reporting 1.4×-2.3× improvements over baselines [25].	5	[1, 4, 19, 25, 39]

The methods fall broadly into two levels. At prompt level, In-Context Learning is the most frequently discussed technique (9 articles), followed by Prompt Engineering, Chain-of-Thought and Structured Chain-of-Thought approaches that improve output through contextual or structured reasoning [30].

At process level, Human-In-The-Loop development is the dominant principle, because model output remains variable, developers retain responsibility for review and acceptance before implementation [23]. AgileGen formalises this pattern by concentrating human decisions at scenario definition and final acceptance [4]. Multi-agent, retrieval-based and tool-augmented techniques are less frequently reported. “Vibe Coding” appears only once in the corpus and therefore remains weakly represented as a studied

methodology [1].

4.3.4 Developers' and other stakeholders' opinions regarding the use of AI and LLMs in Software Development

The fourth research question addresses stakeholder perceptions, which differ sufficiently to distinguish three groups in Section A.1.

- **Public and researchers.** The dominant concern is fallibility rather than capability. Across six articles, misinformation and inconsistent output are central concerns, particularly where generated code appears valid while failing the intended requirement.
- **Developers and practitioners.** Productivity and efficiency gains are the most frequently cited positive views, also appearing in six articles. LLMs are valued for knowledge acquisition and reduced dependence on extended search or colleagues [13], but over-reliance, skill erosion and prompt-design effort remain important countervailing concerns.
- **Organisations and management.** Adoption is primarily framed as risk management. Automating routine work may support retention of skilled staff [32], but data security and privacy are the most frequent concerns, appearing in six articles. Organisation-specific or fine-tuned models are sometimes preferred despite higher operational cost.

Across all three groups, the strongest shared position is the need for human oversight, identified in seven articles.

4.3.5 Industry sectors adopting AI and LLMs for Software Applications Development

The fifth research question examines where AI is being adopted for application development and the benefits reported by sector; the extracted evidence is summarised in Section A.2.

Adoption is concentrated in Software Engineering, Software Development and IT Consulting, where common uses include code generation, completion, explanation and refactoring. Several studies describe a shift towards workflows in which developers direct, validate and refine generated output rather than produce all code manually [4, 23, 32]. Consulting additionally benefits from faster onboarding and knowledge transfer, while infrastructure-heavy and mathematically intensive domains often require model adaptation or structured domain knowledge [3, 5, 17, 25].

Beyond core IT, adoption appears in finance, banking and regulated industries (8 articles), transportation and other safety-critical domains (7 articles), healthcare and biomedical research (5 articles), education and training (5 articles), and legal reasoning (4 articles). Education mainly uses LLMs for exercise generation, personalised feedback and formative assessment rather than educator replacement [2, 15, 36].

Across regulated and safety-critical sectors, adoption is consistently associated with stronger requirements for reliability, auditability, compliance and human review [1, 3, 22, 33]. No sector in the reviewed evidence reports adoption without a human check, and higher-risk sectors impose the most explicit controls.

4.4 Discussion

Read across the five research questions, the review converges on a single pattern. Every benefit reported under RQ1 has a corresponding risk under RQ2 threatening the same property it claims: speed and code quality against hallucination, security vulnerabilities and homogenisation; democratised access against over-reliance and skill erosion. The field's answer to that tension is not automation but control, every prevalent method under RQ3, those being In-Context Learning, Prompt Engineering, Chain-of-Thought and above all Human-In-The-Loop exist to shape or supervise AI output before it is trusted, and none proposes removing the human; every stakeholder group under RQ4 pairs enthusiasm with a caveat, human oversight being the one consensus surviving their differing incentives; and under RQ5 it is the organising variable for adoption itself, broad and unconditional where an incorrect output is cheap, explicitly conditioned on verification where it is expensive, as in finance, transportation or healthcare. The absence of a standardised, secure integration methodology, not a shortfall in model capability, is therefore what the literature identifies as the actual barrier to broader adoption, the gap Chapter 5 formalises. Table 4.7 restates the pattern phase by phase; read down its challenge column and the same requirement recurs at every one, a human check of the model's output against something written before that output existed.

Table 4.7: Impact of AI and LLMs across Software Development Lifecycle Phases

SDLC Phase	Main Challenges
Requirements Analysis	Risk of ambiguity amplification, overconfidence in AI-generated requirements, and lack of domain context; increased need for human validation early in the process.
Design	Limited understanding of system-wide constraints; risk of inconsistent or suboptimal architectural decisions without expert oversight.
Implementation	Hallucinated code, inconsistent outputs, security vulnerabilities, and reduced trust due to output variability; heavy reliance on human review.
Testing	Difficulty validating correctness of AI-generated tests; increased testing complexity due to unpredictable AI-generated code.
Deployment	Risk of misconfiguration or insecure defaults; limited contextual awareness of operational environments.
Maintenance	Potential over-reliance on AI explanations; challenges in maintaining long-term system understanding and accountability.

5

Research Problem

The review reported in Chapter 4 shows that AI and LLMs are being adopted across sectors and throughout the SDLC. While these tools offer substantial benefits, their effective use is hindered by the lack of established and consistent methodologies for integrating them into professional software development [13–16].

Several AI-oriented development frameworks have been proposed, as summarised in Table 4.6, but they share no common foundation for lifecycle integration, management practice or evaluation [3]. Consequently, organisations lack standardised benchmarks and repeatable processes for assessing the effects of AI-assisted development on software quality, security and maintainability.

Established methodologies such as Waterfall, Scrum, Kanban and the Agile Manifesto were designed around human contributors and do not fully address AI-driven development [4]. AI-assisted coding can accelerate implementation while shifting effort towards requirements clarification, review, testing and maintenance. Existing approaches also provide limited guidance for governing AI participation, assessing model reliability, managing risk and assigning responsibility for AI-generated output.

Evidence discussed in Chapter 1 further suggests that faster or functionally correct code generation does not automatically improve security or delivery outcomes [9, 12]. This indicates that the central challenge is not simply AI capability, but the process by which AI is integrated, controlled and evaluated within software development. **The core research problem addressed in this work is therefore the lack of structured methodological guidance and standardised evaluation criteria for integrating and managing AI and LLM-based tools across the SDLC, while maintaining human oversight, software quality, security and organisational accountability. This gap limits organisations' ability to adopt AI-assisted software development systematically and sustainably.**

The problem has four dimensions, each traceable to the review and representing a condition that any solution must satisfy:

- **Placement.** There is no shared account of where in the SDLC AI participates, for what purpose, and where its participation ends. As a result, the point at which human responsibility must take over is left to individual judgement rather than fixed by the process (Table 4.6).
- **Governance.** Mechanisms for keeping AI contributions traceable to requirements, validated before progression, and attributable to an accountable person, particularly quality gates and human-in-the-loop practices, are recognised in principle but specified only ad hoc, if at all (Table 4.5).
- **Realisation.** Guidance that remains purely descriptive cannot demonstrate which constructs survive implementation and which fail when enforced in a working system.
- **Practitioner support.** AI shifts effort from code production towards review, with review times increasing as output grows [12] and over-reliance recurring as a risk (Table 4.5). Guidance that is methodologically sound but too burdensome to apply therefore does not resolve the problem.

The first two dimensions define what structured guidance must contain; the latter two concern whether that guidance can be implemented and sustained in practice.

6

Research Proposal

Contents

6.1 Objectives 20
6.2 Purpose and Scope 20
6.3 Design Principles 21
6.4 Framework Overview 22
6.5 Roles and Responsibilities 23
6.6 Lifecycle Phases and the AI Interaction Mapping 24
6.7 Governance Mechanisms and Quality Gates 26
6.8 Operational Playbooks 29
6.9 Continuous Flow in Place of Sprint Cadence 30
6.10 Positioning Against Alternatives 31
6.11 Summary 31

This chapter presents the second and third activities of the DSRM: the definition of objectives for a solution to the research problem formalised in Chapter 5, and the design of the artefact that satisfies them, a process-oriented framework for governed Human-AI collaboration across the SDLC. The framework is described here as a methodology, independently of any implementation; its instantiation as a working system is the subject of Chapter 7. This design-and-development activity addresses the first two dimensions of that problem: Section 6.6 fixes where AI participates in each SDLC phase and where its participation ends, and Sections 6.7.3 and 6.7.4 define the gates, metrics and human-in-the-loop practices that keep AI contributions traceable, validated and accountable. Section 6.4 introduces

the vocabulary the rest of the chapter depends on before the detail follows: who performs the work (Section 6.5), when (Section 6.6), how it is governed (Section 6.7), how it is performed day to day (Section 6.8), and why sprints are replaced (Section 6.9).

6.1 Objectives

The systematic literature review in Chapter 4 led to the problem formalised in Chapter 5: the lack of structured guidance, methodological rigour and standardised evaluation practice for integrating these tools into development processes. The research objectives address that problem directly, and Table 6.1 links each identified gap to the framework objective that resolves it.

Table 6.1: Mapping between Research Questions, Key Findings, Problem Aspects, and Framework Objectives

RQ	Key Finding from SLR	Problem Aspect Addressed	Resulting Framework Objective
RQ1	Usage is unevenly distributed across SDLC phases and lacks structured integration.	No structured guidance on where and how AI should be applied across the SDLC.	Define a process framework mapping AI-supported activities to specific phases.
RQ2	Recurring risks in reliability, validation, security and insufficient human oversight.	Insufficient methodological support for risk, quality and accountability.	Integrate governance mechanisms, human-in-the-loop practice and quality gates.
RQ3	Significant variation among approaches and no consensus on managing adoption.	Fragmentation of existing AI adoption methodologies.	Consolidate practice into a unified, repeatable process-oriented framework.

Table 6.1 derives objectives for placement and governance, the two dimensions a design can satisfy on paper. The other two, realisation and practitioner support, can only be established once the design exists, and are taken up in Chapters 7 to 9.

6.2 Purpose and Scope

The artefact is a method in the sense the DSRM gives that term: a set of lifecycle phases, the responsibilities attached to each, the artefacts each must produce, and the rules for passing from one to the next; the vocabulary is introduced in Section 6.4. It is stated at the level of process rather than of tooling, so that it can be assessed independently of the system instantiating it in Chapter 7 and adopted against a different tool stack. It covers AI-assisted activity in every phase rather than the code-authoring phases alone, the collaboration practices that activity requires, expressed as process obligations rather than a catalogue of prompts, and the governance mechanisms of Section 6.7.

It excludes the training or fine-tuning of LLMs, model architecture design, Machine Learning Operations (MLOps) pipelines and tool-specific detail. These are boundaries rather than omissions, following from Chapter 5's concern with teams that *consume* general-purpose models rather than *produce* them.

MLOps governs a model's lifecycle as a deployed asset, while this framework governs the lifecycle of software a model helped write, and a pipeline can be mature under the first while the second remains ungoverned, which is the deficiency Chapter 4 reports. Tool-specific detail is excluded for the same reason Chapters 7 and 9 are kept separate, since a framework prescribing a vendor cannot be evaluated independently of it.

6.3 Design Principles

Seven principles govern every subsequent design decision. Each is stated in Table 6.2 together with the evidence motivating it, since a principle asserted without grounding is a preference, and a framework built out of preferences cannot be defended as a research contribution. Where the grounding is a design commitment rather than an empirical finding, that is said so.

Table 6.2: The seven design principles and the evidence grounding each

Principle	Statement and grounding
Human-in-the-Loop by Design	All AI-generated artefacts require explicit human review and acceptance before integration because adoption and trust diverge, at around 90% daily use against roughly 30% reporting low or no trust [7], and the best agent and model combination was functionally correct 61% of the time but secure only 10.5%, with 45% of samples introducing an OWASP Top 10 vulnerability [8, 9].
AI as Augmentation, Not Substitution	AI supports rather than replaces human judgement, preserving responsibility for technical, architectural and quality decisions. In a randomised controlled trial, developers took 19% longer with assistance while estimating they had been 20% faster [11], and the SLR reports reduced code understanding as a recurring consequence of unsupervised delegation [2, 13, 18, 32].
Process-Level Governance of AI Usage	AI participation is governed through repeatable processes that ensure visibility, traceability and accountability, since AI amplifies, making capable teams measurably more capable and dysfunctional teams measurably less so [7]; governance must therefore live in the process rather than in prompt discipline alone.
SDLC-Wide and Phase-Aware Integration	AI integration extends across the entire software development lifecycle, with responsibilities and controls adapted to the risks and requirements of each phase, addressing the current concentration of AI usage in code generation [3–5].
Functional, Not Positional, Accountability	Roles are defined as functions rather than fixed organisational positions, ensuring that each AI-assisted activity has identifiable responsibility for initiation, validation and acceptance, irrespective of team composition [3, 13].
Risk-Proportional Validation and Control	Validation and control mechanisms are calibrated to the technical, organisational and ethical risks of each activity. The defect profile is not uniform, with vulnerability rates reaching 72% for Java against 45% overall and clustering in specific weakness categories [8, 9].
Spec-Anchored Continuity	A persistent, versioned specification provides the authoritative reference for AI-assisted activities and their validation, establishing continuity between development phases while recognising the emerging nature and unresolved scalability challenges of spec-driven development [27, 28].

Each principle recurs later as a concrete mechanism, so this table also serves as a map of the rest of the chapter: Human-in-the-Loop by Design as the Interactive Bridge and the quality gates (Sections 6.4

and 6.7.3); Augmentation, Not Substitution as the Consistency Factor and the Developer hat as an active editor (Section 6.7.1); Process-Level Governance as the gates and metrics together (Sections 6.7.3 and 6.7.4); SDLC-Wide Integration as the phase mapping itself (Section 6.6); Functional Accountability as the hats (Section 6.5); Risk-Proportional Validation as the Fix-Bolt's Fast and Secure Lanes (Section 6.7.1); and Spec-Anchored Continuity as the persistent specification mechanism of the same name (Section 6.7.2).

6.4 Framework Overview

The framework's most direct precedent is the AI-Driven Development Lifecycle (AI-DLC) published by Amazon Web Services [41], which organises work into Inception, Construction and Operations. Three of its terms are adopted here unchanged and recur throughout this chapter: a **Bolt**, a short execution cycle measured in hours or days that replaces the sprint; a **Unit of Work**, the top-level grouping of business justification and acceptance criteria, optionally organised into Epics rather than replacing them as AI-DLC does; and **Mob Elaboration**, the collaborative ritual in which the Trio critiques AI-drafted requirements before they are locked. No originality is claimed for any of the three, and AI-DLC's own reported ten-to-fifteen-fold productivity improvement is vendor-reported, unreplicated, and is not carried here as evidence; only its process vocabulary is adopted.

The team that receives a Unit of Work is the **Squad**, comprising the Trio and the Bolt Master, Developer, Quality Assurance (QA) and Deploying hats; the Product Manager (PM) hat acts before it. Every responsibility within it is a **hat**: a function the framework requires rather than a position that must be staffed by a dedicated person, so one individual may hold several and a small team may still leave none unfilled. Three hats, Product Owner (PO), Tech Lead and Design Lead, jointly form the **Trio**, which owns Discovery and Design together; the full set of hats and the gate each owns is detailed in Section 6.5.

Work is organised in a four-level hierarchy. A **Unit of Work** decomposes into **User Stories**, sized extra-small or small, each the minimum unit of observable business value and the unit the QA hat validates; a story decomposes into **Tasks**, granular technical steps queued in the **Bolt Backlog**; and a Bolt is the execution of one task. There are no sprints, since tasks are pulled continuously rather than committed to in batches; the argument is made in full in Section 6.9. Unlike AI-DLC, which replaces the Epic outright, the framework leaves it optional: a team may still group a Unit of Work's stories into Epics, a project-management-tool-native sub-grouping the framework neither requires nor prohibits, and Epic-level organisation is independently supported by evidence of higher correctness and completeness than requirement-level decomposition [42]. The demonstration in Chapter 8 uses Epics throughout for exactly this reason.

Discovery produces **Gherkin** acceptance criteria, structured Given/When/Then scenarios the Trio approves before anything is built. Approved scenarios, together with the technical specification agreed at the Design Gate, are **anchored**: written into a persistent, versioned specification split into a project-wide layer and a per-story layer, detailed in Section 6.7.2. Every phase that follows works against that anchor rather than against the whole project.

Each phase is bounded by a **quality gate**: a named owning hat, a specific body of required evidence, and a defined consequence when that evidence is absent. Five gates, detailed in Section 6.7.3, decide whether a phase may be exited, and three governance metrics, detailed in Section 6.7.4, assess whether

the gates are working.

6.5 Roles and Responsibilities

Every role in the framework is a hat: a required function rather than a position that must be staffed by a dedicated person. One individual may therefore hold several hats, and on small teams responsibilities may be absorbed by adjacent functions. Table 6.3 summarises the hats and the gates they own. Expressing responsibility functionally rather than organisationally serves three purposes: it supports solo and small-team use, preserves accountability when formal titles are absent, and avoids treating structural models as directly transferable blueprints.

The Trio (Section 6.4) draws inspiration from the Spotify Model, but not as a prescriptive template. Spotify's own account was descriptive rather than normative [43], later commentary explicitly framed it as inspiration rather than a blueprint [44], and first-hand retrospective evidence identified unresolved matrix authority and autonomy-without-accountability problems [45]. The framework addresses these through the Tech Lead hat's architectural lock (Section 6.7.1) and by separating Bolt logistics from technical authority, with each gate owned by a named function. Team Topologies provides a closer structural analogue through its emphasis on cognitive load [46], while the PM hat resembles Amazon's single-threaded leadership model [47].

Deployment approval and infrastructure-security review remain mandatory but are owned by the Deploying hat, held by whoever performs the deployment, whether self-reviewed in solo work or peer-reviewed in a larger team. This avoids introducing governance bodies that the process assumes exist but that may never be staffed.

The gates themselves do not weaken as team size decreases. In solo work they are self-checked, but the required evidence in Table 6.5 must still exist, be recorded and be reviewed explicitly at the gate. Their value lies in enforcing a distinct validation step against a specification that predates the artefact, preserving meaningful review even when author and reviewer are the same person.

Table 6.3: The hats: the function each discharges and the gate each owns

Hat	Function the framework requires	Gate owned
PM	Stakeholder alignment, Objective and Key Results (OKRs) and Key Performance Indicators (KPIs), and the business justification of a Unit of Work before it reaches the Squad; receives classified business deviations from Maintenance.	Strategic Alignment sign-off, pre-lifecycle
PO (Trio)	Business value and prioritisation of User Stories; leads the critique of AI-drafted scenarios during Mob Elaboration.	Discovery, with the Trio
Tech Lead (Trio)	Technical specification, architectural feasibility and integration risk; decomposes stories into tasks and flags high-risk ones before any Bolt begins.	Design (technical) and the pre-Bolt architectural lock
Design Lead (Trio)	Prototype or visual element for every feature, and its validation through usability testing.	Design (usability)

Continued on next page

Hat	Function the framework requires	Gate owned
Bolt Master	Continuous flow: prioritises the Bolt Backlog, manages dependencies, assigns by capacity once the Tech Lead hat has approved.	None; holds logistics authority deliberately separated from technical authority
Developer	Executes tasks with the model, generates and validates the Consistency Factor, and remains answerable for correctness, security and being able to explain the logic; also reviews Fix-Bolt security and diagnoses Maintenance signals.	Implementation
QA	Validates the assembled story against the approved Gherkin in a separate environment, verifying generated Behaviour-Driven Development (BDD) scripts before execution and probing edge cases manually.	Testing
Deploying	Held by whoever performs a release, who approves it and reviews the security of any generated Infrastructure as Code (IaC), configuration or migration.	Deployment

6.6 Lifecycle Phases and the AI Interaction Mapping

This section addresses the placement dimension of the problem. The framework organises the lifecycle into six phases, preceded by a Strategic Alignment stage that sits outside the lifecycle proper because it is performed before the Squad is engaged at all. It decides whether a Unit of Work should exist, whereas the six phases decide how one is built. The six are Discovery and Requirements; Design, Prototyping and Architecture; Implementation; Testing; Deployment; and Maintenance. Table 6.4 fixes, for the stage and each of them, what enters, what the model is permitted to produce, what leaves, which hat is accountable, and which control must be satisfied before exit. It is the central artefact of this chapter and is not restated in prose.

The phases are listed in an order, but that order is not the shape of the process, since Maintenance neither terminates the lifecycle nor patches in place. Its exit is always an entry into a named earlier phase, whose gate then applies in full: a business deviation routes to the PM hat for a value decision and, if approved, re-opens Discovery and Requirements; a specification gap re-opens Design, Prototyping and Architecture with the technical specification amended and re-versioned; a production defect re-enters Implementation as a Fix-Bolt and returns through Testing. This is not Waterfall with a maintenance tail. In Waterfall, feedback is measured against a specification treated as frozen, whereas here the specification is exactly what is re-opened, re-approved and re-versioned, and code is regenerated from it. What the framework refuses is the unnamed loop, a change applied to production code with no corresponding change to the specification it is supposed to satisfy, since that is how traceability is lost in practice. Figure 6.1 shows the six phases and these three loop-backs; Table 6.4 is the detail behind each box.

A more detailed, BPMN-style diagram for each individual phase is published within Apex itself and its public repository (Section 10.2), for readers who want the finer-grained process view this single figure necessarily compresses.

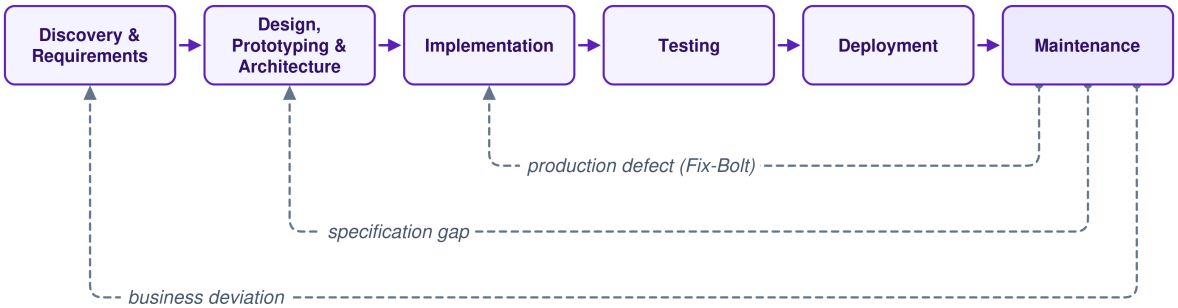


Figure 6.1: The six lifecycle phases in sequence, and the three named loop-backs out of Maintenance. Each loop-back re-enters a specific earlier phase, whose gate then applies in full - Maintenance never patches production code without a corresponding change to the specification.

Table 6.4: The six lifecycle phases and the preceding Strategic Alignment stage: permitted AI contribution, output, accountable hat, and the control that must be satisfied before each is exited. Each phase takes as input the output of the one above it.

Phase	Role of AI	Output	Human accountability	Mandatory control
Strategic Alignment (pre-lifecycle)	Assists market analysis; drafts candidate objectives and indicators	A Unit of Work carrying a stated business justification, OKRs and KPIs	PM hat aligns the product vision with stakeholders and owns the justification	Strategic Alignment sign-off before the Unit of Work reaches the Squad
Discovery & Requirements	Drafts fractional User Stories in natural language; compiles approved scenarios into formal Gherkin	Approved User Stories sized XS or S, with Gherkin acceptance criteria and stable scenario identifiers	The Trio validates correctness, completeness and business value through the Interactive Bridge	Mob Elaboration and the Discovery Gate
Design, Prototyping & Architecture	Proposes architectural options; generates User Interface (UI) structure and a formal technical specification	Approved prototype and an anchored technical specification (interface contract, data model, runtime contract)	Design Lead hat validates usability; Tech Lead hat validates feasibility, scalability and integration risk	Design Gate; no task enters the Bolt Backlog until both validations pass
Implementation	Generates and refactors code constrained by the Consistency Factor test set (Section 6.7.1)	Task-level code passing the pipeline and the Consistency Factor	Developer hat validates output, owns correctness and security, and must be able to explain the logic	Pre-Bolt architectural lock, the Bolt micro-cycle, and the Implementation Gate at story level
Testing	Drafts end-to-end BDD scripts derived from the approved Gherkin	A story-level verdict and, on failure, the exact failing evidence	QA hat verifies and refines generated scripts before execution and probes edge cases manually	The QA Validation Playbook and the Testing Gate, in a separate environment decoupled from the Bolt
Deployment	Answers the infrastructure delta question; generates configuration, IaC or migrations only when required	A released increment and a persistent context updated to the live state	Deploying hat approves the release and reviews its security implications	Deployment Gate, performed by the Deploying hat; no separate gatekeeping team is assumed

Continued on next page

Phase	Role of AI	Output	Human accountability	Mandatory control
Maintenance	Reconstructs a deliberately narrow context to explain one specific failure and proposes a bounded fix	A classified signal routed to the phase that owns it, with a verified diagnosis where applicable	Developer hat classifies the signal and verifies any diagnosis; business deviations pass to the PM hat	Diagnostic firewall (whole-project context prohibited) and governed loop-back to a named phase

The Discovery and Requirements row names the mechanism behind that validation, the **Interactive Bridge**: the model drafts scenarios in natural language, and the Trio confirms, adds, deletes or modifies them before approved scenarios are compiled into Gherkin acceptance criteria (Section 6.4) and anchored into the specification (Section 6.7.2). Separating critique from formalisation this way lets domain experts catch an incorrect assumption without first having to read it through formal syntax. The same row also names **Mob Elaboration**, the Trio's structured session critiquing AI-drafted stories as a group before any are approved; the procedure is given in full as one of the six playbooks in Section 6.8. The table's other rows name mechanisms the same way, the gates, the architectural lock, the QA Validation Playbook among them, each unpacked in turn in Sections 6.7 and 6.8 rather than here.

Where the phase model departs from AI-DLC. Most of the expansion from three phases to six is decomposition rather than disagreement, since Discovery and Requirements together with Design, Prototyping and Architecture occupy the territory AI-DLC calls Inception, and Implementation corresponds to Construction. Two substantive extensions distinguish the proposed framework from it:

- **Decoupled Testing phase.** In AI-DLC, build and test conclude Construction within the same cycle, creating a risk of circular validation, since tests generated by the same model from the same specification may reproduce the assumptions embedded in the implementation. Passing such tests demonstrates internal consistency rather than independent correctness, as evidenced by the gap between functional and security test performance [8, 9]. The proposed framework therefore establishes Testing as an independent phase, with its own environment, responsible role and quality gate. Validation proceeds from the locked functional specification rather than the implementation, shifting its unit from individual Bolts to the assembled User Story, where business value can be assessed.
- **Distinct Maintenance phase with a diagnostic firewall.** AI-DLC's Operations phase addresses deployment and monitoring without defining how post-deployment signals are classified or assigned. The proposed framework distinguishes defects against agreed specifications, previously unspecified requirements and incorrect specifications, preventing undocumented fixes from introducing specification drift. Each signal is classified before code is accessed, after which the Developer hat is restricted to the reported issue, relevant test evidence and implicated code fragment rather than the whole project. This diagnostic firewall limits unnecessary project-wide context and preserves specification integrity [8], as further discussed in Section 6.7.2.

6.7 Governance Mechanisms and Quality Gates

6.7.1 Bolts as the Unit of Execution

A Bolt is the smallest execution iteration the framework recognises: the implementation of one task within a User Story, strictly inside the development environment and measured in hours rather than

weeks. The term is adopted from AI-DLC [41]; what the framework adds is the sequence of obligations around it, since a short cycle without a fixed reference is a fast way to accumulate divergence.

Before any code is generated, the Tech Lead hat feeds the locked context (the approved Gherkin, the prototype and the technical specification) to the model, verifies the task list it proposes, and flags high-risk tasks. This **architectural lock** makes the technical boundary a matter of record rather than negotiation, letting tasks on either side of an interface proceed in parallel. The Bolt Master hat then assigns by capacity, giving Developer hats the story's intent, not authority over the technical scope the lock has settled. A Developer hat pulls one task, at which moment the Bolt begins and Bolt Cycle Time is measured from. The Developer hat and model then produce the **Consistency Factor**, a strict automated test set derived from both specifications that constrains generation rather than reporting on it afterwards; the model generates against that constraint and the Developer hat refines the result as an active editor, not an approver of a finished proposal. A task is done when the code passes the pipeline *and* the Developer hat can explain the logic, since code no one can explain has been substituted for engineering rather than added to it. When every task is done, the story passes the Implementation Gate into staging.

The **Fix-Bolt** is the single exception: triggered by a failed Testing Gate or a verified defect, it omits task breakdown because the context already contains what that step would produce, the exact failing scenario and its evidence. It is justified by what is already known rather than being a licence to skip planning for a change that merely feels small. Its outcome is routed by risk, where the risk-proportional principle becomes operational: a low-risk patch takes the **Fast Lane** directly to the delivery pipeline without re-entering formal QA, while a higher-risk one takes the **Secure Lane** back to staging for a Regression Bypass, in which the QA hat re-runs the existing suite before verifying the targeted patch.

6.7.2 Spec-Anchored Continuity

Spec-Anchored Continuity requires every AI-generated artefact to carry forward the context established by previously validated phases. The mechanism is a persistent versioned specification maintained alongside the codebase and split into two layers with different lifetimes. The **global layer** holds what does not change between stories: architectural rules, technology-stack decisions, security policies and recorded workarounds for already-diagnosed defects. It is written rarely, amended deliberately, and applies to every generation on the project. The **local layer**, the Feature Specs, holds what is specific to the work in hand, the approved Gherkin acceptance criteria and the technical specification produced at the Design Gate; it is written once per story, locked at a gate, and amended only through a recorded versioned amendment. The split is what makes the mechanism usable, since a single undifferentiated context grows monotonically until it contains more irrelevant material than relevant, reintroducing the problem it was meant to solve.

The central argument is that narrowing the model's working context is a control rather than a convenience. A convenience makes a correct outcome easier to reach, whereas a control is a measure whose absence makes an incorrect outcome undetectable. Anchoring satisfies the second definition on three counts. It supplies a reference that existed before the generated artefact did, the precondition for validation being possible at all. It bounds what the model may act upon, an unbounded context being the condition under which a model produces a wide, confident and substantively wrong change, which is why whole-project context is prohibited during Maintenance diagnosis [8]. And because both layers are

versioned and every post-lock edit is a dated amendment, divergence between what was agreed and what was built becomes a detectable event rather than an unattributable drift. This is the spec-driven-development instantiation named in Table 6.2; a recent preprint independently argues, from a multivocal review, that specification discipline rather than model capability is the binding constraint on dependability [28]. The contribution is not the idea of anchoring but the specification of where it is mandatory, which gate locks each layer, and what constitutes a legitimate amendment.

6.7.3 Quality Gates

A quality gate has three named components: an owning hat, a specific body of evidence that must exist and be examined, and a defined consequence when it is absent or inadequate. A checkpoint missing any of the three is a ceremony rather than a control, since it can be passed without anything being established. Table 6.5 specifies the five gates on those terms. Notably, failure never discards work and never silently degrades it, but returns it to a named earlier activity with the evidence attached, the same routing discipline Maintenance applies to post-deployment signals.

Table 6.5: The five quality gates: owner, required evidence, and the consequence of failure

Gate	Owning hat	Evidence required to pass	Consequence of failure
Discovery	The Trio	Testable Gherkin acceptance criteria approved through the Interactive Bridge; every story sized XS or S; the functional specification locked and versioned.	Scenarios return to Mob Elaboration; oversized stories are decomposed before re-submission.
Design	Design Lead and Tech Lead hats	An approved prototype validated by usability testing; a technical specification covering interface contract, data model and runtime contract, anchored into the persistent context.	Feedback returns to prototype generation or architectural specification; no task may enter the Bolt Backlog.
Implementation	Developer hat	Every task complete, passing the pipeline and the Consistency Factor, with the Developer hat able to explain the generated logic.	The story does not enter staging; incomplete or unexplained tasks return to the Bolt Backlog.
Testing	QA hat	The assembled story validated in staging against the locked Gherkin through verified BDD scripts, with edge cases probed manually.	A Fix-Bolt is triggered with the exact failing evidence attached and routed by risk through the Fast or Secure Lane (Section 6.7.1).
Deployment	Deploying hat (whoever performs the deployment)	Explicit approval of the release and a security review of any generated IaC, configuration or migration; the persistent context updated to the live state.	The specific objection is fed back for a compliant regeneration; the release does not proceed.

6.7.4 Governance Metrics

Three governance metrics assess the safety and flow of AI-assisted development rather than output volume, which does not necessarily translate into improved delivery [12]:

- **Bolt Cycle Time.** The elapsed time from a task being pulled as a Bolt to its completion. Measured at task level to isolate execution time from story-level queueing, review and gate transitions, thereby assessing continuous flow.
- **Context Traceability Rate.** The proportion of deployed Bolts and stories whose artefacts can be traced through the locked technical and functional specifications to their originating Unit of Work. It measures whether Spec-Anchored Continuity is maintained in practice.
- **AI Defect Escape Rate.** The proportion of production defects originating in AI-generated logic that passed the Testing Gate. It assesses gate effectiveness by identifying defects that escaped validation despite formal gate completion.

These are expressed in vocabulary continuous with established AI governance standards rather than bespoke to this work. The National Institute of Standards and Technology (NIST) AI Risk Management Framework's Govern, Map, Measure and Manage [48] correspond respectively to this chapter's design principles, phase mapping, metrics and gates, and ISO/IEC 42001:2023 offers a complementary plan-do-check-act vocabulary for the same loop [49]. This is a claim about shared language only. Neither the framework nor Apex has been audited against either standard, and nothing here asserts compliance or intent to certify, only that these constructs occupy the conceptual space standards bodies are converging on.

6.8 Operational Playbooks

The phases, gates and hats state what must hold; the six playbooks in Table 6.6 state how the recurring procedures are performed, each with its trigger, its participants and its termination condition. A team that follows these six and observes the gates in Table 6.5 is operating the framework, irrespective of the tooling it uses. In Maintenance, the prohibition on whole-project context is the single most restrictive rule in the framework, for the reason given in Section 6.7.2. In the Fix-Bolt, the Regression Bypass re-runs the existing BDD suite rather than regenerating it, because a regenerated suite is a different measurement instrument, and comparing a patched system against it cannot distinguish a fixed defect from a changed test.

Table 6.6: The six operational playbooks: trigger, procedure, and termination

Playbook	Trigger	Procedure	Terminates when
Mob Elaboration	A Unit of Work brought to the Squad	The Trio defines intent, personas and constraints, then reviews agent-drafted stories through iterative refinement. Approved scenarios are compiled into Gherkin and anchored.	The Discovery Gate passes and the functional specification is locked
Design and Prototyping	Locked Gherkin acceptance criteria	The Design Lead and Tech Lead hats define interfaces, data models and runtime contracts. Usability testing and architectural review validate the resulting artefacts.	Both validations pass and tasks enter the Bolt Backlog

Continued on next page

Playbook	Trigger	Procedure	Terminates when
QA Validation	A story that has passed the Implementation Gate	The QA hat independently derives and manually verifies end-to-end BDD scripts from the locked functional specification before execution, followed by exploratory testing.	The Testing Gate passes, or a Fix-Bolt is triggered with the failing evidence
Fix-Bolt	A failed Testing Gate or a verified defect	Immediate corrective work addresses the isolated defect against the anchored specification. The Developer hat reviews security implications and verifies existing and new unit tests.	Low risk takes the Fast Lane to the pipeline; higher risk takes the Secure Lane through a Regression Bypass in staging
Deployment and Release	A story that has passed the Testing Gate	The agent assesses infrastructure, configuration and migration requirements, generating artefacts only when necessary. The Deploying hat reviews them for security and policy compliance.	The Deployment Gate passes and the persistent context is updated to the live state
Maintenance and Evolution	A post-deployment signal	Signals are classified before code access. Business deviations return to the PM hat and specification gaps to the phase that owns them; technical defects undergo restricted-context diagnosis and manual verification.	The signal is classified and routed, with a verified diagnosis where applicable

6.9 Continuous Flow in Place of Sprint Cadence

The framework replaces sprints and sprint backlogs with continuous flow, the most disruptive single element of the proposal and therefore the one requiring the strongest empirical justification; the Epic, unlike the sprint, is not replaced and remains an optional grouping (Section 6.4). Sprint cadence manages a world in which implementation is the binding constraint. A fixed-length iteration lets a quantity of implementation work be forecast, committed to and inspected at a boundary, calibrated to how long a useful amount of implementation takes. The evidence in Chapter 1 indicates that premise no longer holds where AI assistance is heavy. Developers in a randomised controlled trial were measurably slower with assistance while believing they had been faster [11], and telemetry across 1,255 teams pairs the merged-pull-request increase reported there with pull requests 154% larger, 91% longer review times, and no significant correlation with delivery outcomes [12]. The constraint moved rather than disappeared, so a cadence calibrated to the old one now obscures the new one. The structural response has two parts which are the execution compressed into Bolts, so the queue in front of validation refills in small increments rather than batches, and validation decoupled into its own phase, environment, hat and gate, so the constraint is scheduled and measured in its own right. Both are falsifiable: if review is not the constraint in a given setting, Bolt Cycle Time and the QA queue will say so. The work hierarchy that takes the sprint backlog's place, Unit of Work, User Story, Task and Bolt, is set out in Section 6.4.

The project management tool should provide states corresponding unambiguously to the framework's phases, without requiring a specific board configuration.

6.10 Positioning Against Alternatives

Four alternatives are considered below. Each has legitimate advantages, but the evidence motivates a different design choice.

- **No framework at all.** This minimises process overhead and may be appropriate for short-lived prototypes with little coordination or maintenance. However, functionally correct agent output can remain insecure at scale, with security not improving alongside correctness [8, 9]. Validation therefore cannot be treated as optional process overhead.
- **Adopt AI-DLC unmodified.** AI-DLC offers a coherent, commercially deployed three-phase model with less process complexity [41]. The divergence here is limited to gaps identified in Section 6.6, since build and test are folded into Construction, while Operations provides no published mechanism for classifying post-deployment feedback. The proposed framework extends rather than replaces its useful structure.
- **Retain Scrum or SAFe cadence and treat AI as a faster development environment.** Existing ceremonies are familiar, well tooled and inexpensive to retain, but Section 6.9 shows the premise they rest on, that implementation is the binding constraint, no longer holds under heavy AI assistance. A cadence calibrated to implementation therefore risks measuring and managing the wrong activity.
- **Metrics-only governance, with no phase structure.** This is cheaper, flexible and consistent with evidence that no single metric captures software delivery performance [50]. However, metrics indicate whether governance is working; they do not provide governance themselves. A dashboard can reveal rising review time or escaping defects without defining what an AI system may act upon, where validation must occur, or how post-deployment feedback is routed.

6.11 Summary

The placement and governance dimensions of the problem are addressed by the artefact this chapter has specified: the six-phase model of Section 6.6 with the Strategic Alignment stage preceding it and the mapping in Table 6.4, and Section 6.7's governance set of Bolts, the Consistency Factor, Spec-Anchored Continuity, five gates and three metrics. Two properties of that design are worth isolating, because a phase list and a gate list alone would not achieve them: the model is constrained at every phase to that phase's anchored specification rather than the whole project, and accountability attaches to functions rather than positions, so it survives a team of one while every failed gate routes work back to a named phase rather than discarding it.

Both are design claims and not yet evidence. The framework's constructs are an original synthesis argued from the literature but not demonstrated in use, and spec-driven development, on which Spec-Anchored Continuity rests, is characterised by its own principal source as emerging with unsettled scaling behaviour [27]. Chapter 7 presents Apex, the reference implementation, and reports which constructs were implementable as specified, which were weakened or proxied, and which resisted implementation, which is the realisation dimension of the problem.

7

Implementation

Contents

7.1 Purpose and Positioning 32
7.2 Architecture 33
7.3 Operationalising the Framework 35
7.4 The Governance Analytics Surface 39
7.5 Design History and Practitioner Feedback 40
7.6 Quality Assurance of the Implementation 41
7.7 Limitations of the Implementation 41

The framework in Chapter 6 defines the methodology, while this chapter presents Apex as its working instantiation. Apex demonstrates which governance constructs can be implemented in practice and provides the system used in Chapters 8 and 9. In DSRM terms, the method and its instantiation are distinct artefacts from the same Design and Development activity. This chapter therefore addresses the realisation dimension of Chapter 5, including both successfully implemented constructs and those that resisted implementation.

7.1 Purpose and Positioning

Apex is not a coding agent and is not positioned against one. A coding agent accelerates the writing of code; Apex governs whether the code that results still answers the question that prompted it. The two are complementary, and the distinction is the whole reason the system exists because acceleration

without anchoring is the condition Chapter 1 shows moves the bottleneck to review rather than removing it. Every artefact Apex produces carries a stable specification identifier back to the requirement it derives from, so divergence between what was agreed and what was built becomes a detectable event rather than a drift nobody is positioned to notice.

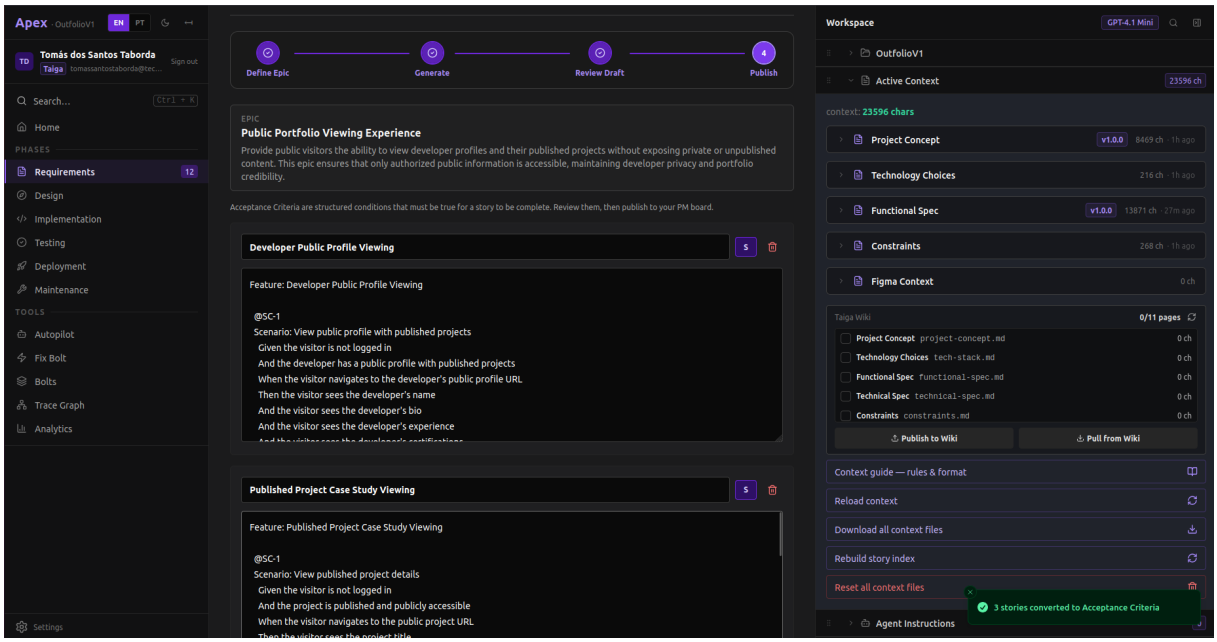


Figure 7.1: Apex Phase 1 Requirements interface, showing phase navigation.

Apex also has explicit boundaries. It does not train or fine-tune models, provision infrastructure, or replace an existing project-management tool. Models are treated as external services; deployment artefacts such as configuration or migration scripts are generated for human or Continuous Integration (CI) execution rather than applied directly. Apex instead operates above the team's existing project-management system, which remains the system of record, requiring only that its workflow states map onto the framework phases described in Section 6.9.

7.2 Architecture

Apex is a split full-stack application. The frontend uses Next.js 15 with TypeScript, while the backend uses Python 3.12 with FastAPI, connected through a typed REST API. React Query manages server state and Zustand client state. Backend routers remain thin, with workflow logic isolated in a service layer and schemas defining request and response boundaries. Figures 7.2 and 7.3 show the overall arrangement. Six architectural properties matter for realising framework constructs:

- **Spec-anchored storage.** Workflow state is stored in per-tenant, per-project specification files rather than an application database. A single storage service abstracts local and cloud backends while enforcing project isolation, making Spec-Anchored Continuity concrete.
- **Repository and design grounding.** Two additional context files extend the project-management

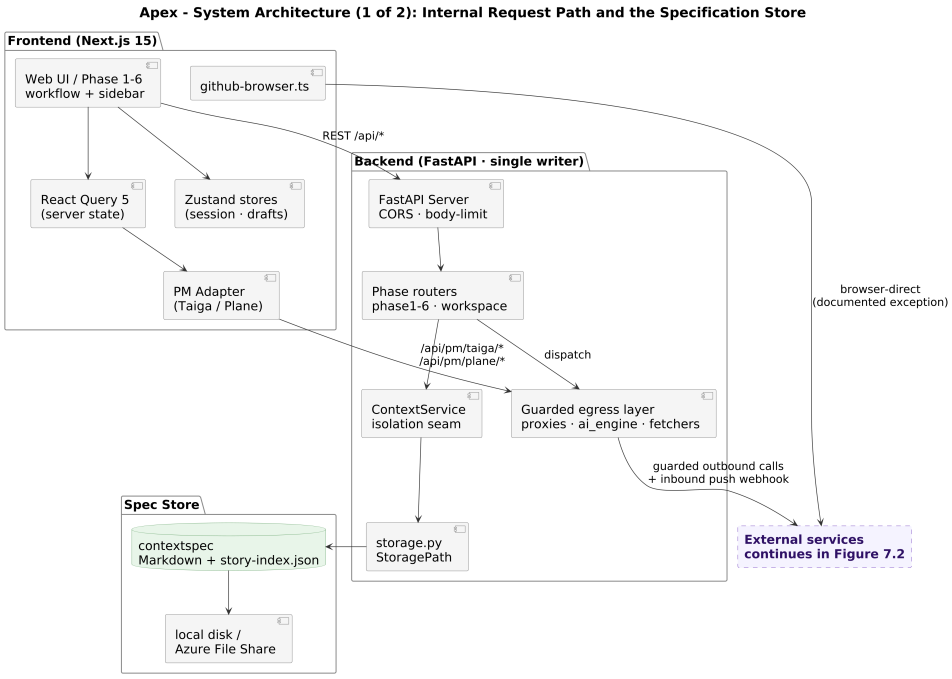


Figure 7.2: Apex architecture, part one: frontend, backend and per-tenant specification store.

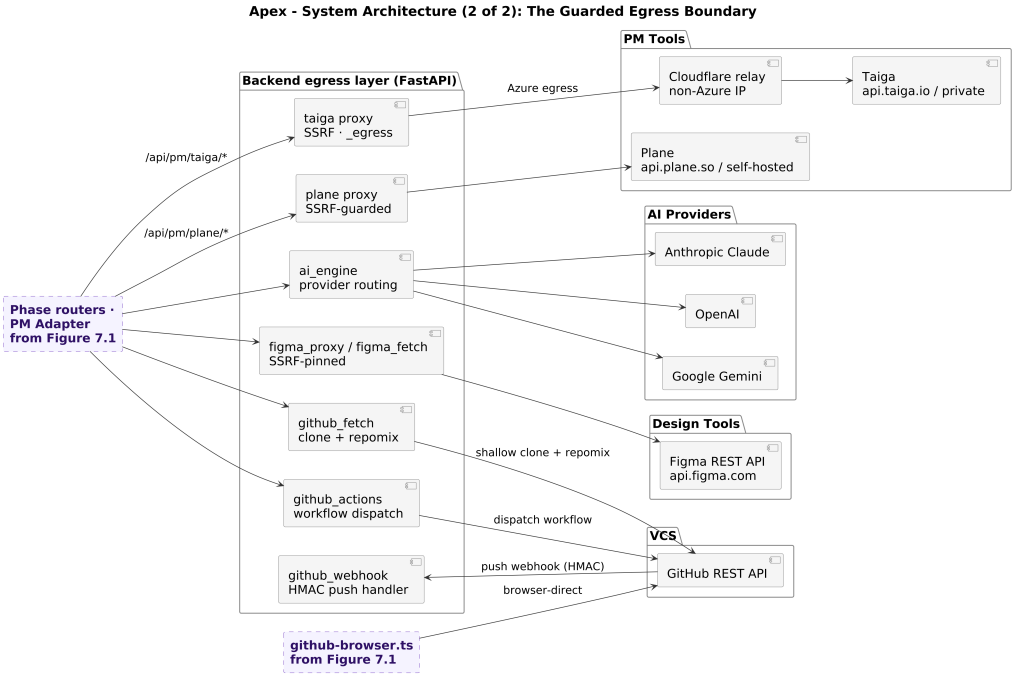


Figure 7.3: Apex architecture, part two: guarded egress layer, external integrations and the browser-direct version-control exception.

data. `github-context.md` is generated by shallow-cloning and repacking the connected repository with `repomix`, grounding Phase 2-6 generation in the actual implementation and refreshing automatically after repository pushes. `figma-context.md` provides equivalent grounding for connected

Figma files through the guarded proxy and can render a linked frame as an image, allowing developer packs to reference the designed screen directly.

- **Single-writer concurrency.** Shared file state was initially protected by process-local locks, requiring a single replica. An optional coordination layer now provides cross-replica locking while preserving the same behaviour when disabled.
- **Proxy-first external integration.** User-configurable outbound requests pass through a guarded server-side proxy that validates destinations and limits responses, reducing Server-Side Request Forgery (SSRF) risk. The browser-direct integration is the documented exception and keeps its credential outside the backend.
- **Project-management adapter.** Apex integrates with existing boards through one adapter interface. Taiga was the original and received the bulk of the implementation effort; Plane was completed behind the same interface, while the Jira adapter was started but left incomplete once it became clear Taiga and Plane were the priority targets. Tool-independence holds at the operation level, although provider-specific Application Programming Interface (API) limitations still constrain the guarantees Apex can offer.
- **Provider abstraction.** Model requests are routed through a provider-independent boundary, with vendor-specific failures mapped to common typed exceptions so the rest of the system does not branch on provider. Three providers are supported today, selected from the model identifier alone: Anthropic, OpenAI and Google.

The framework defines Spec-Anchored Continuity as a two-layer structure by lifetime but leaves file granularity open, meaning it is not a one-to-one translation. Apex implements this through a fixed set of cumulative global and project-local Markdown documents, an implementation choice rather than a framework requirement. Three additional Apex-specific artefacts extend that structure. The **developer pack** is a per-task implementation brief generated from the locked Gherkin, technical specification, design bundle and related task packs; saving it marks the task `pack_ready`. The **test plan** is the QA hat's per-story BDD script set, while the **deploy pack** records the infrastructure-delta verdict for Deployment. All three are revisable Markdown artefacts that can be exported individually or project-wide for use outside Apex.

7.3 Operationalising the Framework

This section assesses how far the framework's governance constructs can be realised in a working system and which of them resist it. Table 7.1 maps each construct to implemented mechanisms and classifies them as realised as specified, realised through a proxy, or only partially realised where implementation exposed granularity limits. Figures 7.4 and 7.5 illustrate the resulting governed practitioner workflow.

Table 7.1: Framework constructs against the mechanisms implementing them in Apex, and the outcome of each

Construct	Implemented mechanism	Outcome
The six lifecycle phases	Dedicated workflow surfaces enforce phase order through each phase's exit condition.	Realised
Quality gates	Per-story phase status controls access through the sequence from locked requirements to deployment.	Realised
Spec-Anchored Continuity	Versioned Markdown context files and a story index provide per-tenant, per-project continuity across sessions.	Realised
Stable specification identifiers	Generated identifiers for epics, entities, screens and scenarios link later artefacts back to requirements.	Realised
The Interactive Bridge	Phase 1 supports natural-language scenario drafting and compiles approved scenarios to Gherkin.	Realised
The Consistency Factor	A generated pre-implementation test set is included in each task's developer pack.	Realised
Bolts and the Fix-Bolt	Each task follows an independently timestamped micro-cycle from pack readiness to completion.	Realised
Bolt Cycle Time	Task-level elapsed time from pack readiness to completion, reported as median and 90th percentile.	Realised, after a proxy was found and replaced
Context Traceability Rate	Deployed stories are aggregated by whether their artefact chain exists and contains no unresolved traceability break.	Realised
AI Defect Escape Rate	Post-deployment, linked project-management bug reports are used as an observable proxy for escaped AI defects.	Realised as a proxy
Spec-drift detection	Drift detection is applied only where specification items have stable story-level ownership.	Partially realised, resisted at the technical-specification granularity
Server-side reconciliation of generated output	Generated coverage claims and task dependencies are revalidated against the parsed specification before acceptance.	Realised, and added because it was needed

Three rows deserve emphasis because they mark the limits of realisation:

- **Bolt Cycle Time.** The initial implementation measured story-level gate transitions while presenting the result as the framework's task-level metric. Correcting it showed that a construct can appear implemented while actually measuring a proxy, and that only comparison with the written definition exposes the mismatch.
- **AI Defect Escape Rate.** The construct remains observable only through a proxy: reported, linked, post-deployment defects. The system cannot directly identify every production defect or reliably attribute it to generated logic, so the metric is deliberately retained as a weaker but still useful operationalisation rather than presented as full implementation.
- **Spec-drift detection.** Implementation revealed that drift detection requires story-scoped specifications to remain attributable. Because the technical specification lacks that granularity, detection was removed there, while the story-scoped functional specification supported a partial revival, refining an unstated framework assumption.

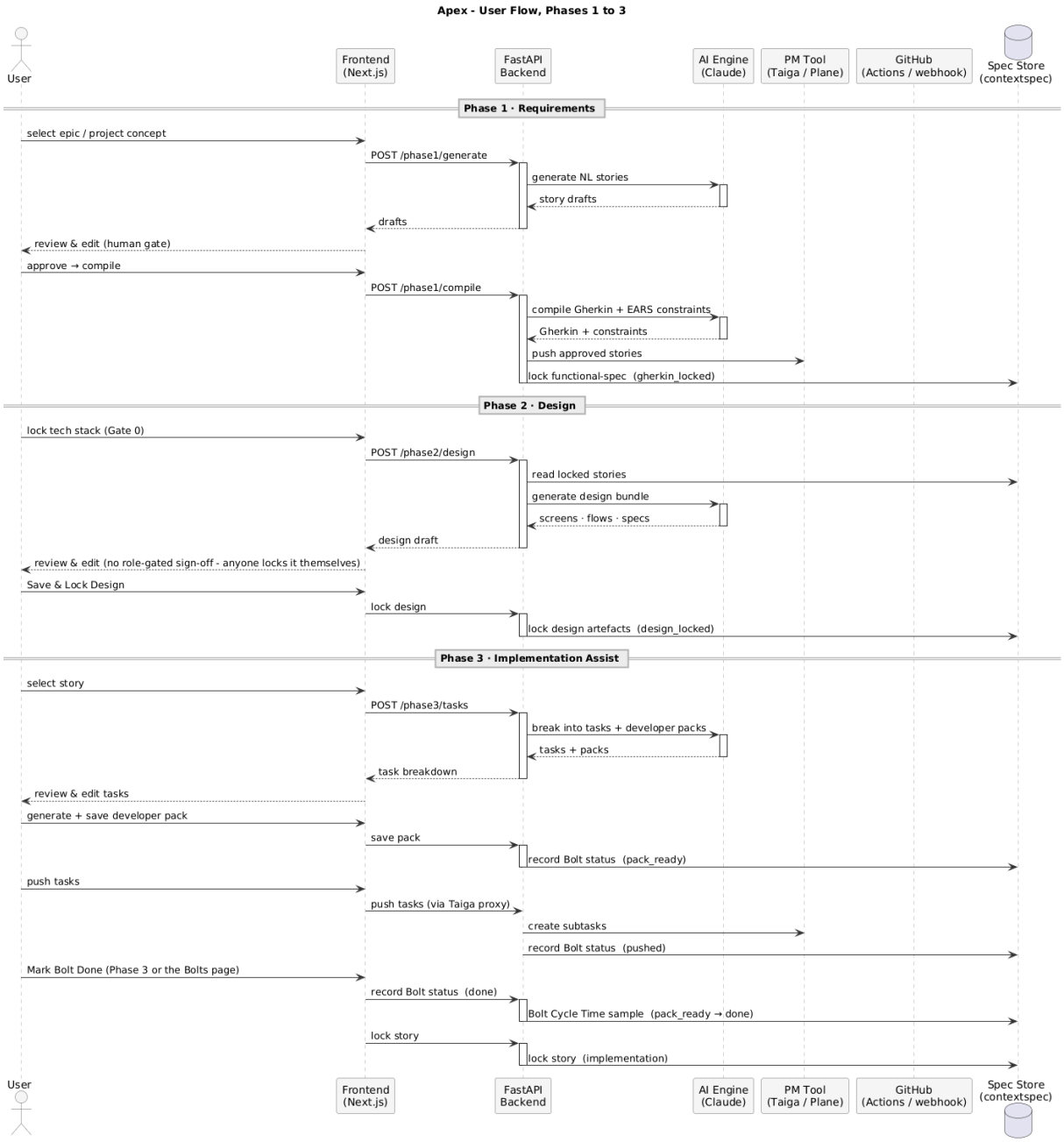


Figure 7.4: A representative walk through Apex, Phases 1 to 3. Each phase ends in a specification-store write and human acceptance, so no phase can be exited by generation alone.

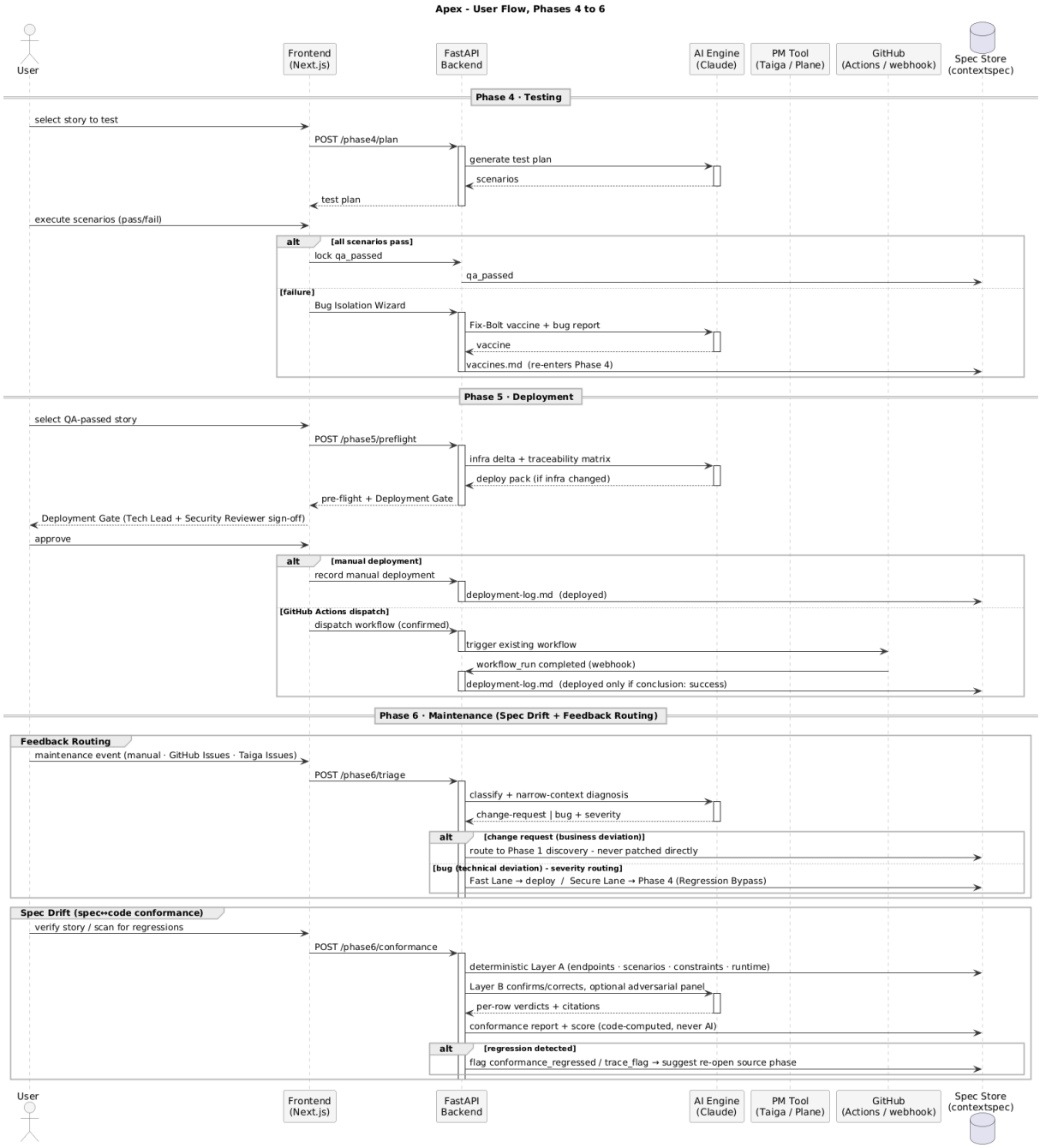


Figure 7.5: A representative walk through Apex, Phases 4 to 6. Loop-backs from Testing and Maintenance re-enter named phases whose gates apply in full.

7.4 The Governance Analytics Surface

Table 7.1 establishes which governance constructs are realised; this section describes how the corresponding measures are produced. Apex computes all governance analytics on demand in a single backend service using the story index and context files already generated by the workflow. At project scale, this requires no caching. The measures are therefore derived from operational state rather than practitioner or model self-report, and most require no separate instrumentation step. The surface exposes six figures:

- **Context Traceability Rate.** The proportion of deployed stories whose context chain remains complete. Apex verifies the required artefacts, confirms the story in the deployment log, and rejects stories with unresolved backward-trace flags, unacknowledged conformance regressions, or incomplete verification records. The metric deliberately does not claim item-level ownership within the technical specification, since that relationship is not represented by the system.
- **Bolt Cycle Time.** Computed per task from the earliest `pack_ready` timestamp to the earliest `done` timestamp and reported as median and 90th percentile in hours. Story-level gate-transition time is retained separately so queueing and review are not confused with the task-level quantity defined by the framework.
- **Spec Conformance Rate.** The mean Phase 6 conformance score, from 0 to 100, across eligible stories with a saved conformance report. Stories that were never checked are excluded from the mean; coverage is therefore displayed separately as checked versus eligible stories.
- **Fix-Bolts.** Counts corrective micro-cycles raised before deployment, reported as a total, the number of affected stories, and the mean per story. Because these defects are caught by quality assurance, a higher count represents detected pre-deployment correction rather than escaped failure.
- **AI Defect Escape Rate.** The proportion of deployed stories linked to at least one confirmed post-deployment bug recorded after deployment. This remains a proxy: unreported or unlinked defects are invisible, and Apex cannot determine whether a defect originated specifically from generated code.
- **Stories Tracked.** The total number of indexed stories, together with the deployed subset. This provides the denominator needed to interpret the other proportions and aggregates.

The measures differ in the provenance of their inputs. Bolt Cycle Time, Context Traceability Rate, Spec Conformance Rate and Stories Tracked derive entirely from state Apex writes while the workflow executes. Fix-Bolts and AI Defect Escape Rate additionally depend on issue data from the connected project-management tool, which Apex can read but cannot guarantee to be complete. Their interpretation therefore carries stronger caveats. The analytics surface is treated here only as an instrument; its readings on the evaluated project are reported in Chapter 9.

7.5 Design History and Practitioner Feedback

Feedback gathered during the construction of Apex informed both the framework and the tool. To make the DSRM iteration explicit, the changes below are reported as observations and resulting design responses rather than as a release history. Reverted changes are included because they also reveal where assumptions failed in practice. Two changes affected the framework itself:

- **Removal of a dedicated DevOps role.** An external DevOps Alliance originally held deployment approval and infrastructure-security responsibilities. Practitioner feedback showed that a role defined mainly by supervision is unlikely to be staffed consistently, so those responsibilities were reassigned to the Deploying hat, held by whoever performs the deployment. The resulting framework rationale is discussed in Section 6.5.
- **Removal of mandatory dual sign-off in Phase 2.** Design locking originally required two separate approval steps, one from the Design Lead hat and one from the Tech Lead hat. In smaller teams this either blocked progress or reduced the gate to duplicate approval by the same person. The two steps were therefore merged into a single evidence-based gate, still owned jointly by both hats, as reflected in Table 6.5.

Three further changes affected the Apex implementation:

- **Design-conflict detection.** File overlap within an epic generated false positives, so detection was narrowed and ultimately removed when it proved insufficiently reliable.
- **Spec-drift detection.** Project-wide technical specifications caused changes to cascade across unrelated stories, one real edit flagging 44 stories at once. The detector was removed at that granularity while its audit trail was retained, leading to the partial implementation described in Section 7.3.
- **Authentication for a design integration.** An authentication flow requiring an operator-registered application and backend-held secret was reverted because it conflicted with Apex's zero-configuration deployment model.

The formative feedback came from five practitioners consulted individually during construction, with one consulted again after substantial revision. The sessions were informal: there was no interview guide, fixed question set, recording or demographic collection. Feedback was captured in contemporaneous author notes and linked to the resulting changes. This record is therefore suitable for showing that the design changed in response to use, but not for estimating the prevalence or strength of practitioner opinion.

Three of those practitioners later participated separately in the semi-structured interviews reported in Section 9.3.4, where they are coded T2-T4. The two evidence sources serve different purposes: this section documents formative design change, whereas the interviews provide a structured account of practitioner views using a written guide and consistent thematic analysis.

The second consultation round produced the most consequential reframing. Feedback challenged the work to distinguish the framework from a general-purpose coding agent, justify it against explicit alternatives, state its relationship to existing models more precisely, and present Apex as an instantiation of the method rather than as the primary contribution. Those points are reflected in Section 7.1, Section 6.10, Section 6.6 and Chapter 8, making the revision traceable across the final dissertation.

A later sixth consultation with an Agile/Scrum expert is treated separately in Section 9.3.4. It evaluated the demonstration argument rather than shaping the framework during construction and produced no direct change to either the framework or Apex.

7.6 Quality Assurance of the Implementation

This section establishes that the platform the demonstration in Chapter 8 runs on is sound; it is not itself a research contribution, and the detail below is kept proportionate to that purpose.

Testing is layered across the system, with approximately 1,638 backend tests, 318 frontend unit tests, and 24 end-to-end cases across 12 specifications at the time of writing. Several defects exposed methodological weaknesses. These included specification truncation during parsing, a quality gate being marked as passed without verification, and cross-project data leakage through tenant identification and tool integration. Scenario coverage was also initially accepted from the model at face value; it is now reconciled server-side against the parsed specification, with unmatched claims discarded.

7.7 Limitations of the Implementation

The limitations of Apex as a system must be separated from the limitations of the framework as a methodology, because conflating them would let a defect in one be read as a refutation of the other.

System-level limitations:

- The single-writer fallback path, which most deployments would run, is a real ceiling rather than a theoretical one (Section 7.2).
- Apex observes only what passes through it: it can see a story's specification chain, and, once a defect is reported through the project-management tool and linked to a story, a proxy signal that a deployment shipped a fault, but it cannot observe a defect never reported there, or attribute one to generated logic specifically.

Framework-level limitations exposed through implementation:

- **AI Defect Escape Rate, at the observation point the framework's own definition presumes, is not instantiable by this class of system**: nothing in the framework says who owns production telemetry for generated logic specifically, and a tool sitting above a project-management board has no access to it regardless of who does; what is instantiable is the weaker proxy described in Section 7.3.
- **Spec-drift detection presumes a granularity the framework does not state**, namely a per-story rather than a per-project specification, and the instantiation revealed the omission by failing on it; it now holds for the functional specification and remains open for the technical one.

Both framework-level limitations remain unresolved and are carried into Chapter 9 as findings. Table 7.1 names both proxies the implementation produced. Bolt Cycle Time became one accidentally and was replaced, whereas AI Defect Escape Rate was designed as one from the outset and remains one. Naming the substitution is essential to avoid conflating the proxy with the construct itself.

8

Demonstration

Contents

8.1 Demonstration Design 42
8.2 Context 45
8.3 Application of the Framework 46
8.4 Observations 50
8.5 Summary 53

This chapter reports the Demonstration activity of the DSRM: the application of the framework, through its reference implementation, to a real software development project. The purpose of the demonstration is to show that the artefact can be used to address the research problem in a genuine setting, and to generate the evidence on which the evaluation in Chapter 9 rests. Together with that evaluation, this chapter addresses the realisation and practitioner-support dimensions of the problem: whether Chapter 7's realised constructs survive real use, and whether the framework gives practitioners usable, cognitively sustainable support in a real setting.

8.1 Demonstration Design

This section is written before the demonstration is performed. Committing in advance to what will be observed, and against which criteria, is what separates a demonstration from an anecdote, and it is the only protection against the reasonable suspicion that the observations reported later were selected because they suited the artefact. Where a statement below turns out to be wrong once the work is done,

it is corrected in Section 8.4 and the correction is marked, rather than the commitment being quietly revised.

Two settings, a third attempted, and what each can and cannot show

The framework is demonstrated in two complementary settings, while a third was attempted but did not materialise. The objective is not to demonstrate superior code generation, but to establish whether the framework's governance constructs, specification-based gates, Spec-Anchored Continuity and classified loop-backs can be operationalised (Section 7.5).

- **Apex itself.** Apex is the framework implemented as executable software, demonstrating that its governance constructs can be realised rather than merely described. In DSRM terms, it provides an inspectable existence proof: phases are gated, artefacts are traceable to requirements, and generation alone cannot satisfy a gate. However, this retrospective demonstration establishes technical feasibility, not practical usefulness, team acceptance or transferability. The construct-by-construct evidence is presented in Section 7.3.
- **Outfolio.** A separate portfolio and project-documentation platform for developers, built using the framework after its design was fixed. Unlike Apex, it applies the method to an unrelated software domain and exercises the complete lifecycle, Phases 1 to 6, through the development and deployment of a complete product. However, as an author-developed project, it remains subject to researcher-role bias and cannot independently establish transferability.
- **Partner organisation (attempted).** A partner organisation was identified as a potential third setting because it already managed its own projects in Taiga, the project-management tool Apex supported at the time. Partway through the thesis the organisation migrated to a different tool, Plane, which Apex did not yet integrate with; building that adapter (Section 7.2) was undertaken so the setting could still proceed, delaying rather than ruling it out. Despite repeated outreach afterwards, communication with the organisation did not lead to a scheduled engagement, and participation did not materialise within the demonstration window. Consequently, neither completed setting establishes transferability to independent teams. Practitioner and Agile-expert interviews address this question in Section 9.6, but provide reported opinions rather than evidence of observed application.

Unit of analysis, scope and duration

The unit of analysis is **one Unit of Work carried from Strategic Alignment through to a deployed increment**, not an individual task. That level is chosen because it is the smallest unit at which every gate in Table 6.5 is exercised at least once, and because the framework's own claims are made about the passage of work through phases rather than about the output of any single phase; where a setting's scope exceeds one unit, as Outfolio's does, the unit remains the level at which observations below are organised.

Outfolio is scoped as a small but complete product rather than a single unit run in isolation, so that more than one unit is observed and the framework's behaviour across units, not only within one, is part of the record; the epics and stories fixed for it are stated in Section 8.2. The instantiation setting has

no unit count, for the reason given above. The demonstration runs until Outfolio's units reach either a deployed increment or an abandonment recorded with its reason, whichever comes first; an abandoned unit is reported, since a framework that cannot carry work to completion in a real setting is a finding. The fieldwork this section requires is complete. No unit was abandoned, and Sections 8.3 and 8.4 report what the demonstration log establishes.

What is recorded, decided in advance

At every gate crossed, in every setting, the following is recorded: the artefact presented as evidence, the decision taken, the hat that took it, and, where the gate failed, the named phase the work was routed back to and what changed before it returned. Loop-backs are recorded as first-class events rather than as exceptions, for the reason stated in Section 6.6, since a demonstration in which nothing was rejected and nothing looped back is evidence that the framework was not exercised, not evidence that it works.

Alongside the gate record, three further classes of observation are pre-committed. First, the governance metrics the framework defines in Section 6.7.4, to the extent the instantiation computes them, with any it does not compute, and any computed only as the proxies Table 7.1 names, reported as such rather than silently presented as the construct. Second, every point at which generated output was modified or rejected by a human, together with what was wrong with it, since the framework's central premise is that this happens and is caught. Third, every point at which the framework was found awkward, over-specified, or was bypassed in practice; these are recorded as they occur rather than reconstructed afterwards, and they are the most valuable input to Chapter 10.

Two commitments about reporting are made here so that they cannot be reconsidered later. Observations unfavourable to the artefact are reported with the same prominence as favourable ones. And the artefacts themselves, generated scenarios, specifications, task breakdowns and test plans, are reproduced rather than described, so that a reader can judge their quality instead of accepting a characterisation of it.

The researcher's role, and the threat it creates

The author designed the framework, built its instantiation, and is the participant in Outfolio. This is the most serious threat to the validity of this chapter and it is named here rather than deferred to Section 9.5.

The threat is not primarily that the author would misreport what happened. It is that a designer applying his own method is the person least likely to notice where it is awkward, because he already knows what it is for, supplies missing rationale from memory without registering that he is doing so, and reads an artefact as adequate against an intention rather than against what is written. The instantiation setting carries a related but distinct problem, because the framework and the system were designed together, a construct that proved inconvenient could be revised in the method rather than recorded as a difficulty, and Section 7.5 reports two occasions on which exactly that happened.

Two things mitigate it and neither removes it. Pre-registering the record above, for the reason given at the head of this section, is what makes a later omission visible. And the gate record is defined against artefacts that exist independently of the author's account of them, so a claim in this chapter can be checked against the specification store rather than taken on trust. The one mitigation that would have been structural rather than merely procedural, a setting in which someone other than the author

operated the framework, is exactly what the abandoned partner-organisation setting above would have supplied; without it, this threat is carried by procedure alone.

What remains unmitigated is stated plainly. For both settings, this chapter reports what a designer observed while using his own design, and no arrangement within this study makes that equivalent to independent application.

8.2 Context

Outfolio addresses a problem reported by a professional OutSystems developer and colleague. Although the platform exports application logic and data models, it preserves no visual record of the application's appearance or behaviour, making completed work difficult to present outside the platform. The problem extends to developers whose projects remain tied to environments that do not export user-facing artefacts. Outfolio addresses this gap through platform-independent project case studies combining screenshots, architecture notes, problem descriptions and outcomes. The complete implementation is published at `https://github.com/thomastabs/OutfolioV1`, with the deployed product itself available at `https://outfolio-v1.vercel.app/`.

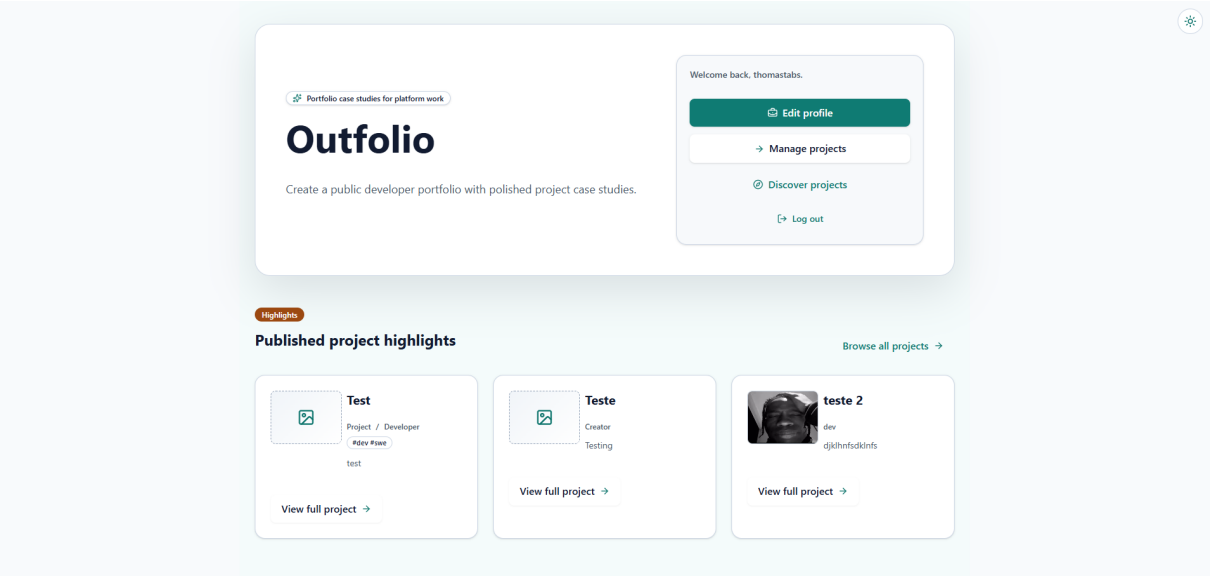


Figure 8.1: Outfolio's dashboard: the deployed product built for this setting's lifecycle demonstration, showing the project owner's published case studies.

The initial scope, recorded in the project's agent-instruction file, comprised six epics and approximately eighteen stories, deliberately limited to support a complete lifecycle demonstration. Three additional epics were subsequently introduced through documented human decisions: a dashboard and visual design system on 16 September 2026, followed by media and file attachments on 20 September through a Phase 6 loop-back after two post-deployment signals were classified as change requests. The resulting product comprised nine epics and fifty-four stories, developed over nine calendar days and 123 commits. The significance of this expansion lies not in its volume but in the recorded decisions and classifications governing it (Section 8.3).

The author performed all eight hats of Table 6.3 while also maintaining Apex. This exercises the hats-not-people principle in Section 6.5, but limits independent oversight since each gate is a self-check rather than a separate review. The demonstration therefore examines whether evidential requirements remain enforceable when multiple responsibilities are held by one practitioner. Outfolio was developed from a greenfield baseline, with its stack fixed at the initial design gate: Next.js 14, React 18 and TypeScript; Express-style handlers, Prisma and PostgreSQL; Supabase for database hosting, credentials and private storage; NextAuth for sessions; and Vercel deployment through GitHub Actions. Testing used Jest, React Testing Library and Playwright. No tests or isolated development database existed initially; the resulting repository contained 62 unit and component test files and three browser specifications, produced during Phases 3 and 4. The baseline exhibited four problems corresponding to the consequences identified in Chapter 4 and table 4.5:

- **Inconsistent validation.** Generated code was reviewed by the same practitioner who prompted it, without an independent or repeatable verification process.
- **Requirements pressure.** Rapid code generation encouraged implementation before specification, allowing scope to expand through incremental additions rather than explicit decisions.
- **Testing complexity.** Frontend components, API handlers, database migrations and hosted services required different validation approaches, while browser testing lacked an isolated database.
- **Undefined accountability.** With one practitioner and one coding agent, the commit history recorded changes but did not distinguish human decisions from those delegated to the agent.

8.3 Application of the Framework

Outfolio was carried from Strategic Alignment to a deployed increment over nine calendar days, 14 to 22 September 2026, under a demonstration log of 422 dated entries recording, at each gate, the artefact presented, the decision taken, the hat that took it and the phase any failure was routed back to. Eight epics and thirty-six stories were locked, built, tested and deployed in the first pass; a further twelve stories, eight of them a new epic, and six individual stories after those, entered later through Maintenance loop-backs. Every claim below is cited by date and story identifier so that it can be checked against that log rather than accepted as a characterisation of it. What follows walks the six phases of Section 6.6 in turn, and holds the loop-backs back for the separate passage that closes the section, since treating them as exceptions inside a phase walk would misstate how much of the work actually went through them.

Discovery and Requirements

The Interactive Bridge ran as specified. The model drafted fractional stories in natural language per epic, the human confirmed, modified or deleted them, and only the approved set was compiled into Gherkin carrying stable scenario identifiers; the steering was explicit rather than passive, entered as free text in a dedicated field before generation. The Discovery Gate was accepted at 16:24 on 14 September for all six initial epics at once, locking the functional specification project-wide. Two unfavourable observations

belong to the gate itself. The hat metadata for that acceptance was, by the log's own note, supplied after the gate statement had been written, so the accountable hat trailed the decision instead of preceding it; and when two further epics were added mid-stream on 16 September, Home Dashboard was drafted at 22:58 and gherkin-locked at 23:06, eight minutes later. Neither is a violation of the gate as Table 6.5 defines it, since the required evidence existed in both cases, but a gate whose owner is named retrospectively and whose evidence is examined in eight minutes is being satisfied at the lower bound of what the control was written to mean.

Design, Prototyping and Architecture

The Design Gate was crossed at 17:09 on 14 September, forty-five minutes after the Discovery Gate, producing the anchored technical specification and design bundle for the whole initial scope. This is the phase where the solo setting bears most directly on the control. Table 6.5 assigns the gate to the Design Lead and Tech Lead hats jointly, and across 422 entries that joint hat is recorded exactly three times, all at Phase 2. The framework's "hats, not people" design in Section 6.5 anticipates one person wearing both, but it does not make the second reading independent of the first, and nothing in the demonstration establishes that it was. What the phase did demonstrate is the amendment discipline. The specification was not frozen but re-opened and re-versioned, once as a recorded major amendment on 20 September at 23:01 when the Maintenance loop-back described below added a new epic, and again with a major design-version bump for story `9564046` on 22 September.

Implementation

Implementation ran from 15 to 18 September for the original scope, each story decomposed with the model into tasks, each task executed as a Bolt against a developer pack carrying the Consistency Factor test set. The Authentication and Session Management epic is representative. Four stories were implemented between 13:51 and 14:41 on 15 September, and the epic concluded at 14:43. Two recurring frictions belong to the record. The first is that the model repeatedly had to be warned against duplicating work already in the repository, a near-identical caution recurring across stories `9543460`, `9543480` and `9543481`, which is the cost of a cumulative project-wide specification rather than one locked per story, as Section 7.3 already concedes. The second is more serious. On 21 September at 20:35, while implementing a seeding task, the agent executed a destructive reset script against the live production database without re-checking which environment the connection string pointed at, wiping all six tables, its root cause discussed in full in Section 8.4. Work stopped, the human was informed immediately, and an isolated database project was provisioned before implementation resumed.

Testing

All thirty-six stories of the original scope reached a passing story-level verdict between 18 and 19 September, tested against the locked Gherkin rather than the implementation. The gate was not a formality. Story `9552560` failed on one scenario while its two siblings passed, with the failure attributable to component code the application had already moved past (Figure 8.2). Two username-casing defects

were caught the same way on the registration and login stories. The phase also produced the demonstration's clearest evidence for why Section 6.6 decouples Testing at all. A session-gating defect was, in the log's words, "invisible to the entire existing automated test suite by construction", because every affected component's tests mocked the exact library whose missing backend was the defect. The manual edge-case probing the QA Validation Playbook mandates, and not the generated scripts, is what surfaced it.

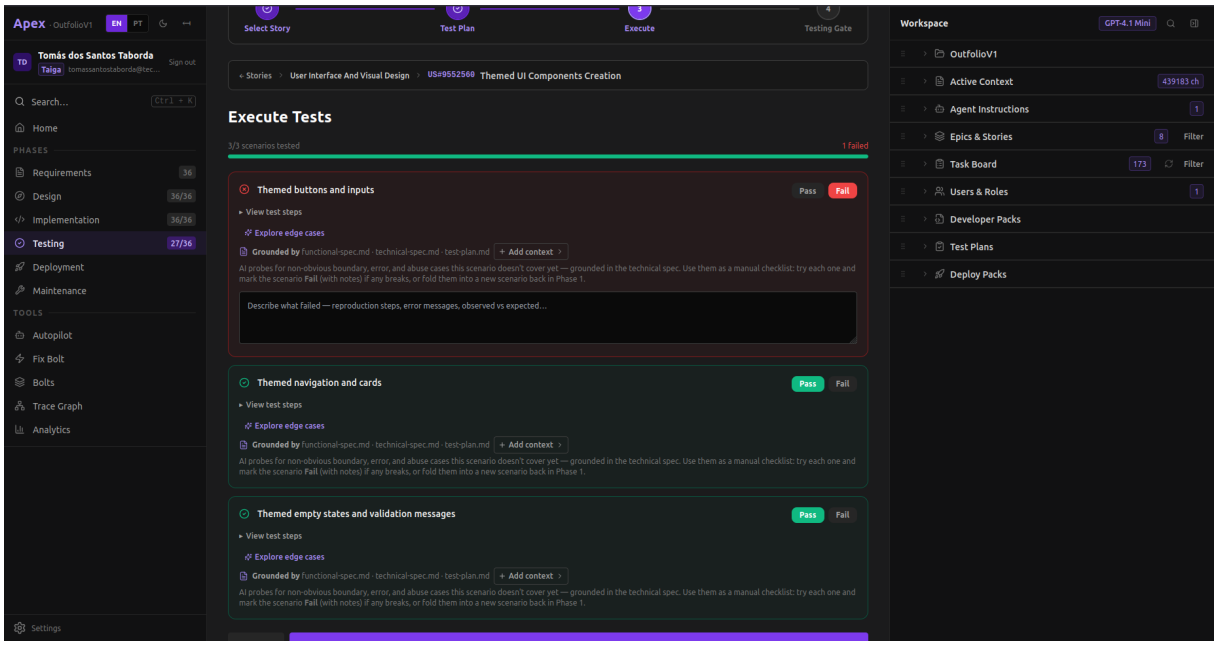


Figure 8.2: Phase 4 Testing for US#9552560: one failed scenario, two passed, and AI-generated edge-case probes provided as a manual checklist.

Deployment and Maintenance

The first production deployment was story 9543457 on 19 September at 20:54, and Phase 5 closed for all eight epics at 12:41 on 20 September. The gate presents the infrastructure-delta verdict, the deployment pack and the traceability matrix in one view, with the human sign-off unchecked until the operator checks it (Figure 8.3). Two findings run against the artefact. The infrastructure-delta answer was wrong twice in opposite directions and overridden both times (Section 8.4), which is the gate working, but an advisory answer that is wrong in both directions supplies no information. And the gate's own required evidence includes a security review, not invoked as a distinct step during the Fix-Bolt deployments of 18 September, a control satisfied nominally by the same hat that authorised the push.

Maintenance opened on 20 September with ten items accumulated from live use. Each was classified before any code was opened, and the classification determined the route rather than being recorded after the fact (Figure 8.4). Nine items were resolved by 02:20 on 22 September, and the tenth, dark-mode support, ran its own full cycle afterwards. Restraint is part of the record too: a scope discrepancy on story 9543480 was deliberately resolved as vacuously satisfied rather than opened as new Discovery work (Section 8.4). The handoff itself was also where the instrument was weakest, because the control

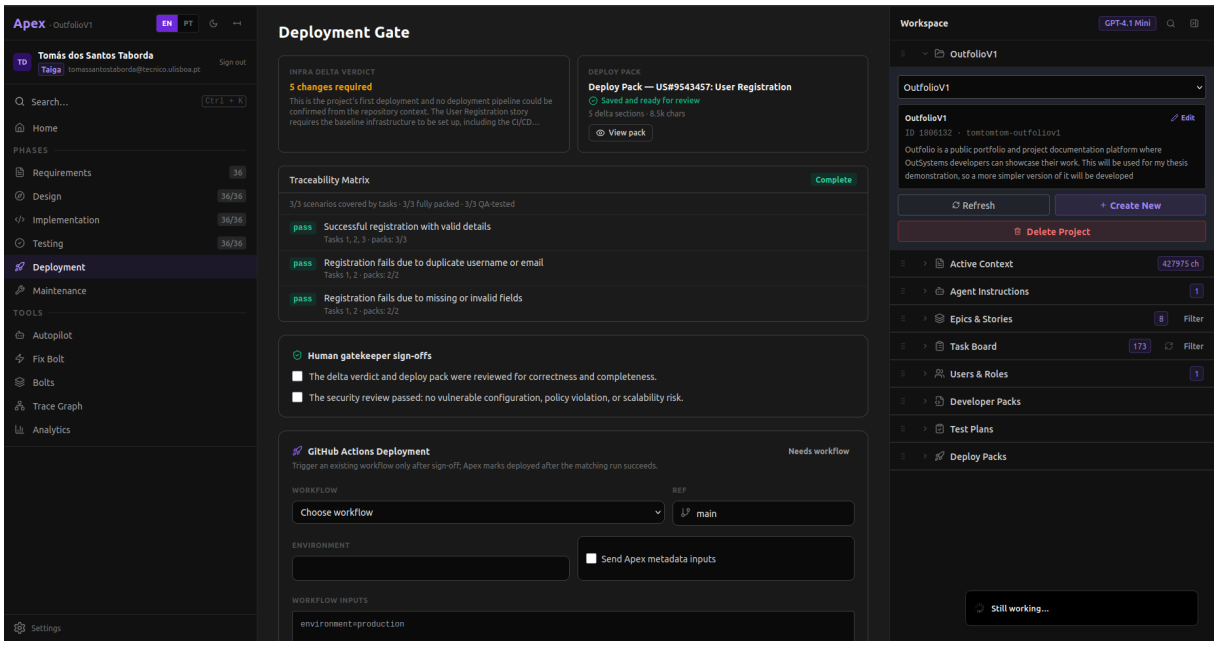


Figure 8.3: Phase 5 Deployment Gate for US#9543457: infrastructure assessment, deploy pack, complete scenario traceability and pending human sign-offs.

that opens an item in Phase 1 carried no payload and silently discarded the change request's content, a defect visible only by using it.

Loop-backs, rejections and regressions

The demonstration exercised all three routes out of Maintenance that Section 6.6 names, and the largest ran on 20 September, when two items classified as change requests were approved by the author under the PM hat, as the sole hat-holder, and routed through Discovery into Phase 1, producing twelve newly gherkin-locked stories: four riders on the Home Dashboard epic and a new eight-story epic, Project Media and File Attachments. The design delta was merged as a major amendment at 23:01, implementation ran until 12:30 the next day, and all twelve stories passed Testing between 13:10 and 16:55 before deploying that evening. The log marks it as "a deliberate framework loop-back from Phase 6 maintenance intake through requirements into Phase 2 design, not an implementation shortcut". Three further items, the end-to-end test layer, storage hardening and dark mode, each ran the same complete Phase 6 to Phase 1 to Phase 5 cycle independently over the final two days.

Generated governance output was rejected as well as generated code. When the end-to-end test item reached Phase 1, the model's own single-story draft was refused at the Discovery Gate as underscoped, on the ground that it bundled two kinds of work into something that was not a boundable, closeable scope, and was split into two stories before locking. More pointedly, the Fix-Bolt Brief that the tool itself generated for story 9552560 was read and explicitly not followed, because it directed a refactor onto the legacy component system, the opposite of the direction already agreed; the human was consulted before proceeding differently. An instrument whose own generated directive can be overridden at the point of use is the premise of Section 6.7.3 holding in the one case that tests it.

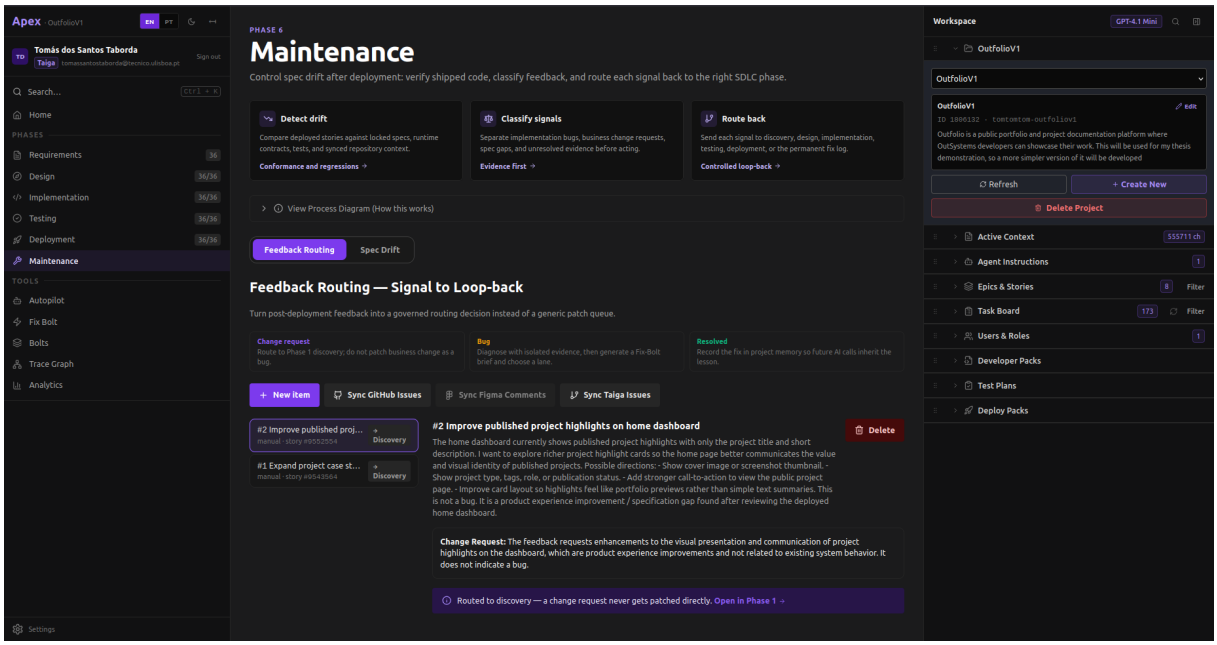


Figure 8.4: Phase 6: a post-deployment change request classified and routed to Phase 1 Discovery rather than patched directly.

Testing returned real failures throughout. Story `9560176` failed because an unauthenticated visitor could not reach a correctly implemented download endpoint through any user interface path; story `9560175` failed on a non-atomic upload that partially persisted a batch it reported as entirely rejected; story `9563893` failed on a non-atomic create leaving partial records; story `9564192` failed on a crash that, in the log's assessment, "would have broken every single page of the app". Each triggered a Fix-Bolt with the failing evidence attached and was retested before release, and by the end of the demonstration the Fix-Bolt record held fourteen per-story bug reports over a permanent dated log. One event is worth more than the rest and is not softened here: a Fix-Bolt introduced its own regression. The session-context fix committed as `6c17344` caused a redirect loop after logout, which a second Fix-Bolt, `ad14660`, had to repair the same evening, logged at the time as a defect introduced by an earlier Fix-Bolt in the same session. A remediation cycle that can itself break the system is the strongest available evidence that the loop-back mechanism is a real control rather than a ceremony, since only a mechanism that actually re-verifies would have caught it, and it is equally evidence that a Fix-Bolt executed under deployment pressure is not a safe operation.

8.4 Observations

The unit-of-analysis statement in Section 8.1 undersold the scope actually carried, and is corrected here. That subsection commits to observing "more than one unit" across "the epics and stories fixed for it", language chosen when the pre-committed scope was six epics and roughly eighteen stories. What was actually carried through all six phases to a deployed increment was a full product: nine epics and fifty-four stories, three of them added after the pre-registration through the framework's own

governed routes rather than through scope creep, as Sections 8.2 and 8.3 report. "More than one unit" is true of that outcome but understates it; the demonstration exercised the framework across a project at realistic scale, with the cross-story dependencies a project of that size actually produces. Two pieces of evidence below exist only because the scope was that large: the shared-endpoint defect that propagated across nine stories (below), and the recurring Phase 3 overlap caution (Section 8.3). Neither would have been observable at the pre-registered scope, and both strengthen rather than undercut the original unit-of-analysis choice.

The governance metrics, and what each of them actually measures

The instantiation reports on 54 stories, all of which reached `deployed`; the figures therefore describe completed passages through the lifecycle rather than work in flight. All values are taken from the instantiation's analytics and specification-drift exports, independently of the author's narrative record. Where those measurements diverge from the constructs defined in Section 6.7.4, the distinction is made explicitly.

- **Bolt Cycle Time is realised at task level, as the framework defines, median 0.22h and p90 1.4h across 254 tasks.** An early implementation measured story-level gate transitions instead (Section 7.3); the fix landed during this window, but the export this report draws on still surfaced only the story-level figures until that export was corrected too.
- **Context Traceability Rate is 50 of 54, or 92.6%** (dashboard: 93). The four incomplete chains share the same final-step ledger gap, indicating missing traceability evidence rather than a breakdown of the construct itself.
- **Spec Conformance Rate, an instantiation-added metric, averages 98.65%** (dashboard: 99) across all 54 stories. The observed drift reduces to one recurring shared-route defect affecting nine stories, which is more consistent with propagation through a common specification than with general specification failure.
- **AI Defect Escape Rate is 1 of 54, or 1.85%** (dashboard: 2), but this is a weak proxy. Its definition excludes defects found and repaired within the same session, despite 19 Fix-Bolts being logged across 14 stories.
- **Zero regressions in the export reflects missing instrumentation, not verified absence.** The exports contain no regression field or prior-score comparison, while the demonstration log records at least one regression introduced by a Fix-Bolt and repaired by a second Fix-Bolt.

Points at which generated output was modified or rejected

Section 8.1 pre-committed to recording every such point. Four classes occurred, all decision-point evidence rather than incidental corrections; the loop-backs they belong to are traced in Section 8.3.

- A generated governance artefact was **rejected outright** on story 9552560: the Fix-Bolt Brief the instantiation produced directed a refactor onto legacy components, the opposite direction already

agreed; the human overrode it before implementation proceeded, the framework's own tooling refused by the hat it was advising.

- The Pre-Flight infrastructure-delta recommendation was **manually overridden twice, in opposite directions**: on story 9560172 it classified genuine infrastructure work (two database migrations) as routine, since its evidence gathering never inspected the migrations directory; on story 9564046 it flagged four already-provisioned or self-provisioning items as required, since it never checked current deployment state.
- A generated Phase 1 artefact was **rejected as underscoped rather than incorrect**: a single story drafted from a maintenance item bundled two kinds of work into a scope that could not be closed, and was split in two before locking.
- A loop-back was **deliberately declined**: the specification-scope discrepancy on story 9543480 was resolved as vacuously satisfied rather than opened as a new Phase 1 story, a restraint decision recorded at the time because a framework that answers every mismatch with new scope is as much a failure as one that answers none.

Where the framework was awkward, over-specified, or was bypassed

Six incidents follow, the first two exposing the most consequential limitations of the framework.

- **Production data loss caused by the AI implementer.** On 21 September 2026, the model executed a destructive database reset without verifying that `DATABASE_URL` still pointed to production, wiping all six tables. With no automatic recovery, work stopped, the human was informed, and an isolated Supabase project was provisioned; a remaining authentication inconsistency was resolved manually. The incident exposed a governance gap: phase-transition gates do not prevent destructive actions within a phase. Containment was therefore entirely human and ad hoc, while the incident's self-reporting remains subject to the researcher-role threat identified in Section 8.1.
- **The Testing Gate can pass obsolete tests.** Stories 9552560 and 9552562 revealed that component tests can remain green after the application changes, while Fix-Bolts can invalidate previously generated test plans. The Regression Bypass preserves the existing suite to maintain measurement consistency, but does not verify that the suite still exercises the current implementation. Consequently, a stable testing instrument can become systematically misleading.
- **Automated validation missed defects detected through live human testing.** An icon-colour defect, a deployment terminal-screen failure and a dead navigation link passed automated checks but were identified through interaction with the running application. Despite a Spec Conformance Rate of 98.65%, these incidents demonstrate that specification conformance does not establish functional correctness. The framework requires no explicit live human verification of a deployed increment; the informal verification performed here exposed defects that its existing gates missed.
- **Governance recommendations were confidently incorrect.** Two Pre-Flight overrides produced opposite classification errors, a false negative followed by a false positive, without expressing uncertainty. Another classification was correct only because the required infrastructure was already

deployed, rather than because the story was inherently infrastructure-free. These incidents reproduce the stale-context, high-confidence pattern identified in Section 9.5, showing that an apparently authoritative governance surface can conceal incorrect reasoning.

- **Phase boundaries blurred, rendering governance metrics inaccurate.** A story requiring deployment was completed through a production release while remaining formally in Phase 3, without a distinct Phase 5 gate. Because workflow status was not updated, the analytics panel showed all 36 stories in implementation, with no conformance checks recorded, despite deployed and operational software. The bypassed gate and inaccurate metrics therefore represent the same failure: governance state diverged from actual practice, leaving incomplete traceability chains.
- **Defined responsibilities were not consistently exercised.** During Fix-Bolt deployments, the person holding the Developer and Tech Lead hats, and so also the Deploying hat, repeatedly authorised releases without its security review. Although Section 6.5 permits one person to hold multiple hats, neither the framework nor Apex detects when a responsibility is omitted. Similarly, Phase 1 hat metadata was recorded after the gate decision, rather than before it. These cases expose the difference between formally assigning responsibilities and verifying their execution.

8.5 Summary

What this demonstration establishes is that the framework's governance constructs are not merely implementable, the claim Section 8.1 pre-registered as the instantiation's own contribution, but that they were genuinely exercised under real conditions rather than walked through once cleanly. The scope grew from six epics to nine without anything abandoned; all three of the framework's named loop-back routes ran, more than once each; a generated governance artefact and two generated advisory verdicts were rejected by the human who used them; a real regression was caught by the same re-verification the loop-back mechanism exists to provide; and the traceability, conformance and defect-proxy metrics all resolved to real, checkable numbers rather than to placeholders. A demonstration in which nothing failed would have shown nothing; this one shows the gates holding under real failure, not merely existing.

What it does not establish is stated as plainly. Every observation above was made by the same person who designed the framework, built its instantiation, and is the sole hat-holder in the one fieldwork setting that ran; Section 8.1 names that threat as unmitigated, and nothing in the record above changes that, since a designer who found his own governance mechanism silently broken three times in the same chain is still the designer reporting on himself. The demonstration also surfaced limits the framework does not answer: no gate exists for a destructive action taken inside a phase rather than at its boundary, the Testing Gate can be satisfied by a suite that no longer measures the system it tests, and an advisory governance surface can report high confidence while wrong in either direction with no signal distinguishing that case from being right. These are not incidental defects of one instantiation; they are findings about what phase-boundary, artefact-checking governance can and cannot reach, and Chapter 10 takes them up as such. Chapter 9 complements this chapter with evidence from thirteen participants who did not design the framework, providing the closest approximation to the independent application the abandoned partner setting would have offered. The two forms of evidence are therefore interpreted together rather than as substitutes.

9

Evaluation

Contents

9.1 Evaluation Strategy 54
9.2 Participants 55
9.3 Instruments and Procedure 56
9.4 Results 61
9.5 Threats to Validity 65
9.6 Discussion 66

This chapter reports the Evaluation activity of the DSRM. Following Pries-Heje et al.'s distinction between *when* and *how* an artefact is evaluated [51], the evaluation carried out here is *ex post*, since it assesses the artefact after construction and on the basis of its actual use, and *naturalistic*, since it involves real users, a real project and a working system rather than a laboratory simulation.

9.1 Evaluation Strategy

Two artefacts were produced in Chapters 6 and 7: a framework and its instantiation. They are not evaluated by the same means, and conflating them would weaken both claims. Table 9.1 therefore states which instrument evaluates which artefact, and which dimension of the problem (Chapter 5) it bears on.

Table 9.1: Evaluation instruments, target artefact, and problem dimension addressed

Instrument	Artefact evaluated	Dimension	What it establishes
Demonstration in a real project	Framework and instantiation	Realisation, practitioner support	That the artefact is applicable and produces the intended artefacts and decisions
System Usability Scale (SUS)	Instantiation (Apex)	Practitioner support	Perceived usability of the supporting tool
NASA Task Load Index (NASA-TLX)	Instantiation (Apex)	Practitioner support	Perceived cognitive workload imposed on the practitioner
Apex UX questionnaire (author-designed)	Instantiation (Apex), with two items scoped to the framework	Practitioner support	Trust in AI-generated content, legibility of phase gates, and error surfacing, which SUS and NASA-TLX are not designed to reach
Practitioner interviews	Framework	Placement, governance	Practicality, feasibility, and perceived value in real workflows
Agile/Scrum expert interview	Framework	Placement, governance	Alignment with, and departure from, established delivery methodology
Analytical assessment against criteria	Framework	Placement, governance	Degree to which stated design criteria are satisfied

SUS and NASA-TLX measure Apex, not the framework. Both are administered to someone who has spent a session operating a user interface, and neither contains an item about phase decomposition, gate placement, or accountability for generated artefacts. A favourable SUS score therefore establishes that the tool is usable and nothing at all about whether the method it implements is sound; an unfavourable one, correspondingly, is not evidence against the method. That is a design decision rather than a defect. The framework's own validity is carried by the practitioner and expert interviews (Section 9.3.4), the analytical assessment against criteria fixed in advance (Section 9.3.5), and the demonstration (Chapter 8), each of which reaches the method directly.

One instrument spans both artefacts, and the split is handled rather than ignored. Two items of the author-designed UX questionnaire ask about the process rather than the product, and are analysed apart from the fifteen instantiation-scoped ones, as Section 9.3.3 sets out.

9.2 Participants

Participants were recruited through the author's own professional and personal connections. There was no open call, no advertisement and no mailing list, and the number of people approached was not recorded, so no response rate can be reported and none is claimed. Thirteen sessions were com-

pleted, coded P0 to P12; all thirteen returned all four instruments and no participant withdrew. The pre-registered target was twelve; the achieved sample is thirteen, and the sample is not drawn from a single organisation but from one person's professional and personal network, which is a different limit rather than a weaker one.

The sample comprised thirteen participants and was predominantly junior. Three were students with no professional experience, three had under one year, and seven had one to three years; none exceeded three years. By role, six were students, four developers, one an AI engineer, and two worked outside software development. Exposure to AI coding assistants ranged from daily use (four participants) to no prior use (one), with the remainder distributed across weekly, monthly and occasional use. Familiarity with the session's supporting tools was limited. Nine participants had no prior exposure to Gherkin or BDD, and ten reported no familiarity with Taiga. Nine participants selected English as their session language and four Portuguese; questionnaire language differed for two participants, as discussed in Section 9.5. The sample therefore supports only bounded claims. Thirteen respondents meets the twelve at which a SUS mean is generally treated as stable, so the mean is reported alongside its distribution and individual scores, but the convenience sample remains junior-skewed and drawn from a single network; the resulting limitations are discussed in Section 9.5.

- **Consent.** All participants provided informed consent before data collection through separate choices for participation, anonymised quotation and follow-up contact. All thirteen consented to participate; eleven consented to quotation and eleven to follow-up contact, the two sets overlapping at nine, with quotations restricted accordingly. The consent form is reproduced in Chapter B. No institutional ethics submission was required under the guidance obtained for this study.
- **Data collection.** Sessions and interviews were not audio, video or screen-recorded. Collected data comprised questionnaire responses, submission timestamps and contemporaneous interview notes written up immediately after each interview. The limitations of note-based rather than transcript-based analysis are addressed in Section 9.5.
- **Anonymisation and retention.** Participants are identified only as P0-P12; names, employers and identifying project details were not retained with responses. Only wording recorded verbatim is presented as quotation, with other interview material explicitly paraphrased. Anonymised responses and interview notes are retained with researcher-only access as described in Section B.2, permitting the reported aggregates to be recomputed. Live links to every instrument below, in both languages, are collected in Section B.1.

9.3 Instruments and Procedure

The usability study was administered entirely unmoderated. Participants worked independently, at their own pace, from a fixed written task script and completed the questionnaires online without the researcher present. Without recordings, observation would have produced only facilitator notes, and because the author designed both artefacts under evaluation, facilitation would also have introduced a direct risk of steering, hinting or reacting to participants. The trade-off is the absence of measured task completion, time on task, assist counts and observational evidence of where participants struggled, which is discussed in Section 9.5.

9.3.1 System Usability Scale

The SUS [52] is a ten-item, five-point questionnaire administered after task completion. Participants using Portuguese received the validated European Portuguese version [53]; those using English received the original wording, both reproduced in Section B.4. The two language groups are reported separately rather than pooled.

Scoring follows the standard procedure: odd-numbered items contribute the response minus one, even-numbered items contribute five minus the response, and the total is multiplied by 2.5 to produce a score from 0 to 100. The result is not a percentage and is therefore interpreted using published reference scales rather than in isolation.

Two interpretations are reported. Bangor et al. associate score ranges with qualitative labels, with reference points including *Poor* at 35.7, *OK* at 50.9, *Good* at 71.4 and *Excellent* at 85.5 [54]. Sauro and Lewis instead map scores to percentile ranks and letter grades against their corpus, where the commonly cited mean of approximately 68 corresponds to the fiftieth percentile rather than to a pass threshold [55]. Both are reported in Section 9.4.

The Lewis and Sauro two-factor decomposition is also reported: **Usability** from the eight items other than 4 and 10, and **Learnability** from items 4 and 10 [56]. These dimensions are useful because a tool may be difficult to learn while remaining usable once learned, or perform poorly on both.

The Portuguese results are interpreted cautiously. The validation study reported acceptable construct validity against the Post-Study System Usability Questionnaire ($r = 0.70$), but weak inter-rater reliability (ICC $= 0.36$, 95% CI 0.01-0.63) and 76.67% agreement, below the authors' 80% threshold [53]. Since only three participants completed the Portuguese version here, those results are reported separately and treated as indicative.

9.3.2 NASA Task Load Index

The NASA-TLX [57] measures perceived workload across mental demand, physical demand, temporal demand, performance, effort and frustration. This study uses the unweighted Raw NASA-TLX [58], administered after each marked task rather than once at the end of the session, thereby producing a workload profile by lifecycle phase.

The Raw variant was chosen because per-task administration would otherwise require ninety pairwise weighting comparisons per participant. More importantly, Hart's review of 550 studies found no consistent sensitivity advantage for the weighted form across direct comparisons [58]. The six subscale scores, which are the primary results in this study, are identical under both variants; weighting affects only the aggregate.

Two departures from the published instrument were made. Responses used a 0-10 linear scale rather than the original twenty-interval line and were rescaled to 0-100 by multiplying by ten. This simplifies unmoderated administration but reduces resolution, so small differences are not interpreted as meaningful and the values are not treated as directly comparable with studies using the original response scale. The reversed performance scale was also placed last rather than fourth to reduce the risk of inversion errors during an unmoderated session. Neither change alters the Raw NASA-TLX computation. Repeated administration may introduce anchoring and fatigue, particularly on later tasks. These risks are discussed in Section 9.5 and weighed against the benefit of obtaining a phase-specific

workload profile. Both instruments were applied to one fixed task set, identical in wording and order for all participants. Nine tasks carried a single password-reset Epic from requirement to deployed increment:

Task	Activity
1	Sign in and open the seeded project.
2	Connect the project to a GitHub repository.
3	Convert the requirement into locked Gherkin.
4	Produce and lock the design.
5	Sign out and resume the session.
6	Decompose the story into implementation tasks.
7	Generate a test plan and record the QA sign-off.
8	Deploy the feature.
9	Export the resulting artefacts.

Tasks 3, 4, 6, 7, 8 and 9 each carried a Raw NASA-TLX form; tasks 1, 2 and 5 were setup or session-resumption steps and were not rated. All 78 expected workload administrations were received across thirteen participants and six marked tasks. Before each session, the project environment was reset to the same seeded state.

Task 8 was initially designed as a refusal test in which deployment would be blocked before QA completion. On 30 August 2026 it was redesigned so that tasks 3-9 form one continuous workflow. Deployment succeeds in task 8 because the QA sign-off is completed in task 7. The consequence for the UX questionnaire is discussed in Section 9.3.3.

9.3.3 Apex UX Questionnaire

An eighteen-item questionnaire was developed for this evaluation to capture aspects not addressed by SUS or Raw NASA-TLX those being trust in AI-generated content, perceptions of phase gates, and whether reviewing generated output felt cheaper than producing it. The items were grouped into five constructs: process understanding, interaction with AI-generated content, control and reversibility, feedback and errors, and fit to real work. Because the questionnaire is not validated, results are reported item by item rather than as composite or subscale scores. It was administered after SUS and is reproduced in Chapter B. Two items in the fifth construct concern the framework rather than Apex itself, asking whether the artefacts and process fit real work. Although presented on the same questionnaire for participant convenience, they are analysed separately under the framework evidence in Table 9.1. Two cross-item relationships were defined in advance:

- **Trust and review effort.** Item B4 asks whether participants trusted generated content enough to use it in a real project, while B2 asks whether reviewing it required less effort than writing it. Read together, they provide a trust-calibration indicator. High trust with low review effort may suggest acceptance without sufficient scrutiny, whereas high trust with substantial review effort is more consistent with confidence formed through checking. This pair provides the closest available proxy, in an unmoderated self-report design, for whether generated output was critically reviewed rather than accepted wholesale.
- **Gate comprehension.** Item A4, *when the tool refused to let me continue, I understood why*, was

originally intended to be interpreted alongside task 8 NASA-TLX Frustration and free-text responses. When task 8 was redesigned from a deliberate refusal test into a successful deployment flow (Section 9.3.2), A4 was removed from both language versions on 30 August 2026. Seventeen of the original eighteen items were therefore administered. A4 represents no missing data, since it was absent for all participants; however, its removal eliminated the study's only direct measure of whether a phase gate was perceived as a guardrail or an obstruction, a limitation carried into Section 9.5.

9.3.4 Semi-Structured Interviews

The interviews used a population separate from the unmoderated task-based study in Section 9.2. Practitioners and an Agile/Scrum expert were recruited independently because the interview guides in Chapter B address framework practicality, adoption cost and alignment with established delivery methods rather than task-level usability. The Agile-expert guide therefore does not require prior use of Apex. Themes were derived from the contemporaneous notes described in Section 9.2. An initial codebook was developed from the first interview in each group and extended as additional themes emerged. Session duration was not recorded systematically. Interviewees are identified only by code: T1-T5 for practitioners and E1 for the Agile/Scrum expert. These codes are intentionally separate from the P-codes used in the unmoderated study because the two populations are disjoint.

Five practitioners and one Agile expert had been consulted at the time of writing. This is at or below the lower bound of the intended recruitment targets, so the results should be read as a small set of attributable, information-rich interviews rather than a saturated qualitative sample. T3 was consulted twice, the second time after substantial framework revision. T1 and T5 took part only in the informal formative consultations reported in Section 7.5 and are not analysed here; the evidence in this section therefore comes from the semi-structured interviews with T2, T3, T4 and E1.

A - Practitioners. Three themes emerged:

- **Specification as durable project memory.** T2 endorsed the idea of the locked specification as a persistent source of project knowledge, combining guidelines, requirements, plans and tests. The same practitioner also warned that implicit assumptions are difficult to make fully explicit.
- **Versioning and non-linearity.** T4 argued that specification history should remain inspectable and editable through semantic versioning, and independently raised the risk that a strictly linear phase sequence could reproduce waterfall-style rigidity. The latter concern is notable because it was also anticipated in the Agile-expert interview guide, giving it support from two independent directions.
- **Interface and framing issues.** T3's first interview raised two narrower concerns: richer project-management context in AI-generated requirements and clearer on-screen state during UI review. T3's second interview produced the strongest challenge to the framework and is reported in Section 7.5, where its traceable design consequences are discussed.

B - Agile/Scrum expert. The expert interview followed a live framework presentation and Apex demonstration. The strongest positive reaction concerned the value of executable instantiation. The expert found the framework easier to interpret when roles, phases and gates were demonstrated in

a working tool rather than described only in prose. The flexible use of hats rather than fixed named roles, Mob Elaboration, and the governance metrics were also identified as points of difference from more rigid organisational methodologies. Two limitations were also raised. First, the expert experienced first-contact navigation confusion, consistent with the orientation difficulties identified elsewhere in the evaluation (Section 9.6). Second, the expert described an organisational setting in which AI use is centrally restricted and metered, but with no dedicated governance role. This prompted a broader question about whether AI participation produces a net cost or benefit. The framework's metrics in Section 6.7.4 address governance effects, but do not yet operationalise token or usage cost specifically; this is carried forward in Chapter 10.

C - Read across both groups. The strongest convergent finding is the concern over excessive linearity. It was raised unprompted by a practitioner and independently anticipated in the expert guide. The strongest divergence concerns the specification-as-source-of-truth idea. T2 endorsed it, while the sharpest criticism of the same construct is reported separately in Section 7.5. The evidence is therefore mixed rather than uniformly favourable.

9.3.5 Analytical Assessment

The framework is additionally assessed against five criteria fixed before the results were seen rather than derived from them. *Clarity* asks whether each construct is defined precisely enough to be applied without its author present. *Completeness* asks whether the framework covers the lifecycle it claims to cover, including its own defined measurements. *Alignment with SDLC phases* asks whether the phase decomposition maps onto recognised lifecycle stages without merging, splitting or omitting one. *Support for human-in-the-loop practice* asks whether the framework makes human review of generated output structurally necessary rather than merely advisable. *Governance of AI-enabled activity* asks whether AI contributions are made traceable, attributable and measurable by mechanism rather than by discipline.

Each is rated on a three-level ordinal scale. *Met* means the criterion is satisfied and evidence exists that it is. *Partially met* means it is satisfied in part and the shortfall is identifiable rather than merely suspected. *Not met* means it is not satisfied. No finer scale is used, because the evidence available cannot distinguish more than three levels.

Each rating is justified against evidence that exists independently of the author's opinion of his own design: the construct-to-mechanism mapping in Table 7.1, whose negative rows are as usable here as its positive ones; the interviews in Section 9.3.4; and incidents recorded during the study sessions themselves. Where the only available support for a rating would be the author's own assessment, the rating is lowered rather than argued, on the principle that a criterion which cannot be evidenced is not distinguishable from one that is not met. The ratings are reported in Table 9.4.

9.4 Results

9.4.1 System Usability Scale

Thirteen participants returned complete SUS responses, with all individual scores reported in Table B.7 (Chapter B). Given the small sample, the distribution is reported alongside the mean rather than relying on a single aggregate.

The overall mean was **56.73** (SD 18.50, median 60, range 15-80), below the published corpus mean of approximately 68. On the curved grading scale this corresponds to **grade D**, in the **15th-34th percentile band**; on the adjective scale it lies between *OK* at **50.9** and *Good* at **71.4**, closer to the former. The language groups are reported separately. The ten English responses averaged 60.50 (SD 19.18, median 63.75), also grade D, whereas the three Portuguese responses averaged 44.17 (SD 9.46, median 40), corresponding to **grade F** in the **0th-14th percentile band**. The Portuguese subgroup is too small to support broader claims and is interpreted cautiously (Sections 9.3.1 and 9.5).

The distribution is uneven. Six participants scored above 60, two exactly 60 and five below, with four at or above 70. P6 recorded the lowest score, **15**, below Bangor's *Awful* anchor at **20.3** and close to *Worst imaginable* at **12.5**. This value is retained, but interpreted alongside the participant's Raw NASA-TLX responses, where 35 of 36 ratings were identical, suggesting a constant-response pattern. With thirteen participants there is neither sufficient justification to exclude the case nor enough statistical capacity to treat it as negligible.

The subscales show a clearer pattern. Usability averaged **59.86**, while Learnability averaged **44.23**. Learnability also showed much greater dispersion, ranging from **0 to 100**: P2 scored **0**, maximally agreeing that technical support and substantial prior learning would be needed, while P11 scored **100**, disagreeing with both as strongly as the scale permits. Usability ranged from **12.50 to 87.50**. This suggests that the stronger difficulty lies in initial learning and orientation rather than in continued use once the system is understood, a point taken up in Section 9.6.

9.4.2 Raw NASA Task Load Index

Table 9.2 reports the six Raw NASA-TLX subscales across the six marked tasks, with each cell representing the mean over thirteen participants. The subscale profile is treated as the primary result because the dimensions vary independently and would be obscured by the aggregate alone.

Table 9.2: Raw NASA-TLX subscale means by task, over thirteen participants, on the instrument's 0 to 100 range. Performance carries reversed anchors and is reported as marked, so a higher value is a self-assessment further from *Perfect*. Every response is a multiple of ten, so differences of a few points between cells are not interpreted.

Task	Mental	Physical	Temporal	Effort	Frustr.	Perf.	Aggregate
3 Gherkin	26.9	17.7	21.5	23.8	13.1	14.6	19.6
4 Design	47.7	21.5	40.8	36.2	20.0	16.2	30.4
6 Packs	40.0	21.5	32.3	36.2	26.2	18.5	29.1

Continued on next page

Task	Mental	Physical	Temporal	Effort	Frustr.	Perf.	Aggregate
7 QA	25.4	13.8	19.2	23.1	15.4	10.8	17.9
8 Deploy	25.4	20.8	26.9	29.2	22.3	21.5	24.4
9 Export	9.2	7.7	6.9	7.7	3.1	10.8	7.6
All tasks	**29.1**	**17.2**	**24.6**	**26.0**	**16.7**	**15.4**	**21.5**

Across all 78 administrations, the grand aggregate was **21.5**/**100**. Mental demand was the largest component at **29.1**, followed by effort at **26.0** and temporal demand at **24.6**; frustration was lower at **16.7**, below physical demand at **17.2**. Performance averaged **15.4** on the reversed scale, placing perceived success close to the *Perfect* end. The overall profile therefore suggests that workload was concentrated more in comprehension and review than in frustration or physical effort.

By task, **Design** was heaviest overall at **30.4**, followed by **Packs** at **29.1**; both involve substantial generated material for review. **Export** was lightest at **7.6** and lowest on every subscale. This pattern is consistent with workload tracking the amount of generated material to assess rather than simply the number of interface actions.

The pre-administration predictions were only partly supported. Task 3 had been expected to produce the highest mental demand, but scored **26.9**, below Design at **47.7**. Task 6 had been expected to be moderate, yet ranked second overall, tied for the highest effort at **36.2**, and produced the highest frustration at **26.2**. Two predictions held: Task 9 was lowest overall and on every subscale, and physical demand remained low, never exceeding **21.5**. Task 8 ranked third overall at **24.4** and recorded the highest Performance value at **21.5**, meaning participants judged their deployment performance furthest from *Perfect*; Section 9.6 compares this with an independently recorded defect in the same feature.

9.4.3 Apex UX Questionnaire

Table B.5 reports all seventeen administered items individually using response counts and medians, with no composite or construct scores. The complete matrix appears in Chapter B; Table 9.3 reproduces the nine items discussed here. Responses marked *N/A - did not use this* are reported separately and excluded from medians.

Table 9.3: Selected Apex UX questionnaire results ($N = 13$). Scale 1-5 from strongly disagree to strongly agree; *N/A* responses are excluded from medians. E2-E3 assess the framework; remaining items are reported in Chapter B.

	Item	1	2	3	4	5	N/A	Mdn
B. Working with AI-generated content								
B1	Could tell generated content from my own	0	2	0	3	7	1	5
B2	Reviewing took less effort than writing it	0	0	2	6	5	0	4
B3	Felt able to reject or change the output	0	0	0	1	11	1	5
B4	Trusted it enough to use in a real project	0	0	3	8	2	0	4

Continued on next page

	Item	1	2	3	4	5	N/A	Mdn
D. Feedback and errors								
D1	Was told clearly what had happened when something went wrong	0	1	6	4	2	0	3
D2	Messages said what to do next, not only what failed	0	0	4	3	6	0	4
E. Fit to real work								
E1	The artefacts are ones my team would use	0	1	3	5	1	3	4
E2	*(framework)* The process fits how my team works	0	2	2	6	0	3	4
E3	*(framework)* The effort is worth what it produces	0	0	2	5	6	0	4

Of the fifteen instantiation-scoped items, **ten had a median of 4** and **four a median of 5**. The median-5 items were B1, distinguishing generated content from one's own; B3, ability to reject or change output; C2, confidence that a step could be undone; and C4, acceptability of generation wait time. The weakest closed item was **D1**, with a median of **3**: six participants selected the neutral point and one responded below it. D2, on whether messages explained what to do next, had a median of **4**. Together, these responses suggest that guidance on next actions was stronger than explanations of what had actually gone wrong.

The pre-defined trust-calibration pair did not diverge at the median. B2, reviewing requiring less effort than writing, and B4, trust sufficient for real use, both had a median of **4**, with no participant below **3** on either item. This is consistent with calibrated trust at the aggregate level, but P7's free text described the opposite pattern directly: "I did not review in great detail and I trusted it completely". Because the sessions were unobserved, the study cannot determine how common that behaviour was; this limitation is carried into Section 9.5.

The framework-scoped items show a more nuanced result. E3, whether the process effort is worth its output, received no disagreement. **0** responses at 1 or 2, **2** neutral, **5** at 4 and **6** at 5, giving a median of **4**. E2, whether the process fits the participant's team, was the weaker-supported of the two framework items. **3** marked *N/A*, none strongly agreed, **2** disagreed, **2** were neutral and **6** agreed, also yielding a median of **4**. The same three participants marked *N/A* on E1, so those missing values are related rather than independent; all three answered E3. The main distinction is therefore between perceived value and organisational fit because participants were more willing to say the process was worth its cost than that it matched how their own team works.

9.4.4 Analytical Assessment

Table 9.4 gives the rating for each of the five criteria stated in Section 9.3.5, with the evidence each rests on. No criterion is rated *not met*, and that is not a favourable finding in itself. These are the framework's own design criteria, so a *not met* would record a failure at something it explicitly set out to do. The information is in the three *partially met* rows, each of which names a specific shortfall rather than a general reservation.

Table 9.4: Analytical assessment of the framework against the five criteria fixed in Section 9.3.5, with the evidence supporting each rating.

Criterion	Rating	Justification and evidence
Clarity	Partially met	All constructs resolved to concrete mechanisms in Table 7.1, and T2 independently described the central specification construct as a shared repository for requirements, plans and tests (Section 9.3.4). However, spec-drift detection exposed an unstated assumption that specifications are scoped per story (Section 7.7), so some preconditions remain insufficiently explicit.
Completeness	Partially met	Strategic Alignment, all six lifecycle phases and three loop-back routes are covered (Table 6.4, Figure 6.1). However, AI Defect Escape Rate is only operationalised as a proxy, and spec-drift detection is complete for the functional but not technical specification (Table 7.1). The shortfall is therefore in construct definition rather than lifecycle coverage.
Alignment with SDLC phases	Met	Each phase maps to a recognised lifecycle stage with a defined AI contribution, accountable hat and exit control (Table 6.4), and instantiation required no phase to be merged, split or skipped (Table 7.1). T4 questioned linearity rather than the phase set itself (Section 9.3.4).
Support for human-in-the-loop practice	Met	Every phase requires both a specification-store write and human acceptance, so generation alone cannot satisfy a gate (Figures 7.4 and 7.5). The Interactive Bridge, Consistency Factor and server-side reconciliation are realised (Table 7.1). Participants also reported strong ability to distinguish and reject generated output, although P7 shows that the framework cannot detect superficial review.
Governance of AI-enabled activity	Partially met	Gate enforcement, traceability and Context Traceability Rate are machine-checkable rather than purely procedural (Table 7.1). However, AI-attributable defects remain observable only through a proxy (Section 7.7), and the Phase 5 stale-context incident showed that a governance surface can report confidently from unsupported evidence (Section 9.5). Independent validation is therefore still required.

9.4.5 Demonstration in a Real Project

Table 9.1 names the demonstration as an instrument for realisation and practitioner support alongside the four above; Chapter 8 narrates it in full and Section 8.5 already states its headline facts (nine epics, fifty-four stories, all three loop-back routes exercised, one regression caught and repaired). What follows is the governance-metric detail behind those facts and how it bears on Table 9.4.

The instantiation's own governance exports return the three metrics Section 6.7.4 defines, and one it does not. Bolt Cycle Time, computed per task from `pack_ready` to `done`, is median 0.22h and p90 1.4h across 254 tasks. Context Traceability Rate is 50 of 54 deployed stories, 92.6 per cent (the shortfall's single root cause is given in Section 8.4). Spec Conformance Rate, an instantiation-added metric rather than a framework one, is a mean of 98.65 per cent across all 54 checked stories, with the finding-level detail and its shared-component root cause in Section 8.4. AI Defect Escape Rate is 1 of 54, 1.85 per cent, subject to the same-session blind spot already noted there. Nineteen Fix-Bolts were logged across 14 of the 54 stories.

Two results from the demonstration bear on Table 9.4's ratings above and are cited there rather than repeated: the Testing Gate was shown passable by a suite that had stopped measuring the system it

tested, and a governance surface, the Phase 5 pre-flight check, was found reporting high confidence on stale evidence on two further occasions beyond the one already recorded in the study proper (Section 9.6). A third result has no instrument above to attach to: on one occasion the AI implementer executed a destructive command against the live production database without checking which environment it was pointed at, a failure inside a phase rather than at a gate, which no construct in Table 7.1 was positioned to catch (Section 8.4).

9.5 Threats to Validity

A - Construct validity. Every instrument here measures perception, and none measures delivered software quality. SUS and Raw NASA-TLX are standardised and comparable against published corpora, which is the mitigation that does apply; the UX questionnaire's construct-validity limit is that it is author-written and unvalidated, addressed by treating it as such (Section 9.3.3). Two departures already described (Section 9.3.2) reduce what the workload instrument can resolve: the 0 to 10 response scale, and the anchoring and fatigue risk of six repeated administrations. One respondent's data is consistent with exactly that. Thirty-five of P6's thirty-six workload ratings are the same number. No mitigation was available for either.

The sharpest construct gap concerns phase gates. The item A4/task-8 cross-read described in Section 9.3.3, this study's only planned evidence on whether a gate reads as a guardrail or as obstruction, did not survive task 8's redesign away from a refusal test (Section 9.3.2). The gap is therefore worse than the absence of an observer check; there was no guaranteed refusal trigger for any of the thirteen participants, so gate comprehension is unmeasured rather than weakly measured, the framework's most distinctive mechanism the one this evaluation has least evidence about.

B - Internal validity. The author designed both artefacts, wrote the task script and the seventeen UX items, built the seeded demonstration project, and conducted the interviews. The unmoderated design removes that conflict from the sessions structurally (Section 9.3), the strongest mitigation available, but removes nothing from the instruments themselves. Two incidents make the residue concrete rather than hypothetical. A configuration defect let an interface-language setting leak between projects, so a nominally English session was answered by the model in Portuguese, and it was noticed only because the author recognised the symptom in a participant's free text and root-caused it himself. A second defect, in which Phase 5 read a stale repository-context file as current and reported at high confidence, was found during the first session and was still present for every subsequent participant. In both cases the person best placed to detect and interpret the anomaly was the artefact's designer.

The same window produced a second internal threat. The system was not frozen. One defect fix was deployed during data collection and one configuration change followed it, so participants did not all use a byte-identical build. Against that, the project environment was reset to an identical seeded state before every session (Section 9.3.2).

C - External validity. The sample (Section 9.2) is a convenience sample from one person's professional network, skewed junior, with no open call and no recorded count of those approached. It supports

a description of how these thirteen people experienced the tool and no claim about experienced practitioners, teams or organisations. The pre-registered concern about a single organisational setting turns out to apply in a different place than expected. The participants are not from one organisation, but the *task* environment is a single seeded project on one project-management tool, which ten of the thirteen had never used, so unfamiliarity with that board is confounded with unfamiliarity with Apex throughout. The observation window is one session of roughly an hour, which precludes any claim about long-term maintainability, adoption or organisational change; one participant makes the point himself, expecting that "with regular use, I believe the user would become very familiar with the interface" (P12), which is a hypothesis this design cannot test. The Portuguese figures rest on three respondents administered an instrument whose own validation reports weak inter-rater agreement, and are reported as indicative only.

D - Conclusion validity. Thirteen participants spread over five levels of prior AI exposure cannot support inferential statistics or subgroup comparison, and none is computed. Two administration artefacts show what an unmoderated design cannot catch as it happens: one participant declared Portuguese as the session language and then completed every instrument in English, and another declared English but submitted the Portuguese consent form. Neither was correctable at the time because nobody was present to correct it.

E - The cost of recording nothing. Two threats follow from the decision not to observe or record, and both are consequences of a deliberate design rather than oversights. First, there is no observational evidence of any kind: no measured completion rate, no time on task beyond a form-submission timestamp that cannot distinguish task difficulty from an interruption, no assist or error counts, and no independent check on comprehension anywhere. This leaves the perception-versus-quality threat standing rather than partly answered, and it is why the trust-calibration pair in Section 9.4.3 can report that trust and review effort did not diverge without being able to say whether any individual accepted output unread. Second, the interview analysis rests on contemporaneous written notes rather than transcripts (Section 9.2), and the selection was made by the person who designed the artefact under discussion; Section 9.3.4 names which interviewee raised which point precisely so that the selection is at least attributable.

9.6 Discussion

The instruments do not converge on a single narrative; the most informative findings lie in where they agree for different reasons and where they diverge.

The first apparent tension is between SUS and the UX questionnaire. The overall SUS mean was **56.73** (grade D), while several mechanism-level UX items reached medians of **5**, including the ability to reject generated output, undo a step, and tolerate generation time. These results concern different levels. SUS evaluates the system as a whole, whereas the UX items evaluate individual mechanisms. Free-text responses reinforce this distinction: **9 of 13** participants identified density, orientation or navigation as the main source of confusion, rather than the AI mechanisms themselves. The SUS subscales point in the same direction, with Learnability at **44.23** compared with Usability at **59.86**. The combined

evidence therefore suggests that the mechanisms are individually understandable, while the interface that assembles them imposes substantial first-contact cost. This is a finding about Apex rather than the framework itself (Section 9.1).

A second cross-read concerns error visibility. D1, on understanding what had happened when something went wrong, was the weakest closed UX item. Independently, Design and Packs were the two heaviest workload tasks, and Packs produced the highest frustration score at **26.2** despite being predicted to impose only moderate load (Section 9.4.2). P8's free text connects the two findings directly. An error was insufficiently surfaced, allowing subsequent stages to be completed on an incorrect assumption. The weakness in D1 therefore reflects more than message wording; poor error legibility can allow a formally successful workflow to continue from a faulty state.

The clearest convergence occurs around deployment. Task 8 recorded the highest Performance value of any task at **21.5**, meaning participants rated their deployment success furthest from *Perfect*. Four participants also commented on Phase 5 pre-flight reasoning, while an independently recorded defect showed that the same mechanism read stale repository context as current and reported its verdict with high confidence (Section 9.5). Questionnaire data, free text and an observed implementation defect therefore converge on the same governance mechanism, supporting the *partially met* rating in Table 9.4.

Trust remains unresolved. B2, on review effort, and B4, on trust sufficient for real use, both had a median of **4**, with no aggregate divergence. However, one participant explicitly reported trusting the generated output completely and not reviewing it in detail. Because the sessions were unobserved, the study cannot determine whether this behaviour was exceptional or common. The instrument therefore found no evidence of systematic trust miscalibration, but the design was not capable of ruling it out.

Finally, the tool and the framework received different forms of evaluation. Apex scored below average on usability, while practitioner and expert interviews were broadly positive about the framework, including its specification-centred approach and flexible hat structure (Section 9.3.4). This difference is not contradictory: the usability instruments evaluate the implementation, whereas the interviews address the methodology. As established in Section 9.1, weaknesses in the interface do not by themselves constitute evidence against the framework.

9.6.1 Addressing the Research Problem

The evidence above is read here against the four dimensions of the problem stated in Chapter 5.

A - Placement. The framework bounds AI participation at phase granularity through Table 6.4: each phase defines the permitted AI contribution, required output, accountable hat and exit control. The decomposition survived instantiation without any phase being merged, split or omitted, and the alignment criterion is rated *met* in Table 9.4. An independent expert also recognised the phase and hat structure as distinct from more rigid organisational methodologies (Section 9.3.4). The qualification is clarity, since phase boundaries are explicit, but at least one construct depended on an unstated precondition that only implementation exposed.

B - Governance. The strongest governance mechanisms are specification-based gates, stable identifiers and server-side reconciliation of generated claims (Sections 6.7, 6.7.3 and 7.6). These make

artefacts traceable to requirements and prevent generation alone from satisfying a phase. Human-in-the-loop support is therefore rated *met* structurally, and participants independently reported that generated output could be distinguished, rejected and revised (Table 9.4, Table B.5). Accountability remains weaker and Governance of AI-enabled activity is therefore rated *partially met*: AI-attributable defects can only be observed through a proxy, and one governance surface was found reporting confidently from stale evidence. The framework therefore makes AI contributions traceable and reviewable, but not yet fully measurable in terms of downstream consequences.

C - Realisation. Of the twelve constructs mapped in Table 7.1, **ten are realised as specified**, **one is realised only as a proxy**, and **one is partially realised**. The exceptions are analytically important. Bolt Cycle Time was initially implemented using the wrong proxy and corrected only by comparison with its written definition (Section 7.3); spec-drift detection exposed an unstated assumption that specifications are scoped per story rather than per project.

The demonstration shows that operationalisation also survives real use. Across the deployed project, Context Traceability resolved for **50 of 54 stories**, and one shared-component conformance defect was detected consistently across **nine stories** (Section 9.4.5). However, a realised construct is not necessarily a reliable one. The Testing Gate was shown passable by a suite that no longer measured the current system, and stale-context governance failures recurred. Operationalisation is therefore substantial but incomplete in reliability.

D - Practitioner support. The evidence is mixed but internally consistent. Cognitive workload was comparatively low. The grand Raw NASA-TLX aggregate was **21.5**/**100**, frustration averaged **16.7**, and the heaviest task scored **30.4**. Workload was concentrated in mental demand and in phases requiring substantial review of generated material.

Usability was weaker. The overall SUS mean was **56.73**, grade D, with one participant scoring **15**; the Portuguese subgroup averaged **44.17**, grade F. Error legibility was the weakest UX area, and **9 of 13** participants identified density, orientation or navigation as the main source of confusion. The combined interpretation is that Apex imposes relatively modest workload once users understand the workflow, but does not support initial orientation well enough. These are findings about the instantiation, not the framework itself, and remain bounded by the one-session evaluation and sample limitations in Section 9.5. The nine-day demonstration provides complementary evidence of sustained use, but also exposed the same governance surfaces as vulnerable to stale or misleading evidence (Section 9.4.5).

For future work, usability and visual simplicity should be treated as non-negotiable design requirements. Because Apex is the main interface through which practitioners encounter the framework, visual complexity or poor orientation can be mistaken for methodological difficulty, especially by first-time users. Improving first-contact clarity, navigation and visual hierarchy is therefore necessary not only for the interface, but for practical adoption of the framework itself.

10

Conclusion

Contents

10.1 Summary and Contributions 69
10.2 Communication 71
10.3 Limitations 71
10.4 Future Work 72
Declaration of Generative AI and AI-Assisted Technologies in the Writing Process 73

10.1 Summary and Contributions

This dissertation addressed a methodological rather than purely technical problem. AI is increasingly present throughout the SDLC, yet its benefits depend on the process surrounding its use. The evidence synthesised in Chapters 1 and 4 shows that functional correctness does not imply security, perceived and measured productivity can diverge, and increased output does not necessarily improve delivery outcomes. Existing software-development methodologies were designed around human contributors, while emerging AI-oriented approaches remain fragmented in their treatment of lifecycle integration, oversight and evaluation. The resulting research problem is the absence of structured and repeatable guidance for integrating AI across the SDLC while preserving human oversight, quality, security and accountability.

Two artefacts were produced. The first is the framework in Chapter 6: Strategic Alignment followed by six lifecycle phases, each defining the permitted AI contribution, required output, accountable hat

and exit control (Table 6.4). Its governance mechanisms include Bolts, the Consistency Factor, Spec-Anchored Continuity, five quality gates, three governance metrics and three explicit Maintenance loop-backs. Responsibilities are expressed as functions rather than fixed organisational roles. The second artefact is Apex (Chapter 7), a working instantiation used to demonstrate that these constructs can be implemented and evaluated.

The evaluation deliberately separated evidence about the framework from evidence about Apex. The demonstration carried a complete product over nine calendar days, growing from six planned epics to nine delivered, fifty-four stories carried through the lifecycle to a deployed increment, while exercising all three loop-back routes (Section 9.4.5). The framework was also assessed against five predefined analytical criteria, of which two were *met* and three *partially met*, and discussed with five practitioners and one Agile/Scrum expert. Apex itself was evaluated through an unmoderated nine-task study with thirteen participants, producing a SUS mean of **56.73** and a Raw NASA-TLX aggregate of **21.5**/**100**. A usability result for the tool is therefore not treated as evidence for or against the framework itself. Against the four dimensions of the research problem, the outcome is as follows:

- **Placement.** AI-supported activity can be bounded at phase granularity by defining, for every phase, its AI contribution, output, accountable hat and exit control. The decomposition survived instantiation without any phase being merged, split or omitted and was independently recognised in expert interview. The qualification is that some lower-level construct assumptions remain insufficiently explicit.

- **Governance.** Specification-based gates, stable identifiers, Spec-Anchored Continuity and server-side validation make AI contributions traceable and reviewable. Human-in-the-loop support is structurally realised because generation alone cannot satisfy a phase gate. Accountability remains weaker, because AI-attributable defects are observable only through a proxy, and governance recommendations were observed operating on stale evidence.

- **Realisation.** Of twelve mapped constructs, **ten are realised as specified**, **one only as a proxy**, and **one partially**. Implementation also exposed framework assumptions. Bolt Cycle Time was initially measured through the wrong proxy, while spec-drift detection revealed an unstated requirement for story-level specification ownership. Realisation does not imply reliability, however. The Testing Gate could be satisfied by tests that no longer exercised the current system, and no existing gate covers a destructive action issued inside a phase.

- **Practitioner support.** Cognitive workload was comparatively low, but usability was not. The overall SUS score was grade D, error legibility was the weakest UX area, and **9 of 13** participants identified density, orientation or navigation as their principal difficulty. Apex therefore imposes relatively modest workload once understood, but insufficiently supports first contact.

The dissertation's contributions are consequently fourfold: the systematic synthesis of evidence on AI participation across the SDLC; the proposed framework of phases, gates, responsibilities, metrics and specification anchoring; Apex as an executable instantiation exposing both successful and resistant constructs; and an empirical evaluation that separates claims about the methodology from claims about its implementation. The work does *not* establish that the framework improves delivered software quality, since no instrument in this study directly measures that outcome.

10.2 Communication

This section completes the Communication activity of the DSRM through three dissemination vehicles:

- **Public repository.** The complete Apex implementation and its documentation, including every architectural, design and governance record referenced throughout this dissertation, are published in full at `https://github.com/thomastabs/apex`, with the deployed instantiation itself available at `https://apex-bolt.com`. Outfolio, the separate product demonstrating the framework end to end (Chapter 8), is likewise published at `https://github.com/thomastabs/OutfolioV1`, with the deployed product itself available at `https://outfolio-v1.vercel.app/`.
- **Extended abstract.** A self-contained technical article accompanies the dissertation, summarising the framework, its instantiation and evaluation for readers who will not consult the full document.
- **Dissertation and defence.** This document and its academic defence communicate the complete work and expose its design decisions to examination by reviewers who did not participate in its construction or evaluation.

10.3 Limitations

These limitations concern the artefacts and research as a whole, rather than the evaluation-specific threats discussed in Section 9.5.

- **The framework is an original synthesis.** Its individual mechanisms are grounded in prior work, including AI-DLC [41], Team Topologies and the Spotify model [43, 46], and specification-driven development [27, 28], but no source provides the combined blueprint proposed here. Implementation exposed several unresolved assumptions: AI Defect Escape Rate is only observable through a weaker linked-issue proxy; technical-specification drift lacks story-level ownership; destructive operations inside a phase fall outside the gate model; and advisory governance can report high confidence from stale evidence. Whether an effective in-phase control can be added without undermining the phase-gate model remains open.
- **Some grounding evidence is recent or non-peer-reviewed.** AI-DLC's productivity claims are vendor-reported [41]; specification-driven development remains an emerging practice [27]; one key dependability argument is a preprint [28]; and several motivating results come from industry telemetry and surveys [6, 7, 9, 12]. The argument therefore rests on convergence across independent sources rather than on any single reported effect size.
- **Governance-standard terminology does not imply compliance.** The framework is positioned against the functions and vocabulary of the NIST AI Risk Management Framework and ISO/IEC 42001:2023 [48, 49], but neither the framework nor Apex has been audited or certified against either standard.
- **The evidence concerns early adoption in a narrow setting.** The end-to-end demonstration covers nine days of solo use by the artefacts' designer; the usability study covers one approximately hour-long session per participant, one seeded project and one project-management tool; and the

planned independent organisational demonstration did not run. The study therefore cannot establish sustained gate discipline, long-term specification integrity, organisational adoption, metric behaviour under management incentives, or transferability to teams that did not design the framework.

10.4 Future Work

Future work follows directly from the limitations and empirical findings above:

- **Evaluate the framework across organisations and team sizes.** The only complete lifecycle demonstration used one practitioner holding every hat, while the thirteen evaluation participants followed a fixed seeded task rather than applying the framework to their own work. The weakest-supported framework UX item concerned fit to real teams. Future studies should therefore distribute hats across independent practitioners and organisations that did not design the framework.
- **Evaluate alternative project-management backends.** Additional adapters demonstrate architectural portability but not behavioural parity (Section 7.2). Both the demonstration and evaluation used the first-class backend. Running the framework against another supported tool would test tool-independence empirically and help separate unfamiliarity with Apex from unfamiliarity with the project-management system.
- **Replace proxy metrics with direct measurement.** AI Defect Escape Rate currently observes only defects later linked to a deployed story; the demonstration recorded **19 Fix-Bolts across 14 stories**, while the escape proxy recorded only **one**. Direct production telemetry is therefore required, together with an explicit framework definition of who owns attribution of defects to generated logic.
- **Extend governance to AI cost.** The framework should add an **AI Token and Compute Cost** metric at Bolt granularity, recorded directly from provider usage data. This should be paired with a new hat, beyond those in Table 6.3, accountable for AI cost, detection of unstaffed responsibilities, and risk- or phase-tiered model selection extending the existing Fast Lane/Secure Lane principle.
- **Improve support for the Design phase.** Task 4 produced the highest mental demand (**47.7**), temporal demand (**40.8**) and aggregate workload (**30.4**), while the next-highest mental-demand task scored 26.9. Rather than adding a heavier gate, future versions should provide worked examples and structured interim critique of usability, scenario coherence and architectural fit before whole-artefact review.
- **Treat first-contact usability as a non-negotiable requirement.** SUS Usability averaged **59.86**, compared with Learnability at **44.23**, with Learnability ranging from **0-100** against Usability's **12.50-87.50**. Error legibility was the weakest UX item, mental demand was the largest workload subscale, and the Portuguese subgroup averaged **44.17** compared with **60.50** for English respondents, although $n = 3$ prevents broader inference. Because practitioners encounter the methodology through Apex, poor orientation or visual complexity can be experienced as methodological difficulty. Future redesign should therefore prioritise visual simplicity, first-contact orientation, navigation and error legibility as prerequisites for practical framework adoption.

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 Chapter 6 and Chapter 7.

Bibliography

[1] J. H. Moore and N. Tatonetti, “Vibe coding: a new paradigm for biomedical software development,” *BioData Mining*, vol. 18, no. 1, pp. 1–3, 2025.

[2] A. Zamfiroiu, C.-B. Păştinică, C. Crăciun, C. Ciutacu, and T. Luca, “Exploring Conversational Programming through the CHOP Paradigm and Artificial Intelligence,” *Informatica Economica*, vol. 29, no. 2, pp. 5–17, 2025.

[3] I. Ahmed, A. Aleti, H. Cai, A. Chatzigeorgiou, P. He, X. Hu, M. Pezzè, D. Poshyvanyk, and X. Xia, “Artificial Intelligence for Software Engineering: The Journey So Far and the Road Ahead,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 5, pp. 1–27, 2025.

[4] S. Zhang, Z. Xing, R. Guo, F. Xu, L. Chen, Z. Zhang, X. Zhang, Z. Feng, and Z. Zhuang, “Empowering Agile-Based Generative Software Development through Human-AI Teamwork,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 6, pp. 1–46, 2025.

[5] W. Hou and Z. Ji, “Comparing Large Language Models and Human Programmers for Generating Programming Code,” *Advanced Science*, vol. 12, no. 8, pp. 1–12, 2025.

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

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

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

[9] Veracode, “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, July 2025. [Online]. Available: https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf

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

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

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

[13] L. Banh, F. Holldack, and G. Strobel, “Copiloting the future: How generative AI transforms Software Engineering,” *Information and Software Technology*, vol. 183, 2025.

[14] M. Farhanna, N. Paramitha, M. Susi, H. Surya, A. Yanti, P. Dea, and S. Deni, “Exploring the Use of Generative AI in Software Development: A Preliminary Study,” in *E3S Web of Conferences*, vol. 645, 2025, p. 04002.

[15] A. Hemmat, M. Sharbaf, S. Kolahdouz-Rahimi, K. Lano, and S. Y. Tehrani, “Research directions for using LLM in software requirement engineering: a systematic review,” *Frontiers in Computer Science*, vol. 7, 2025.

[16] C. Gao, X. Hu, S. Gao, X. Xia, and Z. Jin, “The Current Challenges of Software Engineering in the Era of Large Language Models,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 5, pp. 1–30, 2025.

[17] H. A. Al-Hashimi, R. A. Khan, H. S. Alwageed, A. M. Algarni, S. Ayouni, and A. O. Almagrabi, “Exploring the role of generative AI in enhancing cybersecurity in software development life cycle,” *Array*, vol. 28, p. 100509, 2025.

[18] M. Basha and G. Rodríguez-Pérez, “Trust, transparency, and adoption in generative AI for software engineering: insights from twitter discourse,” *Information and Software Technology*, 2025.

[19] X. Chen, C. Gao, C. Chen, G. Zhang, and Y. Liu, “An Empirical Study on Challenges for LLM Application Developers,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 7, pp. 1–37, 2025.

[20] U. K. Durrani, M. Akpinar, M. F. Adak, A. T. Kabakus, M. M. Ozturk, and M. Saleh, “A Decade of Progress: A Systematic Literature Review on the Integration of AI in Software Engineering Phases and Activities (2013-2023),” *IEEE Access*, vol. 12, pp. 171 185–171 204, 2024.

[21] M. A. Haque, “LLMs: A game-changer for software engineers?” *BenchCouncil Transactions on Benchmarks, Standards & Evaluations*, vol. 5, no. 1, pp. 30–41, 2025.

[22] S. K. Jabrw and Q. I. Sarhan, “A Systematic Survey on Large Language Models for Code Generation,” *ARO-The Scientific Journal of Koya University*, vol. 13, no. 2, 2025.

[23] D.-K. Kim and H. Ming, “Assessing output reliability and similarity of large language models in software development: A comparative case study approach,” *Information and Software Technology*, vol. 185, 2025.

[24] V. Terragni, A. Vella, P. Roop, and K. Blincoe, “The Future of AI-Driven Software Engineering,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 5, pp. 1–20, 2025.

[25] Q. Wang, J. Wang, M. Li, Y. Wang, and Z. Liu, “A Roadmap for Software Testing in Open-Collaborative and AI-Powered Era,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 5, pp. 1–17, 2025.

[26] U. Karlovs-Karlovskis, “Generative Artificial Intelligence Use in Optimising Software Engineering Process: A Systematic Literature Review,” *Applied Computer Systems*, vol. 29, no. 1, pp. 68–77, 2024.

[27] Thoughtworks, “Spec-driven development,” Technology Radar, Vol. 33, November 2025. [Online]. Available: https://www.thoughtworks.com/en-us/radar/techniques/spec-driven-development

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

[29] D.-K. Kim, “Improving Software Development Traceability with Structured Prompting,” *Journal of Computer Information Systems*, pp. 1–21, 2025.

[30] J. Li, G. Li, Y. Li, and Z. Jin, “Structured Chain-of-Thought Prompting for Code Generation,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 2, pp. 1–23, 2025.

[31] Y. Dong, X. Jiang, Z. Jin, and G. Li, “Self-Collaboration Code Generation via ChatGPT,” *ACM Transactions on Software Engineering & Methodology*, vol. 33, no. 7, pp. 1–38, 2024.

[32] V. V. Jensen, A. Alami, A. R. Bruun, and J. S. Persson, “Managing expectations towards AI tools for software development: a multiple-case study,” *Information Systems & e-Business Management*, pp. 1–33, 2025.

[33] K. Qiu, N. Puccinelli, M. Ciniselli, and L. Di Grazia, “From Today’s Code to Tomorrow’s Symphony: The AI Transformation of Developer’s Routine by 2030,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 5, pp. 1–17, 2025.

[34] A. Tkachuk, “Determining the Capabilities of Generative Artificial Intelligence Tools to Increase the Efficiency of Refactoring Process,” *Technology Audit & Production Reserves*, vol. 3, no. 2(83), pp. 6–11, 2025.

[35] X. Bai, S. Huang, C. Wei, and R. Wang, “Collaboration between intelligent agents and large language models: A novel approach for enhancing code generation capability,” *Expert Systems With Applications*, vol. 269, 2025.

[36] T. Ramos, A. Dean, and D. McGregor, “AI-Augmented Software Engineering: Revolutionizing or Challenging Software Quality and Testing?” *Journal of Software: Evolution & Process*, vol. 37, no. 2, pp. 1–7, 2025.

[37] Y. Han and C. Lyu, “Multi-stage guided code generation for Large Language Models,” *Engineering Applications of Artificial Intelligence*, vol. 139, 2025.

[38] S. H. Yoo and H. J. Kim, “Security Analysis of Automated Code Generation: Structural Vulnerabilities in AI-Generated Code,” *Tehnički Glasnik*, vol. 19, no. 4, pp. 560–574, 2025.

[39] M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui, “Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models,” *ACM Transactions on Software Engineering & Methodology*, vol. 34, no. 7, pp. 1–25, 2025.

[40] V. S. Pendyala and N. B. Thakur, “Performance and interpretability analysis of code generation large language models,” *Neurocomputing*, vol. 656, 2025.

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

[42] S. Siddeeq, M. Abbasi, J. Rasku, Z. Zhang, F. Christophe, T. Mikkonen, and P. Abrahamsson, “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, 2026. [Online]. Available: https://arxiv.org/abs/2607.01980

[43] H. Kniberg and A. Ivarsson, “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, November 2012. [Online]. Available: https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify

[44] Leading Complexity, “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, February 2023. [Online]. Available: https://blog.crisp.se/2023/02/06/leadingcomplexity/leading-complexity-series-summary-1-henrik-knibergs-thoughts-on-the-spotify-model

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

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

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

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

[49] ISO/IEC, “Information technology - Artificial intelligence - Management system,” International Organization for Standardization/International Electrotechnical Commission, International Standard ISO/IEC 42001:2023, December 2023.

[50] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, “The SPACE of Developer Productivity: There’s More to it than You Think,” *ACM Queue*, vol. 19, no. 1, pp. 20–48, 2021.

[51] J. R. V. Jan Pries-Heje, R. Baskerville, “Strategies for Design Science Research Evaluation,” *European Conference on Information Systems (ECIS)*, no. 1, pp. 1–13, 2008.

[52] J. Brooke, “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, 1996, pp. 189–194.

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

[54] A. Bangor, P. Kortum, and J. Miller, “Determining what individual SUS scores mean: Adding an adjective rating scale,” *Journal of Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009.

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

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

[57] S. G. Hart and L. E. Staveland, “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, 1988, pp. 139–183.

[58] S. G. Hart, “NASA-Task Load Index (NASA-TLX); 20 years later,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 50, 2006, pp. 904–908.

A

Extended Results for the SLR

This appendix contains the evidence tables supporting the review in Chapter 4. They are placed here to keep the chapter concise while preserving the full material needed to audit and reconstruct the synthesis in accordance with Kitchenham's guidelines.

A.1 Stakeholder opinions reported in the literature

The fourth research question is answered in Chapter 4 from the three tables below, each separating the perspectives of one of the groups named in its caption.

Table A.1: Public / Researchers Opinions on AI and LLMs in Software Development

Opinion Category	Description	#	References
General Trust / Perception	People's perception of systems powered by artificial intelligence as a mixture of "software and sorcery." [3]	6	[2, 3, 18, 35, 36, 40]
Misinformation Risk	"Users highlight its potential to misguide or hallucinate to students." [18]	4	[13, 18, 20, 22]
Bias and Impartiality Concerns	Users express dissatisfaction over perceived "leftist bias and propaganda in ChatGPT." [18]	3	[3, 18, 36]
Output Diversity / Similarity	"57% similarity overall with 43% variability, presenting that each model has its unique characteristics." [23]	2	[4, 23]
Legal, Copyright, and Licensing Issues	"Questions also surround the copyrightability/ownership of generated code [...]" [3]	2	[3, 18]

Continued on next page

Opinion Category	Description	#	References
Evaluation Urgency	Urgent need to "establish an effective, objective, and complete evaluation system" for code-generation LLMs. [3]	2	[3, 40]
Job Automation	"Many users fear that AI will eventually replace human developers [...]" [18]	1	[18]

Table A.2: Developers / Practitioners Opinions on AI and LLMs in Software Development

Opinion Category	Description	#	References
Productivity and Efficiency Gains	"The knowledge and reasoning capabilities of LLMs have the potential to result in significant efficiency gains." [13]	6	[2, 13, 18, 21, 24, 37]
Reliability Issues / Hallucinations	"LLMs sometimes produce highly convincing content that is factually wrong [...] the 'hallucination phenomenon'." [35]	6	[13, 18, 20, 22, 35, 40]
Output Variability and Trust Decline	"I get a different code each time [...] I lose a bit of confidence in the work." [13]	5	[13, 18, 24, 36, 40]
Quality of Generated Code	"Several quality issues: undefined variables, insecure comments, and code requiring significant revision." [22]	4	[4, 22, 24, 32]
Augmentation of Human Capabilities	"LLMs are not merely a product of overhyped marketing; they represent a profound shift in how software is engineered. They should be seen as powerful tools that augment human capabilities rather than replace them." [21]	3	[18, 21, 24]
Learning and Skill Improvement	LLMs "[...] directly support learning processes through interaction and conceptual clarification." [2]	3	[2, 16, 21]
Knowledge Acquisition / Search Replacement	"LLMs present data in a concise and contextually relevant manner." [13]	2	[2, 13]
Controllability	Participants praised its "controllability [...] allowing modification and feature adjustment after generation." [4]	2	[4, 18]
Risk of Over-reliance and Cognitive Decline	"There is almost the danger that you stop thinking for yourself [...] people will tend to become lazy." [13]	2	[13, 18]
Investment of time with prompts	"You spend ages trying to cut the prompt so that it comes out right [...]" [13]	2	[13, 18]

Table A.3: Organizations / Management Opinions on AI and LLMs in Software Development

Opinion Category	Description	#	References
Need for Human Oversight	LLM outputs "still necessitate extensive human review before implementation." [23]	7	[4, 17, 18, 23, 24, 29, 33]
Data Security, Privacy, and Corporate Bans	"Due to concerns regarding data privacy, when considering real-world practice, most software organizations tend to avoid using commercial LLMs and would prefer to adopt open source ones with training or fine-tuning using organization-specific data." [25]	6	[13, 15, 18, 24, 25, 38]

Continued on next page

Opinion Category	Description	#	References
Operational Costs and Resource Constraints	Organizations must "consider the current limitations in terms of computational power or pay close attention to energy consumption," leading them to "tend to fine-tune medium-sized models". [25]	3	[13, 15, 25]
Strategic Goal for Developer Satisfaction	"We have to be able to compete on efficiency and job satisfaction [...] It is difficult to find software developers." [32]	2	[21, 32]
Expectation Management Requirement	"SDOs must carefully manage their expectations towards AI tools." [32]	2	[18, 32]
Strategic Competitiveness	The "early adoption of LLMs in software engineering is crucial to stay competitive in this rapidly evolving landscape." [21]	1	[21]

A.2 Industry sectors adopting AI and LLMs

The evidence behind the fifth research question, answered in Chapter 4.

Table A.4: Industry sectors adopting AI and LLMs

Industry Sector	Description	#	References
Software Engineering, Software Development & Consulting Services	Software Development and Consulting is mentioned a lot, since they are the main sectors where AI and LLMs are being utilized in many different ways, such as AIOps, Cloud Management, Requirements Engineering, Software Testing, Quality Assurance, Code Generation and Development, Cybersecurity, Vulnerability Management, Refactoring, Code Optimization, Specialized Hardware, Scientific Computing and Emerging Technologies (like Blockchain/Crypto).	25	[2–4, 13–26, 32–35, 37–40]
Finance, Banking, and Regulated Industries (Including Insurance)	Specialized LLMs can understand the "regulatory requirements, security needs, and performance constraints" especially in blockchain, cryptocurrency, smart contracts, and regulated environments. [21, 23, 32]	8	[4, 13, 17, 18, 21, 23, 32, 36]
Transportation and Safety-Critical Domains	The lack of transparency of LLMs is problematic, particularly in "safety-critical applications like healthcare, finance, or aerospace, where understanding the reasoning behind code is essential for ensuring security and correctness". [3]	7	[3, 15, 18, 21, 25, 36, 38]
Healthcare, Medical, and Biomedical Research	AI "accelerates the coding process by converting natural language or abstract research intent (a vibe) into functioning software modules, dramatically shortening development cycles in biomedical research and clinical application." [1].	5	[1, 4, 5, 18, 35]
Education and Training	LLMs "directly support learning processes through interaction and conceptual clarification". Students should be taught "how to specify AI, including functional, non- functional, and 'happy and unhappy path scenarios'" and learn "prompt engineering workarounds" and to "research and fact- check AI responses". [36]	5	[2, 4, 5, 15, 36]
Legal Reasoning and Regulatory Compliance	LLMs demonstrate success across multiple fields, such as education, healthcare, and legal reasoning. Specifically, prompt engineering is used to "tailor LLMs for the US legal system, enabling the model to identify relevant cases and predict judicial outcomes with greater precision". [22]	4	[17, 22, 35, 36]

Continued on next page

Industry Sector	Description	#	References
Digital Ecosystems (E-Commerce and Services)	Case studies involve the development of systems such as a food order and delivery system (FODS) and a smart wallet system (SWS). LLMs are applied in contexts like Online Retailers and the service industry. [4, 23]	3	[4, 17, 23]

A.3 Final set of studies retained by the review

Table A.5 lists the 31 retained studies, their venues, and the review questions each contributed to. Studies addressing multiple questions appear once with all relevant questions identified, allowing the synthesis and venue distribution to be audited study by study.

Table A.5: Studies retained by the systematic literature review

Reference	Venue	Year	RQs answered
[3]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5
[17]	Array	2025	1, 2, 4, 5
[35]	Expert Systems With Applications	2025	1, 2, 3, 4, 5
[13]	Information and Software Technology	2025	1, 2, 3, 4, 5
[18]	Information and Software Technology	2025	1, 2, 4, 5
[19]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 5
[31]	ACM Transactions on Software Engineering & Methodology	2024	1, 3
[20]	IEEE Access	2024	1, 2, 3, 4, 5
[14]	E3S Web of Conferences	2025	1, 2, 5
[16]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5
[37]	Engineering Applications of Artificial Intelligence	2025	1, 2, 3, 4, 5
[21]	BenchCouncil Transactions on Benchmarks, Standards & Evaluations	2025	1, 2, 3, 4, 5
[15]	Frontiers in Computer Science	2025	1, 2, 3, 4, 5
[5]	Advanced Science	2025	2, 5
[22]	ARO-The Scientific Journal of Koya University	2025	1, 2, 3, 4, 5
[32]	Information Systems & e-Business Management	2025	1, 2, 4, 5
[26]	Applied Computer Systems	2024	1, 2, 5
[23]	Information and Software Technology	2025	1, 2, 3, 4, 5
[29]	Journal of Computer Information Systems	2025	1, 2, 3, 4
[30]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3
[1]	BioData Mining	2025	1, 3, 5
[40]	Neurocomputing	2025	2, 3, 4, 5
[33]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5
[36]	Journal of Software: Evolution & Process	2025	1, 2, 3, 4, 5
[24]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5
[34]	Technology Audit & Production Reserves	2025	1, 2, 5
[25]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5
[39]	ACM Transactions on Software Engineering & Methodology	2025	2, 3, 5
[38]	Tehnički Glasnik	2025	1, 2, 4, 5
[2]	Informatica Economica	2025	1, 2, 3, 4, 5
[4]	ACM Transactions on Software Engineering & Methodology	2025	1, 2, 3, 4, 5

B

Evaluation Instruments

This appendix reproduces the evaluation instruments used in Chapter 9, including consent and demographics, workload, usability, the author-designed UX questionnaire, and interview guides. Instruments were available in English and Portuguese; scoring, interpretation and results are reported in Chapter 9.

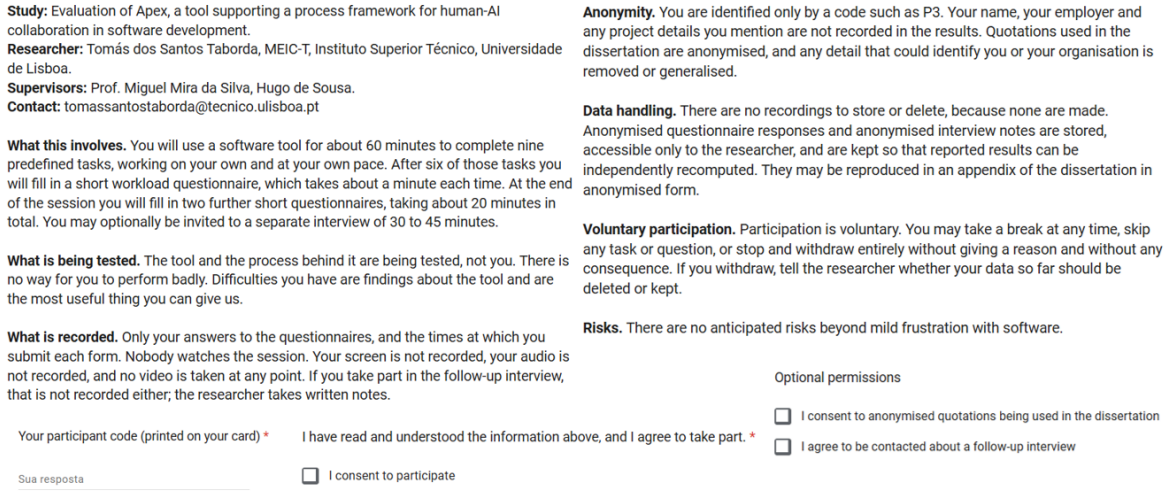
B.1 Instrument Access

Each form is reproduced later in this appendix; the live links are collected here so readers can access the same English or European Portuguese versions administered to participants (Section 9.2).

- **Informed consent and demographics** (Section B.2): EN https://docs.google.com/forms/d/e/1FAIpQLSezG6QwXDkCgLDEk0ebbsmTCs9EeIRoHNNGYavyeeWiOR1hrg/viewform and PT https://docs.google.com/forms/d/e/1FAIpQLSerEC9KkxLQj-nzo7qZsTkwpp9M4h0JHwk_atswO3adfm44kA/viewform
- **Raw NASA-TLX** (Section B.3): EN https://docs.google.com/forms/d/e/1FAIpQLSed3z8FdBJDgQuLcYEX746HHGpxP80799H3TDrG8dfn9CzTYg/viewform and PT https://docs.google.com/forms/d/e/1FAIpQLSdXAbp9UWHtLBkjJUQhylS6Xc-dRveePiRPS1ZGReq2uwEipw/viewform.
- **SUS** (Section B.4): EN https://docs.google.com/forms/d/e/1FAIpQLSc3Tr2pVcRaNEc4MHrThurZSwbg1dXIyeHOyDbL-3_sufSlOg/viewform and PT https://docs.google.com/forms/d/e/1FAIpQLSe0R3huJ3HOclxn8rRqVqcpinYwava5Ywla5FuUigfc4x2k7g/viewform
- **Apex UX questionnaire** (Section B.5): EN https://docs.google.com/forms/d/e/1FAIpQLSfNdyEC5U0eTVWO5aGChNckjW09Oe0R9h1Z9sSHbcQ--oGkrQ/viewform and PT https://docs.google.com/forms/d/e/1FAIpQLSfWmMYRncpvhvmqmncy7fW6_xjqZsuRUUDs1aiDocVnOCIXJA/viewform

B.2 Informed Consent and Demographics

Completed first, before any task or questionnaire; the ethics position it rests on is stated in Section 9.2. The consent form is reproduced in Figure B.1 as administered.



Study: Evaluation of Apex, a tool supporting a process framework for human-AI collaboration in software development.
Researcher: Tomás dos Santos Taborda, MEIC-T, Instituto Superior Técnico, Universidade de Lisboa.
Supervisors: Prof. Miguel Mira da Silva, Hugo de Sousa.
Contact: tomassantostaborda@tecnico.ulisboa.pt

What this involves. You will use a software tool for about 60 minutes to complete nine predefined tasks, working on your own and at your own pace. After six of those tasks you will fill in a short workload questionnaire, which takes about a minute each time. At the end of the session you will fill in two further short questionnaires, taking about 20 minutes in total. You may optionally be invited to a separate interview of 30 to 45 minutes.

What is being tested. The tool and the process behind it are being tested, not you. There is no way for you to perform badly. Difficulties you have are findings about the tool and are the most useful thing you can give us.

What is recorded. Only your answers to the questionnaires, and the times at which you submit each form. Nobody watches the session. Your screen is not recorded, your audio is not recorded, and no video is taken at any point. If you take part in the follow-up interview, that is not recorded either; the researcher takes written notes.

Anonymity. You are identified only by a code such as P3. Your name, your employer and any project details you mention are not recorded in the results. Quotations used in the dissertation are anonymised, and any detail that could identify you or your organisation is removed or generalised.

Data handling. There are no recordings to store or delete, because none are made. Anonymised questionnaire responses and anonymised interview notes are stored, accessible only to the researcher, and are kept so that reported results can be independently recomputed. They may be reproduced in an appendix of the dissertation in anonymised form.

Voluntary participation. Participation is voluntary. You may take a break at any time, skip any task or question, or stop and withdraw entirely without giving a reason and without any consequence. If you withdraw, tell the researcher whether your data so far should be deleted or kept.

Risks. There are no anticipated risks beyond mild frustration with software.

Optional permissions

☐ I consent to anonymised quotations being used in the dissertation

☐ I agree to be contacted about a follow-up interview

Your participant code (printed on your card) *

Sua resposta

I have read and understood the information above, and I agree to take part. *

☐ I consent to participate

Figure B.1: Informed consent form as presented to participants.

Demographics

Filled in immediately after consent, on the same form, in the participant's chosen language. The participant code linked the workload, usability and UX forms.

Table B.1: Demographics items collected on the consent form

Item	Question	Options
G1	Years of professional software development experience	under 1 / 1-3 / 4-7 / 8-15 / over 15 / student
G2	Current role	developer / tester or QA / product owner / designer / engineering manager / student / other
G3	Frequency of use of AI coding assistants	never / tried once or twice / monthly / weekly / daily
G4	Familiarity with Gherkin or BDD	none / have seen it / use it regularly
G5	Familiarity with the project management tool used in the session	none / some / use it regularly
G6	Preferred language for the session	English / Português

B.3 Raw NASA Task Load Index

The Raw NASA-TLX [57, 58] was administered after tasks 3, 4, 6, 7, 8 and 9. Responses used a 0-10 scale, rescaled to 0-100, with Performance presented last using reversed anchors; these adaptations are explained in Section 9.3. Portuguese participants received direct translations with the same subscales, order and anchors.

Table B.2: Raw NASA Task Load Index subscales, in the order presented

Subscale	Description	Scale
Mental Demand	How much mental and perceptual activity was required? Was it easy or demanding, simple or complex, forgiving or exacting?	Very Low - Very High
Physical Demand	How much physical activity was required? Was the task easy or demanding, slow or brisk, restful or laborious?	Very Low - Very High
Temporal Demand	How much time pressure did you feel due to the pace of the task? Slow and leisurely or rapid and frantic?	Very Low - Very High
Effort	How hard did you have to work, mentally and physically, to accomplish your level of performance?	Very Low - Very High
Frustration Level	How insecure, discouraged, irritated, stressed and annoyed, versus secure, gratified, content and relaxed, did you feel?	Very Low - Very High
Performance *(reversed anchors)*	How successful were you in accomplishing this task's goals?	**Perfect - Failure**

B.4 System Usability Scale

The SUS [52] was administered after Raw NASA-TLX using the standard 1-5 agreement scale, with *Apex* substituted for *the system*. Portuguese participants received the validated European Portuguese version [53]; its psychometric limitations are discussed in Section 9.3.1.

Table B.3: System Usability Scale items, as administered

#	Item
1	I think that I would like to use Apex frequently.
2	I found Apex unnecessarily complex.
3	I thought Apex was easy to use.
4	I think that I would need the support of a technical person to be able to use Apex.
5	I found the various functions in Apex were well integrated.
6	I thought there was too much inconsistency in Apex.
7	I would imagine that most people would learn to use Apex very quickly.
8	I found Apex very cumbersome to use.
9	I felt very confident using Apex.
10	I needed to learn a lot of things before I could get going with Apex.

B.5 Apex UX Questionnaire

The author-designed UX questionnaire, described in Section 9.3.3, used a 1-5 agreement scale plus *N/A*. Seventeen items were administered, with E2-E3 evaluating the framework separately from Apex; the removed eighteenth item is explained in Section 9.3.3. Portuguese participants received direct translations preserving the same constructs, items and open questions.

Table B.4: Apex UX questionnaire items, by construct

Item	Statement
A. Understanding where you are	
A1	At any moment, I knew which phase of the process I was in.
A2	I understood what the tool expected me to do next.
A3	I could tell the difference between work the tool had finished and work it had not.
B. Working with AI-generated content	
B1	I could tell which parts of the content were generated by AI and which were mine.
B2	Reviewing the AI's output took less effort than writing it myself would have.
B3	I felt able to reject or change the AI's output when I disagreed with it.
B4	I trusted the generated content enough to use it in a real project.
B5	I could see what information the AI had been given as a basis for its output.
C. Control and reversibility	
C1	I felt in control of what the tool did on my behalf.
C2	I was confident I could undo or redo a step if I made a mistake.
C3	I understood the consequences of locking a phase before I did it.
C4	Waiting for the AI to finish was acceptable.
D. Feedback and errors	
D1	When something went wrong, the tool told me clearly what had happened.
D2	The messages the tool showed me told me what to do next, not only what failed.
E. Fit to real work	
E1	The artefacts the tool produced are ones my team would actually use.
E2	The process the tool imposes fits the way my team works.
E3	The effort of following the process is worth what it produces.
Open Questions	
1	What was the single most confusing thing about using Apex?
2	Was there a point where you did not trust what the tool had produced? What made you doubt it?
3	If you could change one thing about the interface, what would it be?

Full Item-by-Item Responses

Response counts for all seventeen administered items, with the medians reported in Section 9.4.3.

Table B.5: Apex UX questionnaire responses ($N = 13$). Scale 1-5 from strongly disagree to strongly agree; *N/A* responses are excluded from the median. E2-E3 evaluate the framework rather than Apex.

	Item	1	2	3	4	5	N/A	Mdn
A. Understanding where you are								
A1	Knew which phase of the process I was in	0	0	3	4	6	0	4
A2	Understood what the tool expected next	0	0	4	5	4	0	4
A3	Could tell finished work from unfinished	0	0	4	7	2	0	4
B. Working with AI-generated content								
B1	Could tell generated content from my own	0	2	0	3	7	1	5
B2	Reviewing took less effort than writing it	0	0	2	6	5	0	4
B3	Felt able to reject or change the output	0	0	0	1	11	1	5
B4	Trusted it enough to use in a real project	0	0	3	8	2	0	4
B5	Could see what information the AI was given	0	1	2	6	4	0	4
C. Control and reversibility								
C1	Felt in control of what the tool did	0	1	3	8	1	0	4
C2	Confident I could undo or redo a step	0	1	1	2	8	1	5
C3	Understood the consequences of locking a phase	0	1	4	2	6	0	4
C4	Waiting for the AI to finish was acceptable	0	0	0	4	9	0	5
D. Feedback and errors								
D1	Was told clearly what had happened when something went wrong	0	1	6	4	2	0	3
D2	Messages said what to do next, not only what failed	0	0	4	3	6	0	4
E. Fit to real work								
E1	The artefacts are ones my team would use	0	1	3	5	1	3	4
E2	*(framework)* The process fits how my team works	0	2	2	6	0	3	4
E3	*(framework)* The effort is worth what it produces	0	0	2	5	6	0	4

B.6 Interview Guides

Two semi-structured interview guides evaluated the framework rather than Apex in separate, unrecorded 30-45 minute sessions. Questions were used flexibly and freely as prompts.

Table B.6: Semi-structured interview guides

#	Prompt
Practitioners Guide	
1	Describe how your team moves from an idea to production, where AI appears in that process, who introduced it, and what most often goes wrong.
2	Which of the framework's six phase-end decisions would your team genuinely make, which might be rubber-stamped, and which phase would face the most resistance?

Continued on next page

#	Prompt
3	How does your team currently work without a prior specification, what would the framework change, and where might that create unnecessary delay?
4	What conditions would support adoption for one project, and what would cause abandonment after two sprints?
5	Who would need to own the framework for it to persist, and does that role exist in your team?
6	If Apex disappeared but the framework remained, which parts would you continue to use?
7	When AI-generated code fails in production, who is accountable today, and would the framework change that?
8	Does the recorded history of decisions and accountability have value outside the development team?
9	What is missing from the framework, and what important question was not asked?
Agile and Scrum Experts Guide	
1	How would you describe the framework in your own words?
2	Where does it align with or depart from Scrum and Agile, and are those departures justified by AI-specific changes?
3	Are the explicit phase gates useful guardrails or a return to Waterfall-style stage gating?
4	Does treating roles as hats work in practice, or does it collapse into one person holding every responsibility?
5	Which required artefacts would you add or remove, and is the documentation proportionate to its value?
6	What kinds of teams are poorly suited to the framework, and what would a coach need to teach for correct adoption?
7	Under what conditions would you recommend the framework to a client, and when would you refuse?
8	What is the strongest argument for the framework, and what is its weakest point?

B.7 Per-Participant Raw Scores

Table B.7 reports participant-level SUS scores and subscales, Table 9.2 reports workload means by task, and Table B.5 reports complete UX response counts. The larger participant-level workload and UX matrices are omitted for brevity.

Table B.7: SUS score and Usability and Learnability subscales per participant and administration language. All scores range from 0 to 100 and are not percentages.

Participant	Language	SUS	Usability	Learnability
P0	EN	42.5	50.00	12.5
P1	EN	75.0	81.25	50.0
P5	EN	80.0	87.50	50.0
P6	EN	15.0	12.50	25.0
P7	EN	62.5	71.88	25.0
P8	EN	60.0	59.38	62.5
P9	EN	60.0	65.63	37.5
P10	EN	75.0	75.00	75.0
P11	EN	65.0	56.25	100.0
P12	EN	70.0	75.00	50.0
P2	PT	37.5	46.88	0.0
P3	PT	40.0	46.88	12.5
P4	PT	55.0	50.00	75.0
Mean, all 13		**56.73**	**59.86**	**44.23**
Mean, English ($n = 10$)		60.50		
Mean, Portuguese ($n = 3$)		44.17		