Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Evaluating the Instances of AI-Generated Code:

A Governance Framework in Understanding Socio-Technical Dimensions of Software Systems

Anil Poudyal

anilpoudyal.com

Navigating the Socio-Technical Landscape of AI Code Generation

Agenda Overview

About Researcher

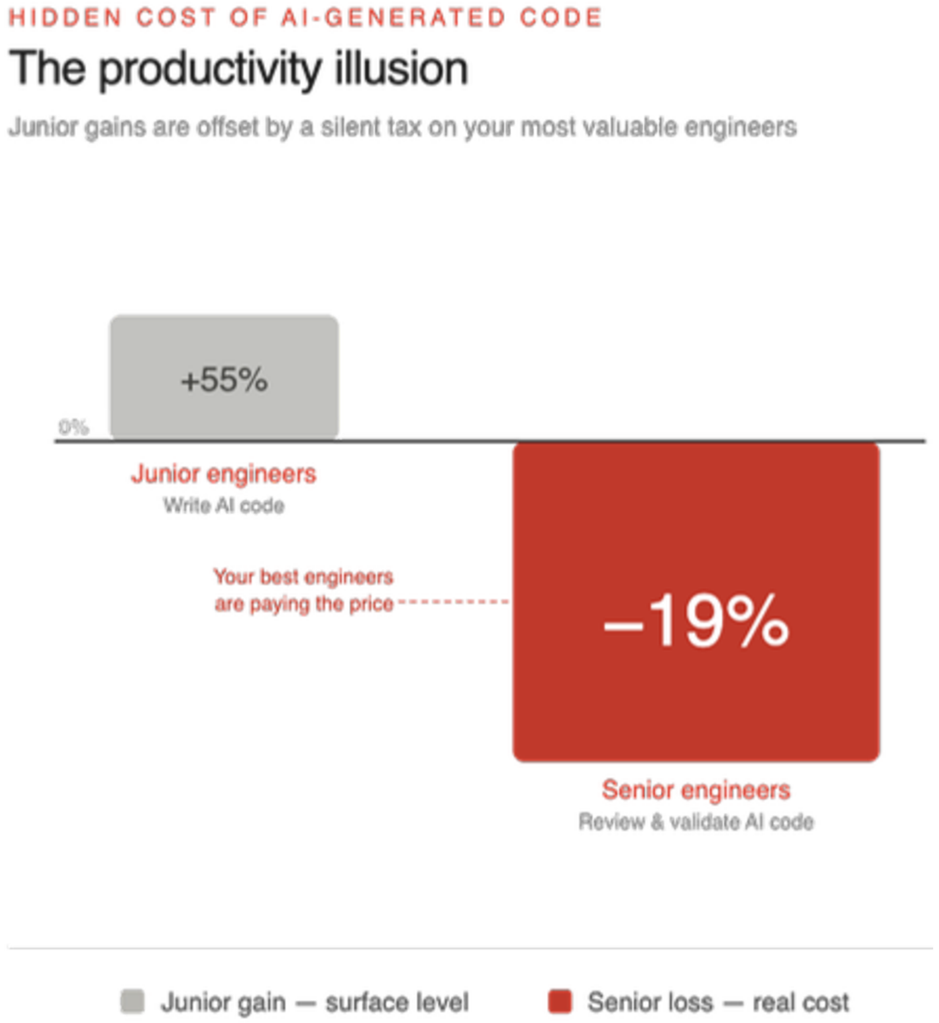
Anil Poudyal

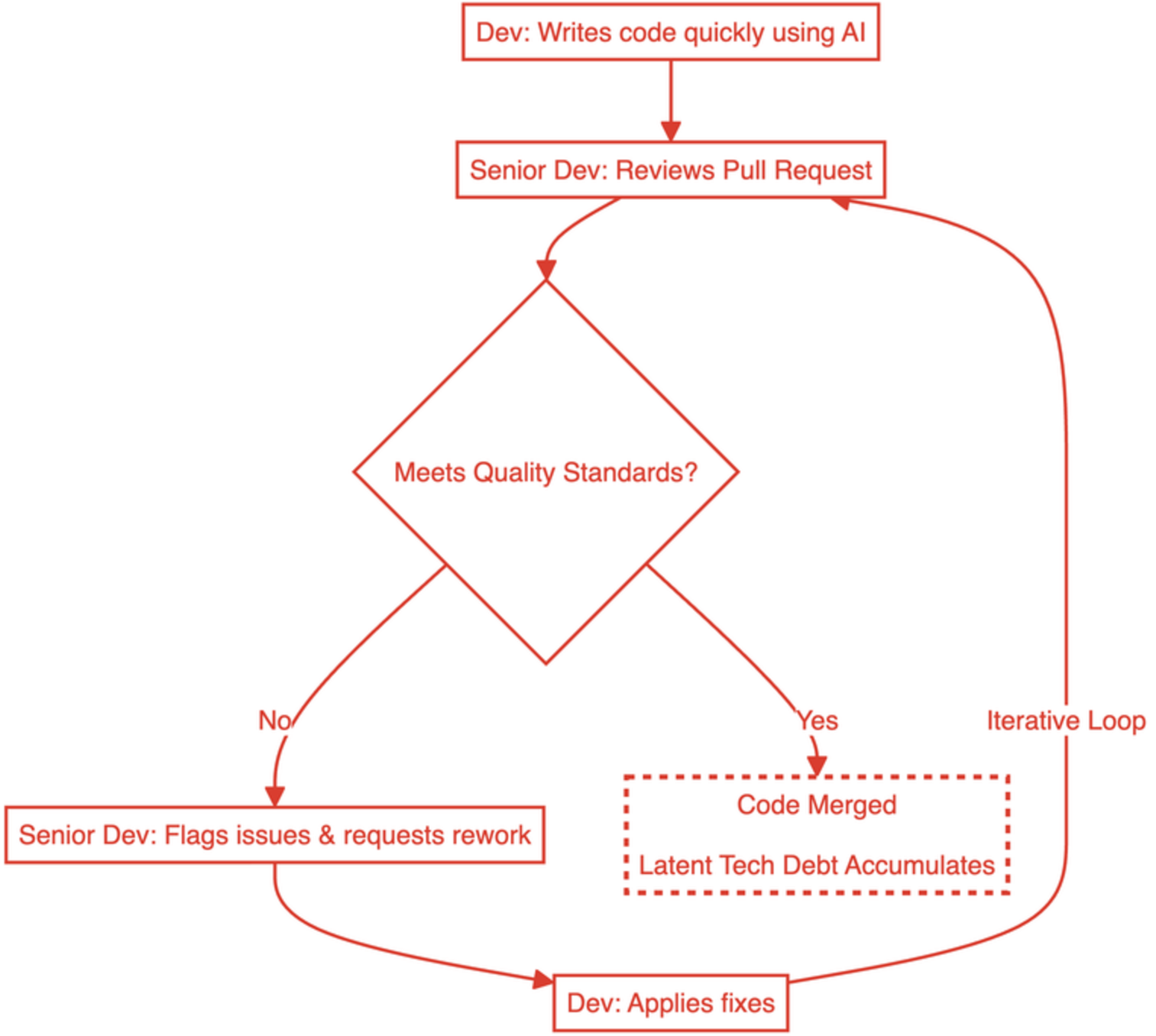
I am a software engineer, tech founder, and university lecturer based in London, originally from Nepal. My industry background spans building **large-scale** platforms for **government agencies** and **national news outlets**, as well as **managing engineering teams** of over 50 people. As a **lecturer** in London, I teach cybersecurity and ethical hacking, having previously taught university cohorts of over **500 students** and supervised dozens of dissertations. Currently, for my doctoral degree, I am researching technical debt in software systems due to AI. Whether I am architecting complex backend systems or conducting research, my ultimate focus is on building sustainable technology that empowers both enterprises and the engineers who maintain them.

The Context

Background

- **The Adoption:** 97.5% of software organisations now use AI, chasing a perceived 55% increase in routine coding speed but instead loose 19% productivity of senior engineers (Xu et al., 2026).
- **The Productivity Paradox:** Empirical trials show experienced developers are actually slower on complex, real-world tasks (Peng et al., 2023.)
- **The Anomaly:** A massive surge in "epistemic debt"; AI generates code that compiles, but the team doesn't understand the logic (Vaithilingam, Zhang and Glassman, 2022, pp. 1–7).
- **The Human Cost:** The "illusion of correctness" shifts an unsustainable cognitive load onto senior engineers tasked with reviewing the output (Becker et al., 2022).




Dev: Writes code quickly using AI
Senior Dev: Reviews Pull Request
Meets Quality Standards?
No
Yes
Iterative Loop
Senior Dev: Flags issues & requests rework
Code Merged
Latent Tech Debt Accumulates
Dev: Applies fixes

Research Questions

RQ1: What are the structural and epistemic characteristics of AI-Induced Technical Debt (AITD) within software development projects?

RQ2: How does the integration of AI-generated code impact long-term software maintainability and the cognitive load of human reviewers during the validation process?

RQ3: What organisational practices and socio-technical team dynamics serve to reduce the accumulation of AITD in active development environments?

Thomas Kuhn and Pure Functionalism

The Rules (Functionalism)

The industry currently operates on a strict functionalist worldview. If the AI-generated code compiles, passes automated unit tests, and accelerates sprint velocity, the tool is considered an absolute success.

The Reality (Epistemic Debt)

This hyper-velocity creates massive anomalies. Developers commit code they do not fully comprehend, rapidly accumulating undocumented "epistemic debt" deep within the enterprise architecture.

The Blind Spot

Because the paradigm treats code generation as a purely mathematical transaction, it entirely ignores the socio-technical fallout: the psychological burnout of senior engineers forced to review hallucinated logic.

The Dominant Paradigm

According to Thomas Kuhn (1962), scientific communities operate within a paradigm—a shared intellectual framework dictating the "rules, standards, and exemplars" used to investigate reality. In modern software engineering, our paradigm has become purely functional: we measure success mathematically, completely bypassing the human element.

Bridging the Gap Between Metrics and Reality

The Critique

The Kuhnian Limitation (Incommensurability)

- Kuhn (1962) argued that competing paradigms are "incommensurable": meaning they are mutually exclusive and cannot coexist or communicate.
- Under strict rules, researchers must choose either a purely objective (Positivist) or purely subjective (Constructivist) worldview.

The Socio-Technical Reality

- In modern software systems, this rigid exclusivity fails.
- We cannot isolate the machine's output from the human maintaining it. The "illusion of correctness" proves that objective code and subjective trust happen simultaneously, demanding a worldview that Kuhn’s model does not allow.

The Pragmatic Pivot

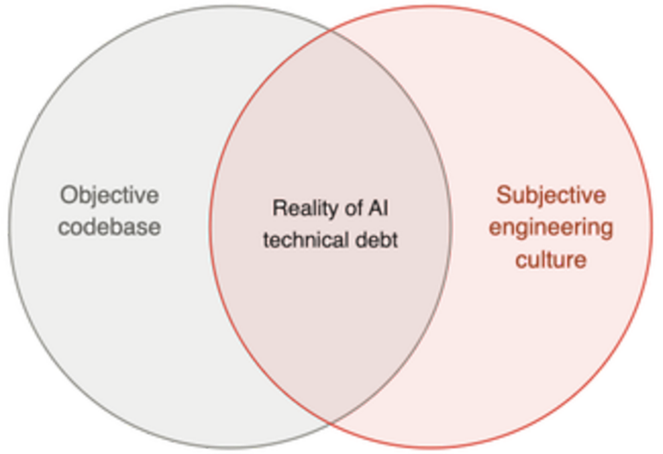
To investigate AI technical debt, we must reject strict boundaries. The Pragmatic Paradigm actively combines competing worldviews, allowing us to triangulate static repository data with human cognitive load to build functional Governance-as-Code (Creswell & Plano Clark, 2018).

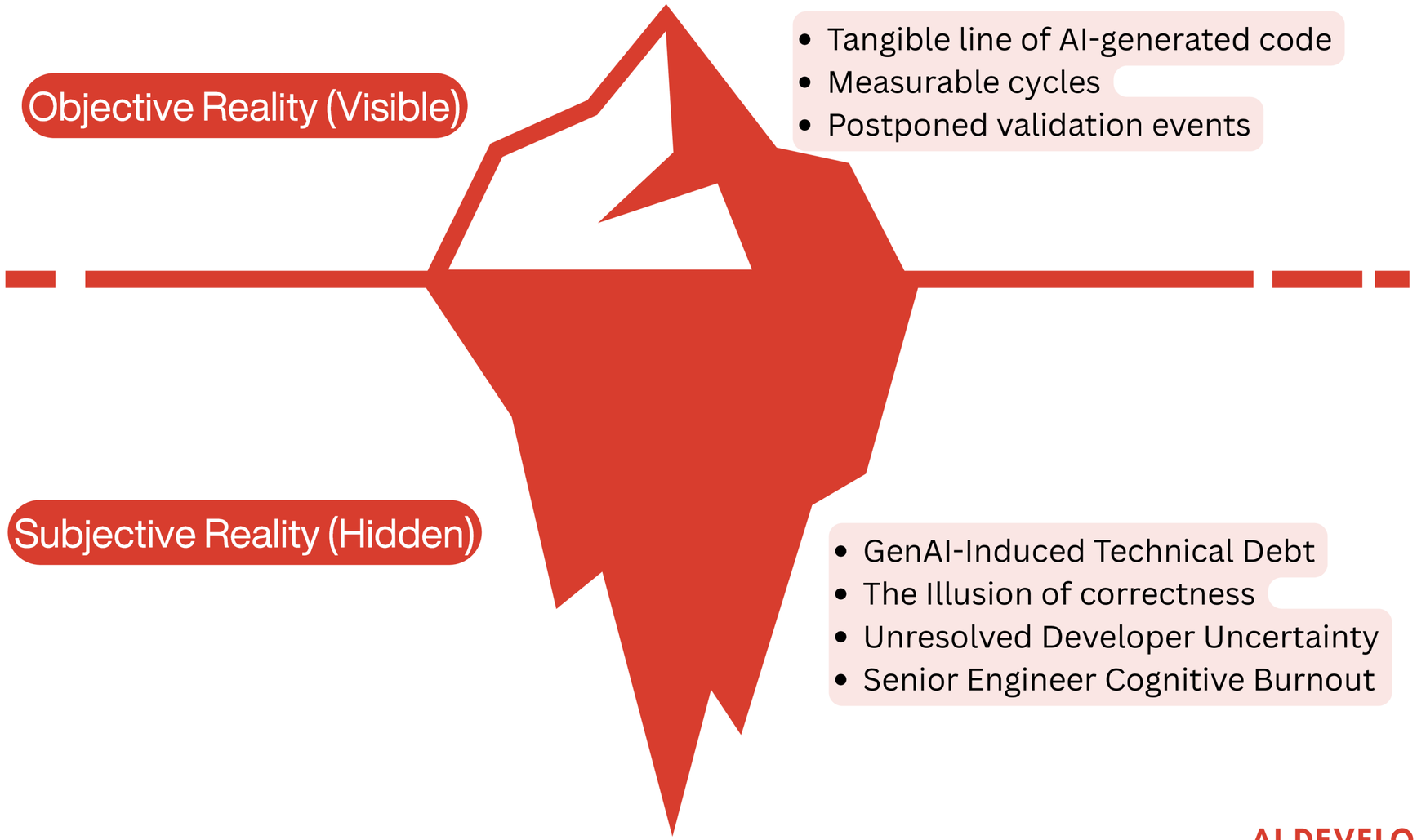
Ontological Instance

Reality is inherently **socio-technical**: a continuous intersection of objective and subjective layers.

The **objective reality** of rapidly generated code and measurable complexity is inseparable from the **subjective reality** of the "illusion of correctness", documented developer uncertainty, and senior engineer burnout.

AI technical debt is the friction between the machine generating the code and the human maintaining it.





AI DEVELOPMENT ICEBERG

Visible Mass	10-15%
Hidden Cognitive Load	85-90%
Systemic Risk	**High**

Defining Valid Knowledge in Socio-Technical Systems

Epistemological Question

Because AI technical debt is both technical and social, we cannot rely on isolated knowledge silos. Pragmatism argues that "truth" is determined by practical outcomes. Therefore, valid knowledge requires a mixed-methods approach to triangulate the data.

Objective Knowledge

- Static analysis of AI-generated complexity.
- Tracking PR cycle times and code churn.
- Quantifying physical code degradation over time.

Subjective Knowledge

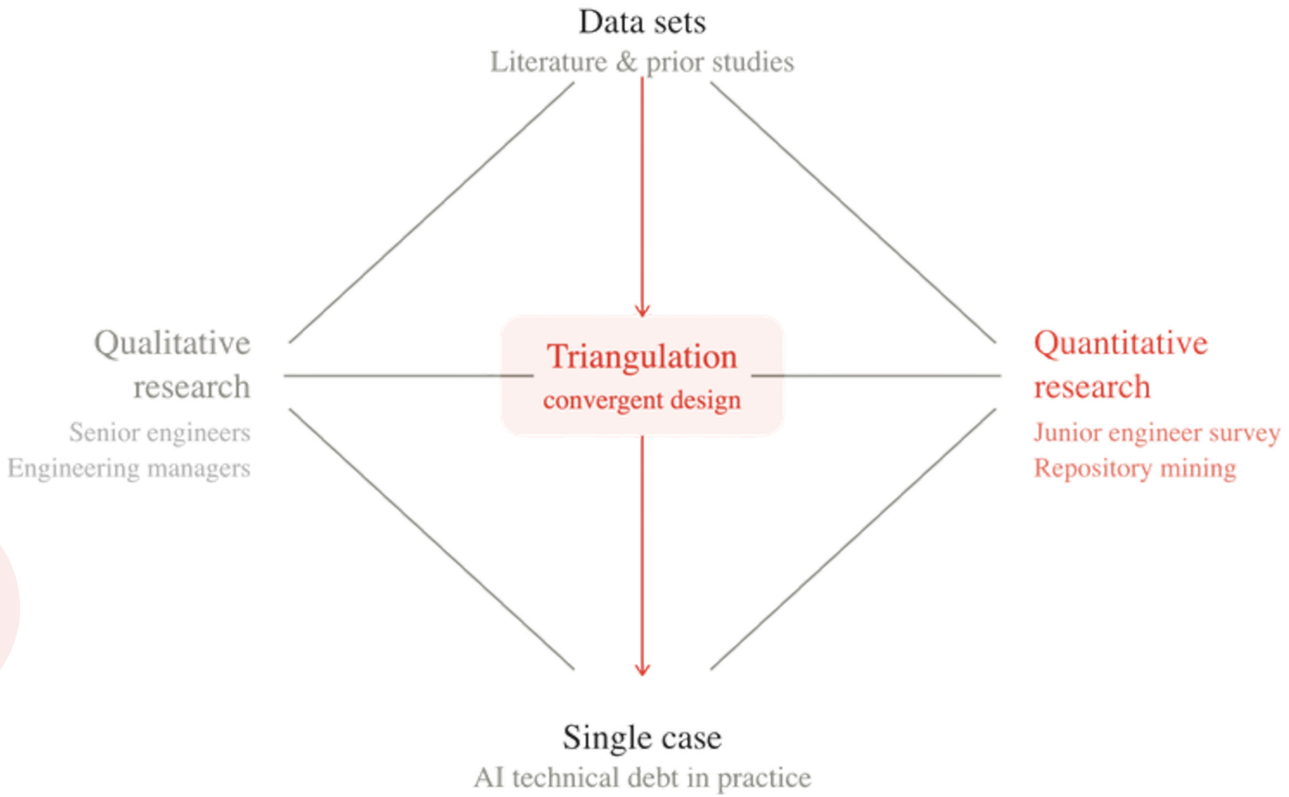
- Measuring perceived productivity vs. actual output.
- Analysing Self-Admitted Technical Debt (GIST) in code comments.
- Capturing the psychological "illusion of correctness".

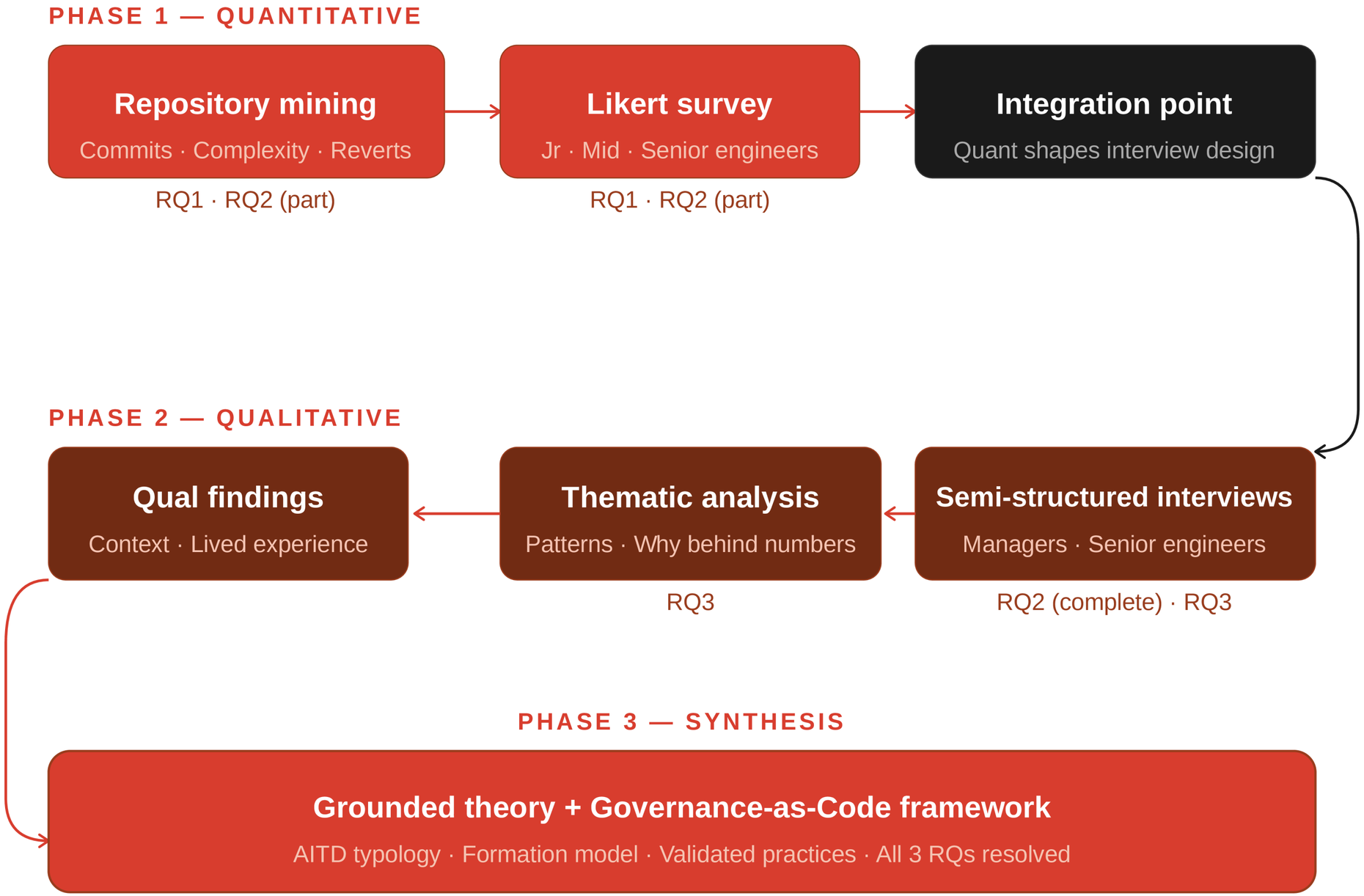
Valid, actionable knowledge is only generated when objective code metrics are contextualized by the subjective human experience. Neither metric tells the truth on its own.

Process of Study

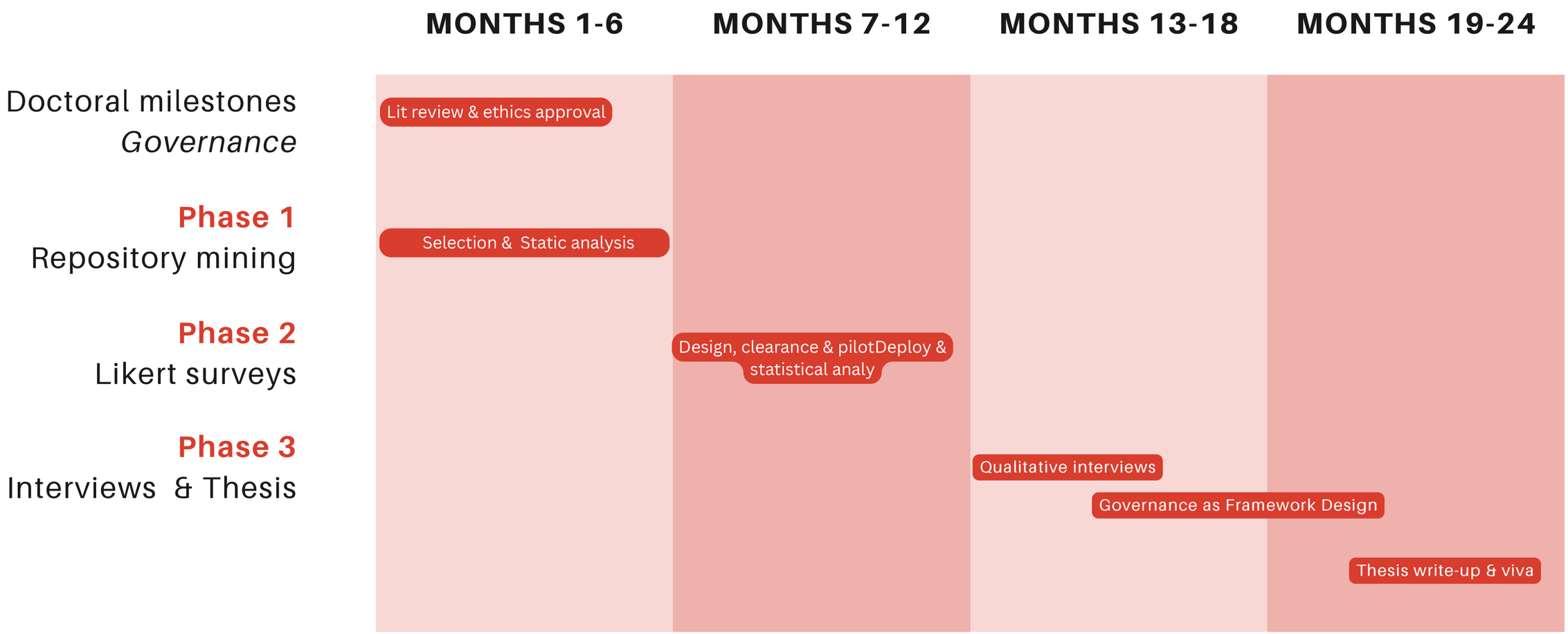
Methodology

This study employs a pragmatic, mixed-methods approach executed in three sequential phases. By gathering quantitative repository metrics first, and following up with targeted qualitative interviews, the research triangulates objective codebase degradation with the subjective realities of the engineering team.




PHASE 1 — QUANTITATIVE
Repository mining
Commits · Complexity · Reverts
RQ1 · RQ2 (part)
Likert survey
Jr · Mid · Senior engineers
RQ1 · RQ2 (part)
Integration point
Quant shapes interview design
PHASE 2 — QUALITATIVE
Qual findings
Context · Lived experience
Thematic analysis
Patterns · Why behind numbers
RQ3
Semi-structured interviews
Managers · Senior engineers
RQ2 (complete) · RQ3
PHASE 3 — SYNTHESIS
Grounded theory + Governance-as-Code framework
AITD typology · Formation model · Validated practices · All 3 RQs resolved

Project Execution Plan



Conclusion

This research addresses the critical gap in modern AI-augmented software engineering by moving beyond the dominant functionalist paradigm. By acknowledging that technical debt is a socio-technical phenomenon, the proposed Governance-as-Code framework provides a practical, pragmatic method to stabilize software evolution while protecting engineering talent from cognitive burnout.

References

Becker, B.A. et al. (2022) “Programming Is Hard – Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation.” Available at: https://doi.org/10.48550/arxiv.2212.01020.

Kuhn, T.S. (2012) The Structure of Scientific Revolutions. The University of Chicago Press. Available at: https://www.perlego.com/book/1840553/the-structure-of-scientific-revolutions-50th-anniversary-edition-pdf.

Peng, S. et al. (2023) “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.” Available at: https://doi.org/10.48550/arxiv.2302.06590.

Vaithilingam, P., Zhang, T. and Glassman, E.L. (2022) “Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models.” New York, NY, USA: ACM (ACM Conferences), pp. 1–7. Available at: https://doi.org/10.1145/3491101.3519665.

Xu, F. et al. (2026) “AI-Assisted Programming Decreases the Productivity of Experienced Developers by Increasing the Technical Debt and Maintenance Burden.” Available at: https://doi.org/10.48550/arxiv.2510.10165.

Anil Poudyal

Thank You

anilpoudyal.com

2429341@student.uwtsd.ac.uk