London School of Economics and Political Science

Department of Management

Rewriting the Developer: Professional Identity Reconstruction in the Age of AI Coding Tools.

Dissertation

MG4D7

Wordcount: 8791

Submitted by Candidate 42439

Supervisor: Associate Professor Maha Shaikh

Abstract

As artificial intelligence coding tools proliferate across the profession, a fundamental question emerges: are developers simply adopting new tools, or is the very nature of programming being reconstructed? This dissertation investigates how AI coding tools transform not just developer productivity, but professional practice, identity, and knowledge production itself.

Through semi-structured interviews with fourteen software professionals across startups, enterprises, and consultancies, this qualitative study employs a novel dual-lens theoretical framework combining Actor-Network Theory and Affordance-Actualisation Theory. This reveals how identical AI coding tools produce different professional transformations based on how the developers integrate these tools into their roles, and their level of comfortability in delegating tasks to these tools. Thematic analysis uncovered three distinct transformation patterns, challenging linear adoption narratives.

Complete Transformers (36%) fundamentally reconstruct their professional identity from "code writers" to "AI orchestrators," using their own AI coding tools, proprietary organisational systems that amplify AI capabilities through recursive knowledge accumulation. **Selective Adopters** (50%) maintain strategic boundaries between human and AI work, articulating sophisticated decision frameworks: "it's a tool, not a rule." **Conscious Resisters** (14%) actively preserve traditional expertise and reject delegating any roles to AI coding tools willingly.

Two critical phenomena emerged across the three role transformation types discovered through the research. First, "vibe coding", where developers iterate conversationally with AI until solutions "feel right", represents an epistemic shift from deterministic to intuitive programming practices. Second, all interviewed participants expressed profound concern about junior developer skill development, as AI automation of entry-level tasks collapses traditional learning pathways, potentially affecting how the profession reproduce essential expertise.

The Network-Affordance Integration Model developed through this research explains differential transformation outcomes, revealing professional practice transformation as ecosystem diversification rather than convergence. These findings have immediate implications for organisational strategy, educational design, and individual career planning. As knowledge professions broadly confront AI integration, understanding software development's early transformation provides crucial insights for navigating the fundamental reconstruction of professional work in the AI era.

Table of Contents

List of Abbreviations

AAT - Affordance-Actualisation Theory

AI - Artificial Intelligence

ANT - Actor-Network Theory

API - Application Programming Interface

GDPR - General Data Protection Regulation

HIPAA - Health Insurance Portability and Accountability Act

HTML - HyperText Markup Language

ID - Identifier

IDE - Integrated Development Environment

IS - Information Systems

IT - Information Technology

LSE - London School of Economics and Political Science

METR - Model Evaluation and Threat Research

ML - Machine Learning

NBER - National Bureau of Economic Research

OECD - Organisation for Economic Co-operation and Development

PMC - PubMed Central

SEC - Securities and Exchange Commission

TAM - Technology Acceptance Model

UI - User Interface

UTAUT - Unified Theory of Acceptance and Use of Technology

List of Figures

List of Tables

Chapter 1: Introduction

Among the software developers interviewed for this study, diverse approaches to AI tool integration are emerging. With 76% of developers using or planning to use AI coding tools daily (Stack Overflow, 2024), senior developers are redefining their role from manually coding towards orchestrating AI driven ecosystems. Already, there have been projections that there will be a majority shift of "manual coders to orchestrators" by 2030 (Ammar et al., 2024).

Junior developers using AI coding tools can produce complex code at an unprecedented level but would struggle to explain how it works or build on it further (Osmani, 2025). Organisations diverge in their approaches, allowing AI coding tool adoption while other organisations maintain their traditional development practices (GitHub, 2024; Google Cloud, 2025). This ongoing role transformation demands investigation beyond productivity metrics to understand how these tools reconstruct this professional practice for software developers.

1.1 The Research Gap

Despite AI coding tools' rapid adoption there is a lack theoretical understanding of its transformative impact. Current literature has three critical limitations that this research paper addresses.

Firstly, the field's quantitative bias toward using productivity metrics neglects the changes to the human technological interaction of this transformation. Traditional IS adoption models for technology like TAM (Davis, 1989) and UTAUT (Venkatesh et al., 2003) assumes stable technology boundaries that recede over time as developers reconsider their roles changing from code writers to AI orchestrators. These critical reviews argue these models have "a narrow perspective, which focuses only on individual adopters' beliefs, perceptions and usage intention" failing to capture the fundamental task and responsibility transformations occurring with AI coding tool implementation (Holden and Karsh, 2019).

The second limitation is that assuming uniform adoption overlooks the different patterns in how people are using AI tools. Current frameworks struggle to capture the nuanced and varied strategies developers employ when integrating AI coding tools, and the separation between AI appropriate and human essential tasks. This selective integration of tools

challenges the technological determinism pervading AI discourse, yet empirical investigation of these boundary-setting practices remains commonly absent from literature.

Industry analyses confirm that entry-level roles face existential challenges, with 77% of leaders expecting AI to enable junior workers to take on greater responsibilities while simultaneously reducing hiring needs (Microsoft, 2024). Industry analysis warns that AI coding assistants may lead to fewer junior developer positions as teams reshape around senior roles overseeing AI-generated code (Gross, 2025). Yet we lack understanding of the systemic implications for professional development and skill transfer.

1.2 Research Question and Objectives

Therefore, this dissertation asks: **How do AI coding tools transform the professional practice of software developers?**

This question explores not only whether developers adopt AI coding tools, but how these tools reshape what developers do, how they work, and how they perceive themselves. The transformation extends beyond productivity to encompass the full spectrum of professional practice of developers.

This central question is addressed through three integrated objectives:

Objective 1: Examine how AI coding tools change task allocation and workflow patterns in software development, investigating role redistribution between human and artificial actors.

Objective 2: Identify emerging core competencies and traditional skill transformations as developers integrate AI coding tools, exploring how different actualisation strategies lead to new skill portfolios.

Objective 3: Understand how developers reconceptualise their professional identity and future career trajectories.

1.3 Intended Contributions

This research aims to advance IS theory by synthesising ANT and AAT to explain the different transformation outcomes in AI coding tool adoption. Through qualitative investigation of software developers' experiences, this study seeks to:

1. Challenge the linear adoption models by showing multiple transformation pathways

2. Identify organisational strategies that determine AI coding tool effectiveness

3. Document user resistance as a legitimate response to address success bias

4. Examine the implications for junior developers' skill development and future employment.

With this research the dissertation contributes to understanding software development's transformation as a potential template for other professions experiencing AI integration. The following chapter examines existing literature across information systems, software engineering, and organisational research.

Chapter 2: Literature Review

The integration of AI coding tools in software development represents a socio-technical transformation. The academic landscape on this topic is young and remains fragmented and superficial. This review synthesises literature from information systems, software engineering, and organisational research to explore how AI tools reconfigure programming work, professional identity, and organisational capabilities. We exclude purely technical evaluations of AI performance, ethical debates about AI consciousness, and speculative future scenarios to maintain focus on empirically grounded professional transformation.

2.1 Traditional Information Systems Perspectives on Analysing Technology Use

Some traditional IS adoption frameworks are not equipped to understand AI coding tools. The Technology Acceptance Model (Davis, 1989) and its evolution into UTAUT (Venkatesh et al., 2003) assumes technology like coding tool would have clear boundaries between user and tool. Recent attempts to extend TAM with AI-specific constructs (Dahri et al., 2024) still treats AI as an enhanced tool rather than a transformative force that reconstructs the role and tasks for the user. AI coding supersedes the boundaries set in these papers.

Orlikowski (2007) sociometrical perspective is better suited for this type of analysis. Her argument that "the social and the material are constitutively entangled in everyday life" (p. 1437) directly applies to developers whose work practices become intertwined with AI functionality. When looking at GitHub Copilot integration, it transcends tool use, it represents a change in the entire production process for code. With AI coding tools, this entanglement is literal: human intention and AI capability merge in the act of producing code. The developer thinks, AI generates, developer evaluates.

Leonardi (2011) concept of materiality states that technology has features exist across different use contexts, but how people use it still depends on the context of the user. For AI coding tools, this means that AI always has the same capabilities such as generating code from prompts. But what happens depends on the organisation (startup vs. bank), the person's skills, and the rules they must follow. This explains how the same AI coding tool produces amazing results for one developer and frustration for another.

2.2 Actor-Network Theory: AI Coding Tools as An Active Participant in Development

Actor-Network Theory (ANT), developed by Bruno Latour, Michel Callon, and John Law in the 1980s, provides an approach to understanding how socio-technical systems treat human

and non-human entities as equally of action within networks. This theory has been further expanded in Latour (2005), which continues to treat human and non-human actors equally.

ANT treats non-humans as actors shaping outcomes (Latour, 2005). This matters because developers already see AI as a partner. Bird et al. (2022) found developers describing the relationship with AI as symbiotic, with AI having an active role in development. When GitHub Copilot suggests code that solves a problem the developer hadn't fully articulated, it's not just responding to commands. It's participating in problem-solving. It is not a passive tool but engaging the user directly and sometimes leading the creation of code.

Callon (1986) translation process shows how AI becomes embedded in development networks. Problematisation stage occurs when organisations flag their competitive disadvantage without using AI tools. Intersegment, by which actors attempt to impose and stabilise the identity of other actors by creating mechanisms that lock actors into set roles within the network. For AI adoption this happens with pilot programs, trainings and incentive structures that position AI tools as indispensable. Enrolment sees developers onboarding new AI-mediated practices. Mobilisation is the achievement of stable AI-integrated workflows.

Law (2009) "network multiplicity" also can be used to describe why developers use AI for some tasks and not others. This can be analysed by ANT by considering each task as a parallel network. AI-augmented for new projects, traditional processes for legacy systems, creating what he terms "partially connected" networks that challenge binary enrolled/not-enrolled categorisations.

Law's approach allows actors to simultaneously participate in multiple, overlapping networks with different rules and relationships. This is more accurate to how developers navigate between AI-assisted and traditional development practices rather than simply adopting or rejecting AI tools wholesale. In modern research on AI integration into work tasks (Barke et al., 2023), developers have been seen shifting between stages of integration, using AI coding tools in specific tasks while resisting in others.

2.3 Affordance-Actualisation Theory: Why Tools have Different User Outcomes

ANT explains network reconfiguration; AAT can be used to explain why the same AI coding tools produce different outcomes for different users. Strong et al. (2014) organisational affordance actualisation framework identifies three components to analyse tool use with

affordance existence (what technology allows), affordance perception (what users recognise) and affordance actualisation (what happens in practice).

This framework, building on Gibson (1979) ecological psychology and Markus and Silver (2008) IS adaptation, explains the varying levels of transformation among users. This framework can move beyond deterministic views of technology adoption to explain how organisational and individual factors shape the context which dictates how technology is used. This is crucial for understanding why some developers adopt AI tools while others resist despite using identical technology. Star and Griesemer (1989) "boundary objects" concept helps here, AI tools mean different things to different communities or users while maintaining enough coherence to enable collaboration and mutual comprehension.

Volkoff and Strong (2013) can help explain the effect of organisational context on how an AI coding tool is used. A senior developer in a startup accepts AI's "rapid prototyping" use and implements it. A junior developer in banking may know about use for the AI Coding tool to prototype, but organisational policies prevent the junior for using the tools in specific way. This means transformation is not dictated solely on technology capability, but the complex interplay of perception, intention, and context.

For example, Strong et al. (2003) research on electronic health record implementation found that the hospitals had access to the same data visualisation tools, yet only the hospitals with analytical culture and dedicated resources used these tools into improved patient outcomes.

While the term "collaborative affordances" requires further theoretical development, recent research on human-AI interaction patterns (Gomez et al., 2024) reveals how human-AI partnerships allow for the creation of new processes that were not possible before implementation. Yet this work exhibits success bias documenting mostly successful integration of AI coding tools while ignoring failed attempts, abandoned adoptions and refused affordances.

2.4 AI Coding Tool Productivity Increases and Task Augmentation

Peng et al. (2023) GitHub study is a corner stone paper for the productivity narrative that dominates discourse. Their controlled experiment found 55.8% faster task completion with Copilot, and this become the most-cited metric in academic and industry discussions. But this metric obscures a deeper change code produced.

However, methodological scrutiny reveals significant limitations on their findings. Tasks were artificially constrained with clear specifications while success was measured on completion time only. In addition, code quality metrics were absent. This approach provided valuable baseline data but failed to capture the complexity of real-world development where requirements are ambiguous, quality depends on more than speed, and long-term maintainability is essential.

Ziegler et al. (2022) found context matters in AI coding tools increasing productivity, finding gains in only new projects, and seeing productivity losses in legacy code bases and older projects. More concerning: Osmani (2024) documents the "70% problem". Developers accept 70% of AI suggestions but cannot evaluate their quality. Brynjolfsson et al. (2023) found similar patterns in other fields, impressive metrics hiding capability erosion.

2.5 Concern on AI Code Security and Declining Developer Skills

The phenomenon of "competence without comprehension" is empirically underexplored. The absence of any longitudinal studies tracking skill evolution represents a critical gap in literature. The universal concern about junior developer mentorship which was documented across industry discussions (Engineering Enablement, 2025) reveals communities recognising threats to existence.

Industry voices confirm fears. Osmani (2025a) warns of "skill atrophy," while Goel (2025) documents juniors who "can't actually code" without AI. The profession risks what Polanyi (1966) called losing "tacit knowledge" the unexplainable expertise that defines mastery. Developers are losing what Schön (1983) would call "reflection-in-action." They produce without understanding, breaking the feedback loop essential to expertise.

Security concerns are also an underexplored risk. Perry et al. (2023) analysis of Copilot-generated code found that 40% of code contained security vulnerabilities, significantly higher than human code. Even then developers expressed increased confidence in that code despite producing less secure code. METR (2025) randomised controlled trial found that senior developers were 19% slower using AI tools despite thinking they were 20% more productive. Traditional metrics like quality of code or quantity of task completion become meaningless when developers shift from writing to orchestrating, from creating to curating.

2.6 Identity and Role Perception

In Wenger (1998) communities of practice framework, though sparingly applied to AI transformation, offers unique insights. Professional communities maintain identity through shared practices, languages, and values. AI tools fracture these communities. When developers "pair program" with AI while others maintain traditional practices, this shared identity erodes.

This pattern echoes earlier technological transformations. Braverman (1974) and Noble (1984) documented how numerical control systems in manufacturing did not just make production faster, they fundamentally altered what it meant to be a machinist. Skilled craftsmen who understood materials and processes became machine operators who followed programmed instructions. The parallel is striking, today's developers risk transitioning from code craftsmen who understand logic and architecture to AI operators who orchestrate outputs they cannot fully explain. Acemoglu and Autor (2011) formalised this with their findings, seeing how technology replaced previously routine tasks and created new required skill for developers.

2. 7 AI Coding Tool Infrastructure

Hanseth and Lundberg (2001) argue that the infrastructure around specific tasks does not just help complete the tasks, but it fundamentally shapes what the task requires from the users. This insight proves crucial for AI coding tools. Organisations are not just adopting AI; they are building entire task infrastructures around it.

Google Cloud (2025) documents how leading organisations create comprehensive AI systems: prompt libraries that capture coding patterns, workflow templates that standardise AI interaction, and knowledge bases formatted for AI consumption. These are not just efficiency tools. Following Latour (1987), they function as "inscription devices “mechanisms that capture human knowledge and stabilise it in forms AI can reliably use.

The AI tool infrastructure emerging in recent studies exemplifies theoretical integration. Through ANT, infrastructure represents "inscription devices" (Latour, 1987) that stabilise new actor-networks and task completion processes. Organisations building proprietary prompt libraries, custom AI training, and specialised workflows gain exponentially more value than those using generic tools.

2.8 Conclusion

The integration of Actor-Network Theory with Affordance-Actualisation Theory provides a framework for understanding AI tools as active participants in development networks, whose impacts vary based on contextual factors. This perspective reveals that developers engage with AI through distinct patterns from comprehensive integration to selective use to minimal adoption rather than following universal trajectories. By combining ANT's recognition of AI agency with AAT's explanation of differential outcomes, we can investigate how developers are not just adopting tools but how they are co-evolving with AI in ways that create divergent professional futures. This framework will be used to guide our empirical investigation into how AI coding tools transform professional practice. This theoretical integration of Actor-Network Theory and Affordance-Actualisation Theory provides the conceptual foundation for investigating how developers navigate AI tool transformation.

Chapter 3: Methodology

3.1 Research Design and Philosophy

This study employed semi-structured interviews with 14 software developers to explore how AI coding tools transform professional practice. An interpretive approach was used in this research to understand how AI coding tools transform software developers' professional practice.

The research utilises Actor-Network Theory (Latour, 2005) and Affordance-Actualisation Theory (Strong et al., 2014), recognising technology adoption as socially constructed rather than technologically determined. ANT reveals how AI coding Tools become active network participants, while AAT explains why identical tools produce different outcomes. This dual-lens approach proved essential for understanding both transformation mechanics and variation.

3.2 Systematic Literature Search Strategy

This review employed Webster and Watson (2002) systematic search strategy to ensure broad coverage. My initial searches across ACM Digital Library, IEEE Xplore, AIS Electronic Library, Web of Science, ArXiv and Google Scholar using paired combinations like ("AI coding tools" or "GitHub Copilot" and "AI pair programming") and ("transformation" with "adoption" or "skills" or "identity") paired with ("developer" or "developer").

There was a focus on papers published between 2022-2025. AI coding tools only became fully relevant in the context of our research which mandated the cut off. Given the speed of progress in these tools, preprint studies that are highly cited and renowned are used as they are relevant to the specific circumstances I am researching.

Paper Inclusion Criteria:

- Published 2020-2025 (coinciding with AI coding tool emergence)
- Empirical evidence (not purely speculative)
- Focus on transformation/practice change (not just productivity)
- Peer-reviewed or highly cited preprints

3.3 Ethical Considerations

This research received LSE Research Ethics Approval (Reference number: 550382) in June 2025. Participants received detailed information sheets and provided written consent for recording and transcription (See Appendix A for Consent forms). Pseudonyms (P001-P014) replaced identities within 24 hours, with their organisation removed before analysis as well. All participants received detailed information sheets explaining the research purpose, data usage, and their rights and additional verbal approval to record and create a transcript was obtained at the beginning of each interview. All participant data will be deleted six months post-dissertation submission.

3.4 Participant Selection and Recruitment

Purposive sampling targeted maximum variation in AI coding Tool adoption approaches. I began by identifying potential participants through my professional network and LinkedIn, specifically seeking developers with diverse experiences with AI coding tools. My selection criteria evolved through initial conversations: I prioritised variation in company size, geographic location, years of experience, and, crucially, attitudes toward AI adoption, from enthusiastic early adopters to principled resisters. Inclusion criteria required minimum 3 years of development experience, or 2 years and a master's degree. The sample spanned six countries, company sizes from 15 to 10,000+ employees. There is likely to be bias towards AI tools as these participants are more likely to engage with research on AI coding tool use, but an effort was made to also include those open resistors.

ID	Descriptor	Role	Years Experience	Organization Type	Organization Size	Primary AI Tools	Interview Length	Location	Cluster
P001	"The Orchestrator"	Startup Developer	4	Tech Startup	Small (<50)	ChatGPT, Claude, GitHub Copilot	23 min	Germany	Complete Transformer
P002	"The Manager"	AI/ML Engineer	5	Healthcare Technology	Medium (50-250)	ChatGPT, Claude Code	32 min	Norway	Complete Transformer
P003	"The Enthusiast"	Enterprise Developer	3.5	Enterprise Software Company	Large (1000+)	Internal AI, Copilot, ChatGPT	35 min	Ireland	Complete Transformer
P004	"The Builder"	Data Analyst / Brand Specialist	3	Large E-commerce Company	Large (1000+)	Internal AI Tools, ChatGPT	34 min	Ireland	Complete Transformer
P005	"The Transcendent"	Full Stack Developer	8	Multiple/Freelance	Various	ChatGPT, Multiple Tools	38 min	Israel	Complete Transformer
P006	"The Guardian"	Full Stack Engineer	6	Banking/Financial Services	Large (1000+)	GitHub Copilot (restricted)	25 min	Ireland	Selective Adopter
P007	"The Boundary Setter"	Backend Developer	4.5	Financial Services (Payments)	Large (1000+)	GitHub Copilot, ChatGPT	29 min	UK	Selective Adopter
P008	"The Balanced"	Backend Developer	3	Aviation (Major Airline)	Large (1000+)	GitHub Copilot, ChatGPT	19 min	Ireland	Selective Adopter
P009	"The Observer"	Backend Engineer	5	Fintech	Medium (50-250)	ChatGPT, GitHub Copilot	31 min	Ireland	Selective Adopter
P010	"The Philosopher"	ML Engineer	6	Startup/Freelance	Small (<50)	ChatGPT, Claude	38 min	Ireland	Selective Adopter
P011	"The Craftsperson"	Junior iOS Developer	4	Aviation (Major Airline)	Large (1000+)	ChatGPT (limited use)	24 min	Ireland	Selective Adopter
P012	"The Strategist"	Product Owner	10+	Telecommunications	Large (1000+)	ChatGPT, Various	28 min	Ireland	Selective Adopter
P013	"The Purist"	Software Engineer	6	Industrial Software	Medium (50-250)	None (conscious choice)	42 min	UK	Conscious Resister
P014	"The Skeptic"	Data Scientist/Consultant	10	Large Consulting Firm	Large (1000+)	Watson (forced), minimal use	36 min	USA	Conscious Resister

Table 1: Participant Demographics Summary.

3.5 Data Collection

Semi-structured interviews (19-42 minutes, average 30.6) were conducted July-August 2025 via Microsoft Teams. The questions were designed to elicit narrative responses rather than simple yes/no answers, encouraging participants to share specific examples and reflect on their changing practices. Interview was designed to take 20-25 minutes per participant; I did have great interviews with some participants who were happy to discuss topics at great depth which is why some interviews stretch to 40+ minutes.

The protocol evolved through three iterations:

- **Version 1** (P001-P003): General AI usage exploration
- **Version 2** (P004-P009): Added infrastructure questions after P001's emphasis.
- **Version 3** (P010-P014): Incorporated identity and junior developer concerns

Questions covered the participants professional background, introduction to AI, use and changes to their role's tasks, new skills required for AI and future perspectives on development. My interview technique evolved from general exploration to targeted probing. Early interviews accepted productivity claims uncritically; in later interviews I explored costs and trade-offs. This evolution, while potentially introducing inconsistency, enabled richer understanding of transformation complexity. A copy of the interview guide can be found in the Appendix E.

3.6 Data Analysis Analytical Approach

I employed thematic analysis (Braun and Clarke, 2006) to identify, analyse, and track patterns within the interviews. This approach allowed me to move beyond surface descriptions to interpret the deeper meanings participants attached to their AI adoption experiences. Through iterative coding and theming processes, I developed a nuanced understanding of how developers navigate the transformation of their professional practice. There was only one researcher coding these interviews which does increase the risk of bias.

3.7 Analysis Software and Coding Process

Data management utilised NVivo 14 for systematic coding and retrieval. Interviews were transcribed using Otter.ai with manual verification for technical terminology. Analysis followed Braun and Clarke (2006) six-phase thematic analysis enhanced by theoretical

sensitivity from ANT and AAT frameworks. Initial coding produced 94 codes, refined through constant comparison to 56 consolidated codes organised into 12 categories.

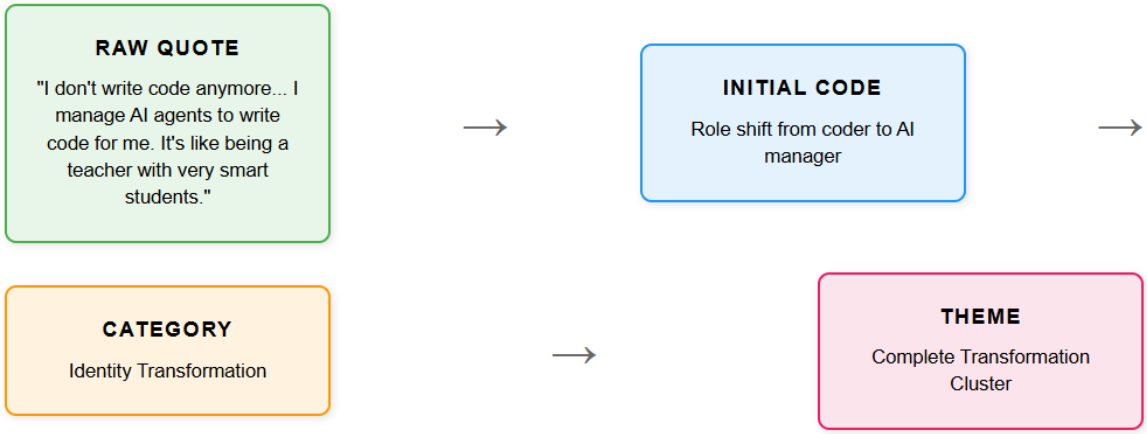

Figure 1: Coding Quotes into Themes: This shows of the coding process used in the thematic analysis. Raw interview quotes were systematically coded, collecting participant statements to initial descriptive codes, then grouped into conceptual categories, and finally organised into thematic clusters that show distinct patterns of AI adoption.

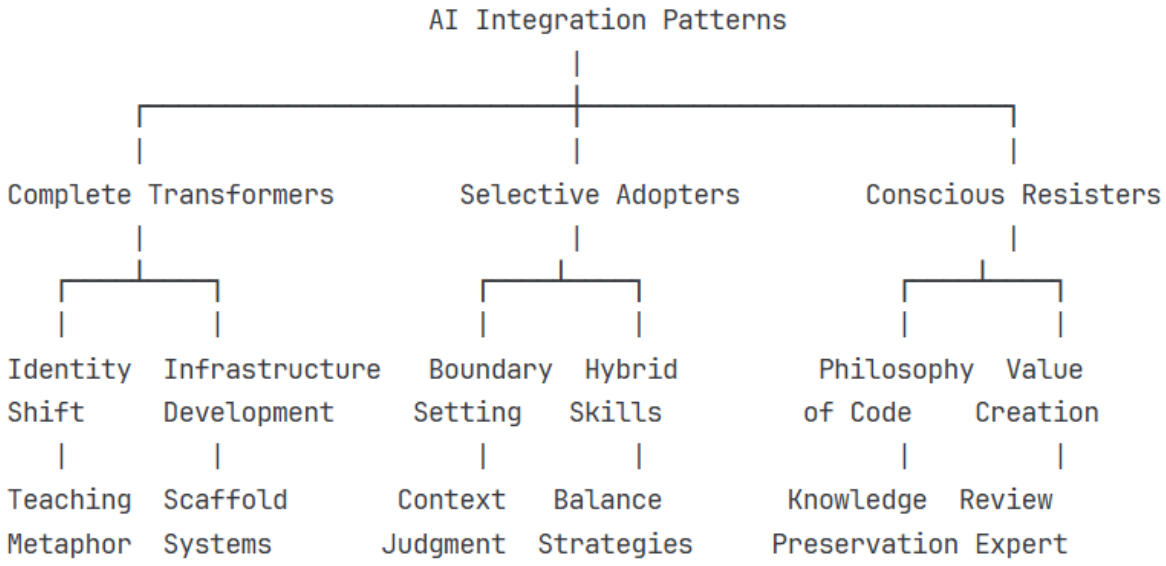

Figure 2: Thematic Analysis Coding Tree - How clustering functioned to create the three clusters.

3.8 Researcher Reflexivity

As someone who uses AI coding tools, I brought both insight and bias. Before the study I would have assumed:

- A high level of adoption of AI coding tools
- Most companies implement basic tools top down onto employees.

- Adoption patterns based on age.
- Universal productivity increases
- Skill loss and gradual separation of developers from the code
- Threats to junior roles and learning code via AI.

My personal relationship with AI coding tools evolved throughout this research. As an active GitHub Copilot user, I initially viewed AI coding tools through an optimisation lens, focusing on productivity gains.

However, three critical moments shifted my perspective: P003's observation about "automating away customers" challenged my productivity-focused assumptions; P008's journey from over-dependence to selective adoption revealed non-linear transformation paths; and P013's principled resistance reframed rejection as potentially strategic rather than regressive. This evolution influenced later interviews, where I probed more deeply into costs and trade-offs rather than accepting benefit claims uncritically.

3.9 Alternative Interpretations

The three clusters could reflect personality types, organisational cultures, or career stages rather than AI-specific transformation. The small sample prevents definitive causal claims. Economic pressures and job security concerns may influence participants' narratives about AI adoption.

Chapter 4: Findings

This chapter presents findings from thematic analysis of 14 semi-structured interviews with software professionals. Through the coding of 428 minutes of interview data three distinct clusters emerged, each representing fundamentally different approaches to integrating AI into professional practice.

4.1 Three Transformation Clusters

Analysis of these fourteen interviews revealed three patterns to AI coding Tool use among the participants. These clusters, Complete Transformers (n=5, 36%), Selective Adopters (n=7, 50%), and Conscious Resisters (n=2, 14%), represent the different level of tool integration into their jobs. The distribution of the participants indicates the adoption of AI Coding tools is not a universal experience and has many factors that influence it. The clusters are grouped based on their overall level of AI coding Tool adoption, but transformation is a spectrum with each participants transformation being unique. These clusters work as more of a general classification of developer rather than a firm white and black interpretation. These three patterns emerged from this sample and may not represent all developer experiences.

4.1.1 Complete Transformers: Developers Who Fully Embraced AI Coding Tools.

Five participants described a complete change in their development practices. This cluster included developers across startups to massive enterprises. Their experience ranged from 3-8 years, suggesting transformation isn't solely linked to career stage or experience.

Identity Transformation and Current Role Perception of Complete Transformers

Participants in this cluster expressed fundamental shifts in professional identity:

"I'm the teacher now. I spend my days teaching the AI what I want, guiding it through iterations. It's completely changed how I work; I don't write code anymore; I educate AI systems. I'm more professor than programmer now." (P001, Startup Developer)

"I'm managing AI students. I'll have one AI working on the API, another on the frontend, another writing tests, while I'm orchestrating and integrating. It's like running a development team where the developers never sleep, never complain, but also never truly understand." (P002, Healthcare ML Engineer)

P005 expressed the most radical identity shift: "I just get things done now. Don't call me a developer, I'm a solution creator. The code is just a byproduct of solving problems. My clients don't care if I wrote it or AI did, they care that it works."

Personal AI coding Tool infrastructure

AI tool infrastructure or infrastructure scaffolding is the creation of collection interconnected tools and processes that developers create to integrate multiple AI coding tools. Rather than using a single AI coding Tool in isolation, developers build comprehensive systems: GitHub Copilot for code generation, ChatGPT for architecture discussions, Claude for documentation, and specialised models for testing - all coordinated through custom workflows. This infrastructure includes version-controlled prompt templates, automated validation pipelines that check AI-generated code, knowledge bases that preserve successful patterns, and feedback loops that continuously improve their AI coding tools performance.

All Complete Transformers described building AI infrastructure to support their transformed practices. P001 detailed "50+ markdown files containing organisational memory, best practices, error patterns, success templates," emphasising that "this infrastructure is what makes me 5x more productive than before." These files were organised hierarchically: company policies at the top, project-specific patterns in the middle, and daily prompts at the bottom.

P002 reported requiring "three months of healthcare-specific implementation" to address domain requirements before achieving productivity gains. This investment included creating specialised prompts for HIPAA compliance, medical terminology databases, and edge-case documentation specific to patient data handling.

P004's organisation "built internal AI coding tools trained on years of documentation," creating what they described as "having a senior developer who knows everything about our systems, every API endpoint, every database schema, every business rule from the past decade."

Despite embracing transformation, participants expressed systemic concerns:

"We're seeing 5x productivity gains, easily. But here is the thing, our product helps businesses reduce headcount. We are automating away our own customers. What happens when every company needs 80% fewer developers?" (P003, Enterprise Developer)

4.1.2 Selective Adopters: Adopts AI Coding Tools but Uses Them Selectively

Seven participants described maintaining deliberate boundaries while integrating AI coding tools. This cluster represented the largest group, spanning banking, aviation, fintech, and telecommunications sectors. Their experience ranged from 3-10 years, with regulatory constraints often shaping boundary decisions.

Common AI Boundaries and Tool shortcomings

Participants consistently emphasised contextual decision-making:

"It's a tool, not a rule. I decide when and how to use it based on what is appropriate. Critical path code, authentication, payment processing, data handling, which is all me. AI handles the scaffolding around it." (P006, Banking Engineer)

P007 highlighted regulatory constraints: "Banking regulations mean some things must remain under human control. I delegate UI components but never core transaction logic. Every line of code touching money has my fingerprints on it."

These boundaries were not arbitrary but reflected what Schön (1983) might term "reflection-in-action", professional judgment developed through experience about where AI assistance helps versus hinders.

Learning Through Experience

P008's journey describes how his role has changed through task automation and code generation.

"I went too deep with AI at first, lost touch with code, then pulled back. Now I know the balance. I got to a point where I was going to write some unit tests and thought 'I shouldn't need to ask AI how to do this.' That was my wake-up call."

This participant's described how AI lacks context on legacy system: "New features get maybe 50% checking. Anything touching existing systems gets line-by-line review. Legacy code has decades of undocumented decisions, AI can't understand what nobody wrote down."

P009 observed community fragmentation: "The developer community is splitting. You have AI evangelists who have drunk the Kool-Aid, balanced users trying to find middle ground, and traditionalists clinging to the old ways. We're losing our shared language."

Skill Preservation Strategies Used by Developers.

P011 implemented structured preservation: "Every Friday is no-AI day. I code everything manually to keep skills sharp. A master carpenter uses power tools but remains a carpenter. I use AI coding tools but remain a developer."

P012 used AI for specific product management tasks: "User stories, acceptance criteria, test scenarios, AI excels at structure. But understanding what stakeholders need versus what they say? That's irreducibly human."

4.1.3 Conscious Resisters: Knowledge Preservation and AI Rejection

Two participants actively resisted AI integration while acknowledging its existence. Both had 6-10 years' experience and worked in contexts where deep understanding and code complexity is more important compared to traditional full stack development.

Philosophical Stance

P013 articulated principled opposition:

"If you're using AI to do the work, you're not learning. You are just getting output. Someone needs to preserve real understanding. While others become AI operators, I maintain true developer knowledge."

When pressed, P013 admitted minimal adoption: "For tests, okay, it's repetitive enough that AI makes sense. But if I must use it for a demo or something, I rewrite everything it produces to ensure I understand every line."

Unexpected Value Creation

P014, despite forced compliance with company requirements, discovered unique positioning: "I've become the go-to person for reviewing AI-generated code. Ironically, resisting AI has made me more valuable because I can spot what others miss, the subtle bugs, the security holes, the architectural anti-patterns that AI perpetuates."

4.2 Infrastructure Development Patterns

The research revealed how the studies participants across all clusters developed their AI coding tool infrastructure, though the nature and purpose of these infrastructures varied significantly on the tasks and role they have as a developer.

Infrastructure, as revealed in this study, represents organisational systems that amplify AI coding tool capabilities through recursive knowledge accumulation. This finding extends beyond individual tool use, revealing how organisations build systematic capabilities that make AI coding tools exponentially more valuable than generic AI infrastructure. Across all clusters, participants developed systematic capabilities representing what Star and Griesemer (1989) term "boundary objects", artifacts coordinating between human and AI actors. These infrastructures varied in scope and purpose but universally represented considerable time investment for long term productivity increases.

4.2.1 Comprehensive Developer Infrastructure – Full Tool Integration

Complete Transformers commonly built comprehensive systems with organised prompt libraries. They invested time developing domain-specific prompts for specialised tasks they can use to quickly tackle common tasks. Their infrastructure is integrated access to their organisation’s knowledge bases, connecting their tools to internal wikis, documentation, and historical code.

4.2.2 Selective AI Infrastructure – Limited Tool Integration

Selective Adopters created systems featuring more extensive validation protocols with specific checklists for AI-generated code. They developed legacy system integration guidelines defining rules for when AI could modify existing systems. Their infrastructure was limited and only focused on specific tasks they felt were beneficial but kept certain tasks entirely in their own purview. AI delegation decisions, team collaboration frameworks establishing protocols for AI-assisted pair programming, and quality assurance workflows implementing multi-stage review processes for AI contributions.

4.3 Task Transformation Patterns

Analysis revealed both universal patterns and cluster-specific variations in task delegation, suggesting certain tasks have inherent characteristics making them suitable or unsuitable for AI assistance. These patterns align with Actor-Network Theory's concept of delegation, where certain activities are redistributed within the human-AI network based on their characteristics and the developer’s willingness to delegate that task to AI coding tools.

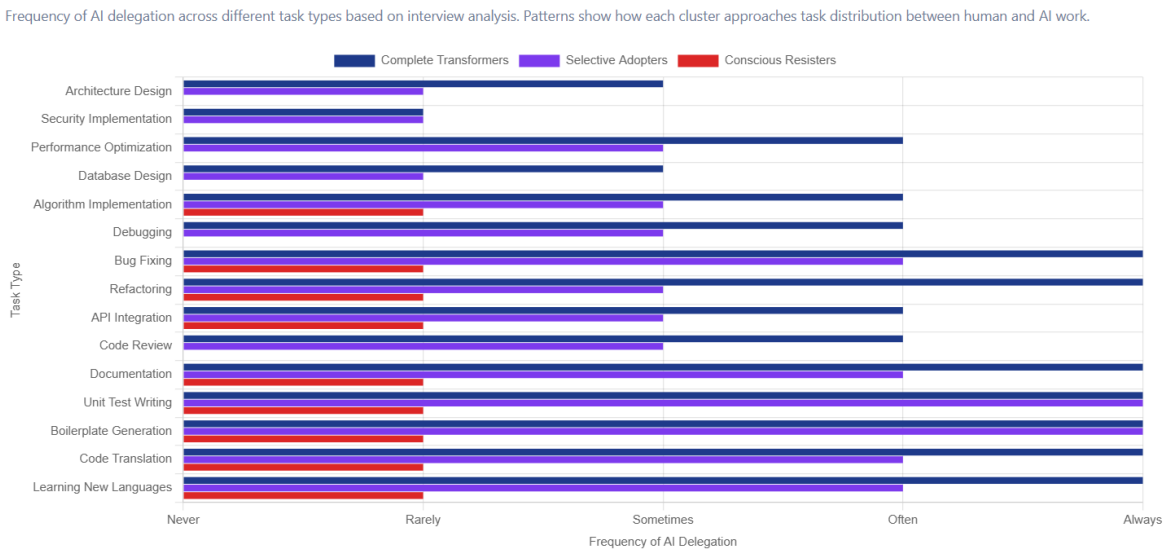

Figure 3: Illustrates these delegation patterns across clusters, revealing both convergent and divergent approach.

Certain tasks showed consistent delegation across clusters:

Test writing (12/14 participants): "Deterministic input-output mappings" Documentation (11/14 participants): "Explaining what exists" Boilerplate code (10/14 participants): "Patterns repeated thousands of times" Data validation (9/14 participants): "Rule-based transformations" API integration (8/14 participants): "Following documented patterns”?

P006 explained the consensus: "Tests are perfect for AI, repetitive, pattern-based, clear success criteria. Even if it gets edge cases wrong, it's faster to fix than write from scratch."

4.3.2 Persistent Human Tasks Across Clusters

Conversely, specific tasks remained human-controlled:

Architecture design (0/14 delegated): "Requires understanding unstated requirements" Security implementation (1/14 delegated): "Too high-risk for probabilistic solutions" Core business logic (1/14 delegated): "Embeds crucial domain knowledge" Performance optimisation (2/14 delegated): "Requires deep system understanding" User experience decisions (0/14 delegated): "Needs human empathy and context"

P003 articulated shared reasoning: "AI can suggest patterns, but architecture requires understanding business context, future scaling, team capabilities, things that exist in people's heads, not documentation."

4.3.3 How Do They Validate and Review AI Generated Code.

Validation approaches varied systematically by cluster. Complete Transformers reported minimal validation (20-30%), with P001 explaining: "Once you've trained the AI properly with your infrastructure, it knows your patterns. I spot-check for logical coherence, not syntax."

Selective Adopters validated contextually (50-80%), while Conscious Resisters maintained complete validation. P013 noted: "Seeing AI's mistakes actually made me better at understanding why certain patterns work. It's like teaching, you learn by correcting errors."

4.3.4 Vibe Coding: A Shift in Programming Practice and Code Generation

An unexpected finding among Complete Transformers was "vibe coding", an intuitive, iterative approach to AI-assisted development representing a fundamental shift in programming epistemology.

P001 articulated this practice: "I iterate with the AI until the solution feels right. It is more art than science now. I can't always explain why something is correct, but I know it when I see it."

This approach challenges traditional programming's emphasis on logical reasoning and explicit understanding. Vibe coding operates through pattern recognition and intuitive validation rather than formal verification. P002 noted: "It's like jazz improvisation, you have a general direction but you're responding to what the AI gives you, building on it, redirecting when needed."

Figure 4: Synthesis of these workflow transformations, demonstrating how each cluster has reconstructed their development practice.

4.4 Emerging Skills in Software Development

Participants reported simultaneous skill acquisition and atrophy patterns varying by cluster, suggesting transformation involves skill substitution rather than simple enhancement. This finding resonates with Affordance-Actualisation Theory's emphasis on how technology affordances reshape capabilities, the same AI coding tools afford different skill developments based on how actors actualise them within their practice.

Each cluster developed distinct new competencies reflecting their transformation approach:

Complete Transformers acquired advanced prompt engineering capabilities for building conversational architectures, multi-agent orchestration skills for managing parallel AI processes, infrastructure design expertise for creating scalable knowledge systems, and rapid technology adoption abilities for learning new frameworks through AI.

Selective Adopters developed boundary judgment skills for knowing when AI helps versus hinders, hybrid workflow management capabilities for seamlessly switching modes, context-specific validation expertise for risk-adjusted review strategies, and collaborative prompting abilities for getting AI to explain its reasoning.

Conscious Resisters enhanced AI error detection capabilities for spotting characteristic AI mistakes, traditional debugging mastery for understanding without assistance, deep

architectural understanding for seeing system-wide implications, and knowledge preservation skills for documenting the undocumented.

4.4.2 Traditional Skill Trajectories

Self-reported skill retention varied dramatically, with participants showing awareness of trade-offs. Complete Transformers acknowledged significant atrophy, with P001 admitting: "I probably couldn't write a complex sorting algorithm from scratch anymore. But I can architect systems 10x more complex than before. It's a different kind of capability."

4.5 The Junior Developer Crisis: Consistent Concern Among Participants

All participants, regardless of cluster affiliation, expressed concern about junior developer skill development - the only finding achieving complete consensus across all interviews. This aligns with industry observations about the "70% problem" where AI tools enable rapid initial progress but struggle with the complex final stages of development (Osmani, 2024).

4.5.1Traditional Learning Disruption

P002 discussed their fears about junior engineer education: "Where will juniors learn debugging when AI fixes errors instantly? How will they develop intuition without struggling through problems? The struggle is where learning happens."

P006 emphasised practical implications: "When production breaks at 3 AM, you need understanding, not just AI assistance. Where will juniors get that experience if they have never debugged without help?"

This concern reflects broader industry patterns where AI tools may be creating what Osmani (2025) terms a generation gap between those who learned programming fundamentals before AI and those who are learning with AI from the beginning.

4.5.2 Organisational and Economic Impacts

P003 noted structural impacts: "Junior developers with AI can produce what previously would have needed senior developers before. It is disrupting our whole team structure and salary bands. Why pay senior rates for AI-augmented juniors?"

P014 identified mentorship breakdown: "Traditional mentorship is breaking down. Juniors ask AI before asking seniors. We're losing knowledge transfer mechanisms that built this profession."

4.6 Professional Futures: Divergent Visions on the future of developers

Participants' future projections reflected their cluster positioning and transformation experiences. These visions revealed deep uncertainty about professional trajectories, with each cluster imagining different endpoints.

P001 predicted radical discontinuity: "In five years, 'developer' will sound like 'typist', a quaint historical role. We'll be AI conductors, not code writers."

P006 envisioned sustained hybridity: "There will always be need for humans who understand both traditional development and AI capabilities. The boundary managers will become the most valuable."

P013 anticipated cyclical return: "When AI-generated technical debt becomes crushing, organisations will desperately seek developers who actually understand code. Mark my words, traditional skills will command premium prices."

4.7 Summary

These findings reveal AI coding Tool integration as producing three distinct, stable configurations rather than universal transformation. Infrastructure development emerged as critical differentiator across all clusters, with each group building different systems to support their transformation approach. While task delegation showed both convergent patterns (tests, documentation) and persistent human domains (architecture, security), these patterns reflected deeper epistemological shifts in how developers conceptualise their work.

The emergence of concepts like "vibe coding" and infrastructure scaffolding suggests AI coding tools are not merely augmenting existing practices but creating fundamentally new forms of development practice. The concerns expressed by all participants about junior pipelines suggests this area warrants attention on the challenges emerge through the implementation of these tools.

The diversity of responses, from Complete Transformers' radical reconstruction to Conscious Resisters' principled preservation, indicates that professional practice transformation follows multiple pathways rather than converging toward a single future state. These findings contribute to understanding technology-mediated professional transformation by revealing how identical tools can produce divergent outcomes based on how practitioners choose to integrate them into their professional identity and practice.

Chapter 5: Discussion

This chapter interprets the findings through integrated theoretical lenses ANT and AAT, examines how AI coding tools transform software development and discusses the three-cluster framework that emerged from the analysis.

5.1 Addressing the Research Objectives

Having demonstrated how the research objectives have been addressed through empirical investigation, these findings now enable significant theoretical contributions that extend beyond the immediate context of software development.

5.1.1 Task Allocation and Agency Distribution (Objective 1)

The first objective examined how AI coding tools reconfigure task allocation, these findings reveal complex patterns of task reallocation that extend beyond simple delegation. Universal delegation of testing (86%), documentation (79%), and boilerplate code (71%) represents what Latour (1986) terms "obligatory passage points", tasks where AI achieves network enrolment across all configurations. This contrasts sharply with Peng et al. (2023) productivity focus, revealing transformation as fundamental practice restructuring rather than efficiency enhancement.

Complete Transformers' parallel development model, orchestrating multiple AI agents simultaneously, represents unprecedented agency distribution. When P002 describes "managing AI students," this is not metaphorical but reflects genuine agency redistribution where AI actors make autonomous decisions within bounded contexts. This empirical finding extends ANT beyond Callon (1986) human-centric translation model, suggesting AI achieves what I term "bounded actor status", autonomous within defined parameters while remaining subordinate to human orchestration.

Conversely, universal retention of architecture design (100%), security implementation (93%), and core business logic (93%) challenges technological determinism prevalent in current literature. These boundaries persist not from technical limitations but from what participants identified as irreducibly human competencies: contextual understanding, ethical judgment, and tacit knowledge integration (Schön, 1983).

5.1.2 Emergent Competencies and Skill Transformation (Objective 2)

The second objective identified emergent competencies, in the findings we saw skill evolution patterns directly contradict linear upskilling narratives dominating practitioner discourse (Brynjolfsson et al., 2023). Infrastructure scaffolding emerges as the critical meta-competency, not individual prompt engineering but organisational capability building. This finding aligns with Hanseth and Lundberg (2001) work-oriented infrastructures while extending it to human-AI contexts.

Complete Transformers' skill profile presents a paradox unaddressed in literature: traditional skill atrophy (manual coding 45% retained) coupled with claimed enhanced capabilities ("architect systems 10x more complex"). This is not simple substitution but fundamental competency restructuring. Their "vibe coding" practice, iterating with AI until solutions "feel right", represents new epistemic practice where validation shifts from logical verification to intuitive assessment.

This phenomenon of vibe coding deserves deeper theoretical exploration. It represents what Polanyi (1966) termed "tacit knowledge" but inverted, rather than explicit knowledge becoming tacit through practice, AI enables tacit pattern recognition to replace explicit reasoning. Participants describe "knowing" when code is correct without being able to articulate why, suggesting AI coding tools enable a form of intuitive programming that bypasses traditional logical scaffolding. This epistemic shift has profound implications for how we conceptualise programming expertise, moving from rule-based reasoning to pattern-based intuition mediated by AI collaboration.

Selective Adopters' balanced approach (75% traditional skill retention) appears optimal but carries hidden costs. P011's "no-AI Fridays" reveals the exhausting vigilance required to maintain dual competencies.

5.1.3 Professional Identity Reconstruction (Objective 3)

The third objective explored professional identity reconstruction, the identity transformation patterns reveal deeper complexity than Wenger (1998) community of practice framework suggests. Rather than shared practice evolution, we observe community fragmentation into distinct professional subspecies. Complete Transformers' educational metaphors ("teacher," "professor," "orchestrator") represent not role expansion but fundamental identity replacement.

The universal junior developer concern transcends individual transformation, revealing collective anxiety about professional reproduction. This finding extends beyond previous technological transitions (such as the shift from manual coding to IDEs or the adoption of Stack Overflow) because AI disrupts the apprenticeship model itself. When P009 observes "juniors ask AI before asking seniors," this represents breakdown of knowledge transfer mechanisms essential for community sustainability.

Conscious Resisters' identity as knowledge guardians provides unexpected insight. Their value increases precisely because others abandon traditional skills, creating what labour economists' call "skill complementarity" (Acemoglu and Autor, 2011), their expertise becomes more valuable as it becomes rarer. This challenges success-biased adoption literature by revealing resistance as potentially rational career strategy.

5.2 Theoretical Contributions

5.2.1 AI coding tool Infrastructure

Infrastructure extends beyond individual tool use to organisational capability building. These finding challenges individual-focused adoption models (TAM, UTAUT) by revealing competitive advantage arising from systematic amplification systems rather than individual proficiency (Venkatesh et al., 2003).

However, infrastructure creates what I term "capability lock-in". The same systems enabling productivity may constrain future flexibility. P001's 50+ markdown files require constant maintenance; P004's enterprise integrations create vendor dependencies. This dark side of infrastructure investment remains unexplored in enthusiastic practitioner accounts.

Infrastructure emerges as potential differentiator in successful AI coding tool use, yet this finding carries troubling implications. As organisations build proprietary AI amplification systems, they create new forms of technical debt and vendor lock-in. The same infrastructure enabling today's productivity gains may become tomorrow's legacy burden, a possibility developer has not fully considered in their transformation.

5.2.2 The Network-Affordance Integration Model

This study's primary theoretical contribution synthesises ANT and AAT to explain differential transformation outcomes. Rather than treating network position and affordance perception as independent variables, common in existing IS literature. Typically, this model reveals their

recursive relationship. Network position shapes which affordances actors can perceive (Complete Transformers' reconstructed networks enable seeing AI as collaborative partners), while affordance actualisation patterns recursively reshape networks (infrastructure investments lock in transformation pathways).

P008's journey empirically demonstrates this recursion: enthusiastic adoption → over-dependence → network degradation → affordance re-evaluation → selective adoption. This movement between clusters reveals transformation as dynamic process rather than stable state, suggesting clusters represent temporary equilibria subject to disruption.

The model addresses a critical gap in Strong et al. (2014) framework, which assumes successful actualisation. By documenting failed actualisations and conscious non-actualisation, this study reveals the complete actualisation spectrum, including resistance as legitimate outcome rather than failure.

Figure 5.1: Network-Affordance Integration Model

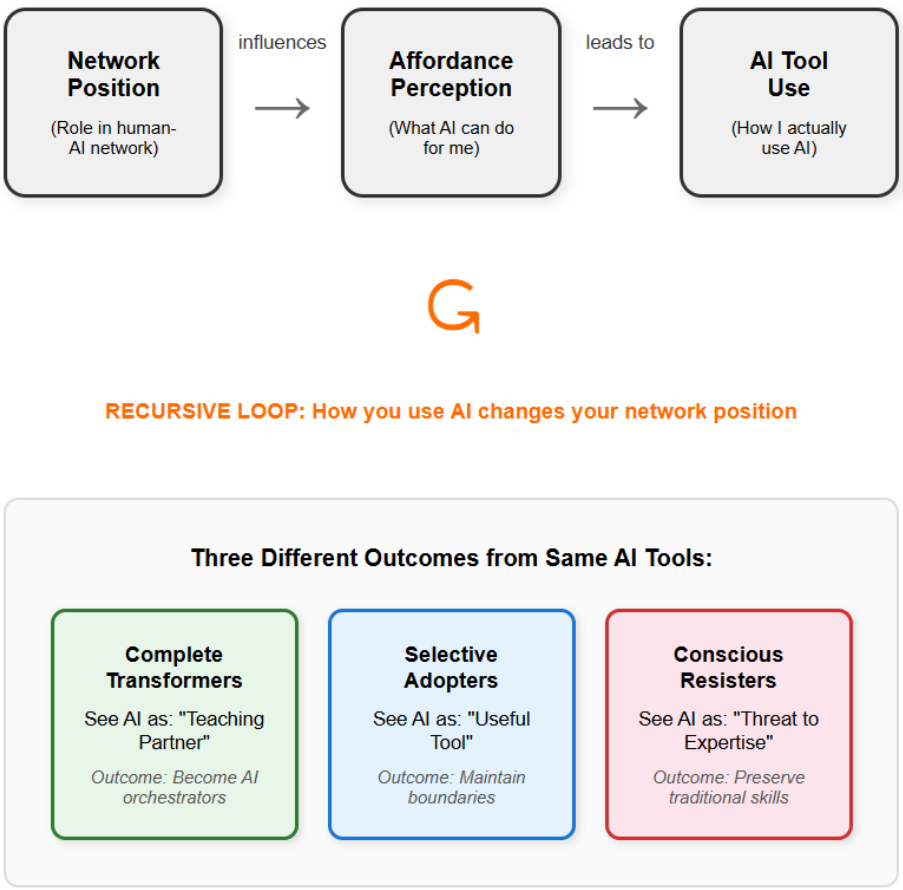

Figure 5: Illustrates this recursive relationship between network position and affordance perception.

5.2.3 Three-Cluster Framework

Rather than adoption stages, the three clusters that emerged from the research represent configurations with distinct advantages:

- Complete Transformers: Maximum productivity at dependency cost.
- Selective Adopters: Sustainable balance through specific task selection.
- Conscious Resisters: Protecting Code Output from AI mistakes.

This framework challenges stage models' teleological assumption of convergence toward universal adoption. Instead, ecosystem diversity may prove essential for professional resilience, with each cluster serving distinct market needs (March 1991).

5.3.4 Transformation Beyond Just Productivity

Peng et al. (2023) 55.8% productivity gain narrative obscures transformation complexity revealed here. Productivity metrics fail to capture identity reconstruction, skill atrophy, or infrastructure investment costs. More critically, they assume productivity equals value. P003's reflection on "automating away our own customers" reveals productivity's potential self-destructiveness.

The "70% problem" documented by Osmani (2024) illustrates this complexity. While AI tools dramatically accelerate initial development phases, they struggle with the nuanced final stages that require deep understanding. This creates a false sense of productivity that may mask growing technical debt and reduced comprehension.

Unlike studies celebrating AI adoption (Bird et al., 2022; Brynjolfsson et al., 2023), this research documents resistance as rational strategy. Conscious Resisters are not technophobes but strategic actors positioning for future value when deep understanding becomes scarce.

5.4 Practical Implications

5.4.1 Organisational Strategies from These Findings

This research indicated that organisations might benefit from considering abandon convergence assumptions and support cluster diversity:

For Complete Transformers: Invest in infrastructure while maintaining skill redundancy. Create "break-glass" protocols for when AI systems fail. Document tacit knowledge before it disappears.

For Selective Adopters: Provide boundary clarification tools and decision frameworks. Recognise the exhausting nature of constant boundary negotiation. Create spaces for reflection and adjustment.

For Conscious Resisters: Value their role as quality guardians and error detectors. Position them in code review and architectural oversight roles. Preserve their expertise through formal knowledge transfer programs.

5.4.2 Addressing the Junior Developer Crisis

The universal concern about junior pipelines suggests areas for consideration:

1. Redesign Learning Pathways: Create structured struggles that develop debugging intuition.
2. Hybrid Mentorship Models: Combine AI assistance with human guidance.
3. Competency Frameworks: Redefine what constitutes junior developer expertise in AI age.
4. Protected Learning Spaces: Establish "AI-free" zones for fundamental skill development.

5.4.3 Individual Developer Strategies

Developers might consider conscious cluster choices based on career goals and risk tolerance:

- Choose Complete Transformation for maximum short-term productivity accepting obsolescence risk.
- Choose Selective Adoption for sustainable balance accepting constant negotiation burden.
- Choose Conscious Resistance for deep expertise betting on future scarcity value.

Movement between clusters remains possible and encouraged, developers should view the current position as a strategic choice rather than permanent identity.

5.5 Implications of Advancing AI Capabilities

As AI coding tools improve, these patterns may evolve along with the technology. Complete Transformers might become more dependent or find new boundaries. Selective Adopters may need to continuously readjust their boundaries. Conscious Resisters might find their expertise

valuable but also could be outpaced by people using AI coding tools. New patterns might emerge beyond these three clusters This study captures a specific moment; longitudinal research is needed to track pattern stability. The conclusion now synthesizes these insights to address the central research question and chart directions for future investigation.

Chapter 6: Conclusion

6.1 Answering the Research Question

This dissertation began by questioning whether developers are simply adopting new tools or experiencing fundamental professional reconstruction. The evidence clearly supports the latter, revealing transformation as ecosystem diversification rather than convergent adoption.

The 14 developers in this study demonstrated three different approaches: Complete Transformers who reconstruct their identity around AI orchestration, Selective Adopters who maintain strategic boundaries through constant negotiation, and Conscious Resisters who preserve traditional expertise as future competitive advantage. This is not linear progression it's ecosystem diversification, with each approach representing a viable response to AI disruption.

Beyond individual transformation lies a collective crisis: the breakdown of professional reproduction. Every participant, regardless of cluster, expressed concern about junior developers. When AI automates entry-level tasks, traditional learning through struggle disappears, threatening the tacit knowledge development essential for expertise.

6.2 Theoretical Contributions

The Network-Affordance Integration Model developed through this research advances IS theory by revealing transformation as recursive causality. Network position shapes what developers see as possible with AI, while their choices reshape their position creating divergent evolution rather than convergent adoption.

ANT revealed how AI becomes an active participant reshaping development network, when developers describe "managing AI students," they recognise AI's actor status. AAT explained why identical tools produce three different patterns based on organisational context and individual perception. Together, they show transformation emerging from network-affordance configurations, not individual choice, or technological determinism.

Infrastructure scaffolding was consistent throughout the interviews. Complete Transformers' elaborate systems of prompt libraries and documentation create exponential value, while Selective Adopters' boundary protocols enable sustainable practice. Yet this infrastructure creates its own trap. Today’s competitive advantage becomes tomorrow's technical debt.

The emergence of 'vibe coding' where developers iterate until solutions 'feel right' without understanding why this represents a fundamental shift from logical to intuitive programming, raising questions about the nature of programming expertise. Increased code will be produced without the creator understanding how it works, we could see this trend continue as AI tools become so good that human coders are no longer competitive.

Contribution Type	Key Finding	Challenges to Existing Literature
Theoretical	**Network-Affordance Integration Model:** Reveals recursive relationship between network position and affordance perception, explaining divergent transformation outcomes	Linear adoption models (TAM, UTAUT) that assume stable technology boundaries and universal progression
Empirical	**Three Stable Clusters:** Complete Transformers (36%), Selective Adopters (50%), and Conscious Resisters (14%) represent distinct configurations, not adoption stages	Success bias in AI literature and assumptions of inevitable convergence toward universal adoption
Conceptual	**Infrastructure Scaffolding:** Organisational AI amplification systems (prompt libraries, workflows) emerge as critical differentiator in transformation success	Individual-focused adoption frameworks that overlook organisational capability building
Practical	**Junior Developer Crisis:** Universal concern across all clusters about skill development pipeline breakdown as AI automates entry-level tasks	Productivity-only metrics (e.g., Peng et al.'s 55.8%) that ignore professional reproduction and knowledge transfer
Methodological	**Resistance as Strategy:** Documents conscious non-adoption as rational career positioning for future scarcity value, not technophobia	Binary adoption/rejection frameworks that frame resistance as failure rather than strategic choice
Epistemological	**"Vibe Coding" Practice:** Iterative, intuitive AI collaboration represents shift from deterministic to pattern-based programming validation	Traditional conceptualisations of programming expertise based on logical reasoning and explicit understanding

Table 2: Research Contributions Summary

6.3 Practical Implications

For Organisations: Competitive advantage may come from supporting diversity, not forcing convergence. Innovation benefits from Complete Transformers, stability emerges from Selective Adopters, quality assurance strengthens through Conscious Resisters. Organisations may benefit from all three operating in productive tension.

Specific practices illustrate these differences. Complete Transformers validate only 20-30% of AI output, trusting their infrastructure to ensure quality. Selective Adopters validate 50-80% based on context, with some implementing 'no-AI Fridays' to preserve skills. Conscious

Resisters maintain 100% validation, catching subtle errors others miss. Notably, all clusters delegate testing to AI but preserve architecture design for humans, revealing shared boundaries even amid divergence.

For Education: Traditional mentorship models assuming gradual skill building face disruption. Institutions might consider developing parallel pathways:

- AI-native curricula for orchestration skills
- Hybrid programs balancing traditional and AI-augmented abilities.
- Preservation tracks maintaining deep technical expertise.

For Individuals: Career choices may involve conscious trade-off evaluation:

- Complete Transformation: Maximum productivity, maximum dependency
- Selective Adoption: Sustainable balance, constant boundary work
- Conscious Resistance: Deep expertise, betting on scarcity value

6.4 Limitations of the Study

The underrepresentation of Conscious Resisters (n=2) particularly limits claim about resistance patterns. Analytical Constraints Single-researcher coding, while systematic it lacks inter-rater reliability validation. The exclusive reliance on self-reported data without behavioural observation means findings capture perceptions rather than verified practices. The three clusters represent analytical patterns in this data, not definitive categories.

Contextual and Temporal Boundaries Findings reflect Western, English-speaking contexts and may not apply to other cultural or regulatory environments. The cross-sectional design during rapid AI evolution means these patterns may already be shifting. What appears stable in 2025 may prove transitional as AI capabilities advance.

6.5 Future Research Imperatives

Longitudinal Studies: Track cluster stability over 2-3 years. Do Complete Transformers sustain enthusiasm as complexity grows?

Cross-Cultural Analysis: How do collectivist cultures shape patterns? How do regulations influence boundaries?

Junior Developer Focus: How can the profession transfer knowledge without traditional pathways? What defines expertise in AI-mediated contexts? Can we design learning that develops both AI fluency and fundamental understanding?

AI Evolution: As AI capabilities advance, these patterns may intensify rather than converge. Better AI could make Complete Transformers more dependent, force Selective Adopters to constantly readjust boundaries, and either vindicate or make Conscious Resisters obsolete. The clusters might diverge further rather than merge, creating even more specialised professional subspecies.

6.6 Broader Significance

Software development's transformation previews knowledge work's AI-mediated future. As legal research, medical diagnosis, and engineering design face similar disruptions, our findings offer crucial insights. The three-cluster framework demonstrates that professions need not converge on single practices, showing diversity enables resilience.

This challenges techno-solutionist narratives. Productivity gains carry hidden costs: skill atrophy, infrastructure dependencies, reproduction crises. Most fundamentally, professional identity suggest that it is more fluid than acknowledged. Participants did not just adopt tools; they reconstructed what it means to be a developer. From "code writers" to "AI teachers," from "developers" to "orchestrators," these shifts represent profound reformation requiring support as AI transforms knowledge work.

6.7 Final Reflections

These findings suggest the profession might benefit from maintaining this ecosystem diversity, not identifying a "winning" approach. Wisdom lies not in predicting the dominance of one approach but accepting the diversity. At this inflection point, the choice is not whether to embrace or resist AI, but how to consciously shape your personal professional practice in software developement. Their divergent paths illuminate not just software development's future, but the transformation awaiting knowledge work in the AI era. The developers in this study, if orchestrating these AI symphonies, negotiating careful boundaries, or preserving traditional craft they are engaged in the essential work of professional reformation.

References

Acemoglu, D., & Autor, D. (2011). Skills, tasks and technologies: Implications for employment and earnings. In O. Ashenfelter & D. Card (Eds.), *Handbook of labor economics* (Vol. 4, pp. 1043-1171). Elsevier. https://doi.org/10.1016/S0169-7218(11)02410-5

Ammar, A., Corda, V., Prisco, G., Foggia, M., Erra, U., & Colonnese, S. (2024). From today's code to tomorrow's symphony: The AI transformation of developer's routine by 2030. *arXiv preprint arXiv:2405.12731*. https://arxiv.org/html/2405.12731v1

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, *7*(OOPSLA1), 85-111. https://doi.org/10.1145/3586030

Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). Taking flight with copilot: Early insights and opportunities of AI-powered pair-programming tools. *ACM Queue*, *20*(6), 35-57. https://doi.org/10.1145/3589996

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77-101. https://doi.org/10.1191/1478088706qp063oa

Braverman, H. (1974). *Labor and monopoly capital: The degradation of work in the twentieth century*. Monthly Review Press.

Brynjolfsson, E., Li, D., & Raymond, L. R. (2023). *Generative AI at work* (NBER Working Paper No. 31161). National Bureau of Economic Research. https://doi.org/10.3386/w31161

Callon, M. (1986). The sociology of an actor-network: The case of the electric vehicle. In M. Callon, J. Law, & A. Rip (Eds.), *Mapping the dynamics of science and technology* (pp. 19-34). Macmillan.

Dahri, N. A., Yahaya, N., Al-Rahmi, W. M., Aldraiweesh, A., Alturki, U., Almutairy, S., Shutaleva, A., & Soomro, R. B. (2024). Extended TAM based acceptance of AI-Powered ChatGPT for supporting metacognitive self-regulated learning in education: A mixed-methods study. *Heliyon*, *10*(8), e29317. https://doi.org/10.1016/j.heliyon.2024.e29317

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319-340. https://doi.org/10.2307/249008

Engineering Enablement. (2025, April 2). 5 strategies for mentoring junior developers in the AI era [Blog post]. https://engineeringenablement.substack.com/p/5-strategies-for-mentoring-junior

Gibson, J. J. (1979). *The ecological approach to visual perception*. Houghton Mifflin.

GitHub. (2024). *Survey: The AI wave continues to grow on software development teams*. https://github.blog/news-insights/research/survey-ai-wave-grows/

Goel, N. (2025, February 14). New junior developers can't actually code [Blog post]. https://nmn.gl/blog/ai-and-learning

Gomez, C., Cho, S. M., Ke, S., Huang, C. M., & Unberath, M. (2024). Human-AI collaboration is not very collaborative yet: A taxonomy of interaction patterns in AI-assisted decision making from a systematic review. *Frontiers in Computer Science*, *6*, Article 1521066. https://doi.org/10.3389/fcomp.2024.1521066

Google Cloud. (2025). *Real-world gen AI use cases from the world's leading organizations*. https://cloud.google.com/transform/101-real-world-generative-ai-use-cases-from-industry-leaders

Gross, G. (2025, May 22). AI coding assistants wave goodbye to junior developers. *CIO*. https://www.cio.com/article/3509174/ai-coding-assistants-wave-goodbye-to-junior-developers.html

Hanseth, O., & Lundberg, N. (2001). Designing work-oriented infrastructures. *Computer Supported Cooperative Work*, *10*(3), 347-372. https://doi.org/10.1023/A:1011290912635

Holden, R. J., & Karsh, B. T. (2019). Beyond TAM and UTAUT: Future directions for HIT implementation research. *Journal of Biomedical Informatics*, *100*, 103315. https://doi.org/10.1016/j.jbi.2019.103315

JetBrains. (2024). *The State of Developer Ecosystem 2024*. https://www.jetbrains.com/lp/devecosystem-2024/

Latour, B. (1986). The powers of association. In J. Law (Ed.), *Power, action and belief: A new sociology of knowledge?* (pp. 264-280). Routledge & Kegan Paul.

Latour, B. (1987). *Science in action: How to follow scientists and engineers through society*. Harvard University Press.

Latour, B. (2005). *Reassembling the social: An introduction to actor-network theory*. Oxford University Press.

Law, J. (2009). Actor network theory and material semiotics. In B. S. Turner (Ed.), *The new Blackwell companion to social theory* (pp. 141-158). Blackwell.

Leonardi, P. M. (2011). When flexible routines meet flexible technologies: Affordance, constraint, and the imbrication of human and material agencies. *MIS Quarterly*, *35*(1), 147-167. https://doi.org/10.2307/23043493

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Sage.

March, J. G. (1991). Exploration and exploitation in organizational learning. *Organization Science*, *2*(1), 71-87. https://doi.org/10.1287/orsc.2.1.71

Markus, M. L., & Silver, M. S. (2008). A foundation for the study of IT effects: A new look at DeSanctis and Poole's concepts of structural features and spirit. *Journal of the Association for Information Systems*, *9*(10), 609-632. https://doi.org/10.17705/1jais.00176

METR. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity*. https://metr.org/blog/2025-01-10-early-2025-ai-experienced-os-dev-study/

Microsoft. (2024). *AI at work is here. Now comes the hard part: 2024 Work Trend Index*. https://www.microsoft.com/en-us/worklab/work-trend-index/ai-at-work-is-here-now-comes-the-hard-part

Noble, D. F. (1984). *Forces of production: A social history of industrial automation*. Knopf.

Orlikowski, W. J. (2007). Sociomaterial practices: Exploring technology at work. *Organization Studies*, *28*(9), 1435-1448. https://doi.org/10.1177/0170840607081138

Osmani, A. (2024, December 4). The 70% problem: Hard truths about AI-assisted coding [Blog post]. *Addy Osmani's Substack*. https://addyo.substack.com/p/the-70-problem-hard-truths-about

Osmani, A. (2025a, April 25). Avoiding skill atrophy in the age of AI [Blog post]. *Addy Osmani's Substack*. https://addyo.substack.com/p/avoiding-skill-atrophy-in-the-age

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*. https://doi.org/10.48550/arXiv.2302.06590

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2785-2799). Association for Computing Machinery. https://doi.org/10.1145/3576915.3623157

Polanyi, M. (1966). *The tacit dimension*. Doubleday.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic Books.

Stack Overflow. (2024). *2024 Developer survey*. https://survey.stackoverflow.co/2024/

Star, S. L., & Griesemer, J. R. (1989). Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, *19*(3), 387-420. https://doi.org/10.1177/030631289019003001

Strong, D. M., Volkoff, O., & Elmes, M. B. (2003). ERP systems and the paradox of control. *Proceedings of the Americas Conference on Information Systems*, Tampa, FL. https://aisel.aisnet.org/amcis2003/63/

Strong, D. M., Volkoff, O., Johnson, S. A., Pelletier, L. R., Tulu, B., Bar-On, I., Trudel, J., & Garber, L. (2014). A theory of organization-EHR affordance actualization. *Journal of the Association for Information Systems*, *15*(2), 53-85. https://doi.org/10.17705/1jais.00353

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly*, *27*(3), 425-478. https://doi.org/10.2307/30036540

Volkoff, O., & Strong, D. M. (2013). Critical realism and affordances: Theorizing IT-associated organizational change processes. *MIS Quarterly*, *37*(3), 819-834. https://doi.org/10.25300/MISQ/2013/37.3.07

Webster, J., & Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, *26*(2), xiii-xxiii.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge University Press.

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2022). Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (pp. 21-29). Association for Computing Machinery. https://doi.org/10.1145/3520312.3534864

Appendix

Appendix A: Participant Consent Form

Research Participant Information Sheet and Consent Form - [redacted] Dissertation Interview

My name is [redacted] and I am a student at the London School of Economics.
Thank you for your interest in this project. In this form there is information about the project and a requret for your consent to participate.
If you agree, please complete this form. **What is the study about?** This study explores how new AI-powered Coding tools effect the tasks, skills and responbilities of Software developers and track how these tools transform these roles. **What will my involvement be?** Your involvement will consist of taking part in a one-on-one semi-structured interview, which will last approximately 30 to 60 minutes. The interview can take place either in person or online, depending on your preference and availability. **Do I have to take part?** Participation is **voluntary.** There are no negative consequences for you if you choose not to take part in this study. It is also fine if you don't want to answer any specific questions – you can just tell me, and we will move on. **What will my information be used for?** Your information will be used solely for the purpose of a Master's dissertation submitted to the London School of Economics. It will not be shared with any third parties. **Will my information be anonymous?** Your participation will be anonymous - your name will not be used in any reports or publications resulting from the study.

1. Participants Name *

Enter your answer

2. I have read and understood the information provided about this research study and have had the opportunity to ask questions. *

- I agree
- I disagree

3. I understand that my participation is voluntary and that I can withdraw at any time without giving a reason. *

- I agree
- I disagree

4. I agree to the interview being audio recorded and understand that the recording will be transcribed. I understand that my responses will be kept confidential, my identity will be anonymized, and no identifying information will be included in any reports or publications. *

- I agree
- I disagree

5. I understand that anonymized transcripts may be analyzed using AI tools on secure platforms. Only on enterprise LLM with no data leakage. *

- I agree
- I disagree

6. I agree that anonymized data from this study may be used in future academic publications and presentations. I understand how to access my rights regarding my personal data and whom to contact with concerns *

- I agree
- I disagree

7. I agree to allow my data to be used in this research study. *

- I agree
- I disagree

Appendix B: Detailed Methodology Documentation

B.1 Interview Protocol Evolution

Version 1 (P001-P003): Initial Protocol

1. **Background**
    - Tell me about your role and programming experience.
    - When did you first encounter AI coding tools?
2. **Current Usage**
    - Which AI coding tools do you use?
    - How often do you use them?
    - What tasks do you use them for?
3. **Impact**
    - How have these tools changed your work?
    - What are the benefits and challenges?

Version 2 (P004-P009): Enhanced Protocol

Added after P001's infrastructure emphasis:

4. **Organisational Context**
    - Does your organisation have AI coding Tool policies?
    - What infrastructure supports AI coding Tool use?
    - How do team dynamics change with AI coding Tools?
5. **Skill Development**
    - Which skills have become more/less important?
    - How do you maintain traditional coding abilities?

Version 3 (P010-P014): Final Protocol

Added after emerging themes:

6. **Identity and Future**

- How would you describe yourself professionally now?
- What concerns do you have about junior developers?
- Where do you see the profession heading?

B.2 Complete Code Definitions Table

Code	Definition	Example Quote	Frequency
boundary_setting	Explicit decisions about when/where to use AI	"Critical path code, that's all me"	47
infrastructure_building	Creating systems to support AI use	"50+ markdown files with prompts"	31
educational_metaphors	Framing AI interaction as teaching	"I'm the teacher now"	28
skill_atrophy	Acknowledged decline in traditional skills	"Can't write algorithms from scratch"	23

Interview	New Codes	Total Codes	Confirmed Patterns	Decision
8	**12**	**78**	**6**	**Continue**
9	**7**	**85**	**8**	**Continue**
10	**6**	**91**	**11**	**Saturation likely**
11	**3**	**94**	**12**	**Saturation confirmed.**
12-14	**0**	**94**	**12**	**Validation only**

Pre-Study Assumptions Map

Assumption	Source	How Challenged
Younger = more adoption	Personal bias	P011 (28) resisted, P005 (42) transformed
Productivity always good	Tech enthusiasm	P003's customer automation concern

Assumption	Source	How Challenged
Linear adoption stages	TAM literature	Three stable clusters emerged
Resistance = failure	Success bias	Resisters valuable for review

B.3 Data Security and Management Protocols

Security Measures

1. **Recording Storage**: LSE OneDrive with 2FA, encrypted.
2. **Transcription Process**: Used Teams AI transcription.
3. **Anonymisation Protocol**:
    - Identity key stored separately (password-protected Excel)
    - Company names replaced with descriptors.
    - Locations generalised (e.g., "major European city")
4. **Retention Schedule**: All data deletion 6 months post-assessment

File Naming Convention

- Recordings: REC_P00X_YYYYMMDD.mp4
- Transcripts: TRANS_P00X_YYYYMMDD_ANON.docx
- Memos: MEMO_XX_YYYYMMDD.docx

Information Sheet Key Points

- Study purpose: Understanding AI coding Tool impact on professional practice.
- What is involved: 20–45-minute interview.
- Confidentiality: Full anonymisation
- Rights: Can withdraw until analysis begins
- Data use: MSc dissertation only

B.4 Theoretical Sampling Decisions

After Interview 8: Realised sample skewed toward adopters. Actively sought resisters.

After Interview 11: Three clusters clear. Sought additional participants to test boundaries specifically looking for "edge cases" between clusters.

Decision to stop at 14: No new patterns emerging. Each cluster had minimum two representatives. Pragmatic time constraints approaching.

B.5 Member Checking Process Detail

Stage 1: Transcript Verification

- Sent within 1 week of interview.
- Asked to verify accuracy, not change meanings.
- 12/14 responded with minor corrections (technical terms)

Stage 2: Interpretation Verification

- Selected one participant per cluster (P001, P008, P013)
- Sent cluster descriptions and key themes.
- Feedback incorporated:
    - P001: Emphasised infrastructure took months to build.
    - P008: Clarified organisational constraints influenced boundaries.
    - P013: Confirmed resistance was philosophical, not technical.

B.6 Quality Criteria Application Examples

Credibility

- **Triangulation**: Multiple industries, experience levels, geographic locations

Transferability

- **Thick description**: Detailed context for each participant
- **Variation sampling**: Startups to enterprises, juniors to seniors
- **Clear boundaries**: Western, English-speaking contexts acknowledged.

Confirmability

- **Negative case analysis**: Resisters challenged initial framework.

- **Data availability**: Anonymised transcripts available on request.

B.7 Interview Quality Measures

Technical Quality

- Recording quality check before starting
- Quiet environment.

Interview Technique Quality

- Avoided leading questions (flagged three instances in reflection)
- Used probing effectively ("Tell me more about...")
- Allowed silence for reflection.
- Balanced speaking time (average 70% participant, 30% researcher)

Post-Interview Reflection Template

1. Key insights from this interview
2. Questions that worked well/poorly
3. My reactions/biases noticed.
4. Follow-up questions for next interviews
5. Technical issues encountered.

Appendix C: Thematic Analysis Coding Framework

Overview

This coding frame presents the key codes that emerged from the thematic analysis of 14 semi-structured interviews with software developers regarding their experiences with AI coding tools. The analysis process began with 94 initial codes generated through line-by-line coding of interview transcripts. Through constant comparison and iterative refinement, these were consolidated to 56 codes, which were then organised into 12 categories. These categories revealed three distinct transformation patterns that form the main themes of this research. The coding frame below presents the most significant codes from each theme to illustrate the analytical process and support the trustworthiness of the findings.

Research Question

How do AI coding tools transform the professional practice of software developers?

Theme 1: Complete Transformers (36%, n=5)

Developers who fundamentally reconstructed their professional practice around AI orchestration

Category: Identity Transformation

Code	Description	Example Quote
IDENTITY_TRANSFORMATION	Fundamental shift in professional self-concept from coder to AI educator/orchestrator	"I'm the teacher now. I spend my days teaching the AI what I want, guiding it through iterations. It's completely changed how I work; I don't write code anymore; I educate AI systems." (P001)
ROLE_METAPHOR_used	Use of educational or orchestral metaphors to describe transformed role	"I'm managing AI students. I'll have one AI working on the API, another on the frontend, another writing tests, while I'm orchestrating and integrating." (P002)
CODER_TO_ORCHESTRATOR	Transition from writing code to managing AI agents	"I just get things done now. Don't call me a developer, I'm a solution creator. The code is just a byproduct of solving problems." (P005)

Category: Infrastructure Development

Code	Description	Example Quote
KNOWLEDGE_INFRASTRUCTURE_creation	Building comprehensive documentation systems to support AI use	"50+ markdown files containing organisational memory, best practices, error patterns, success templates... this infrastructure is what makes me 5x more productive than before." (P001)
ORGANISATIONAL_MEMORY	Embedding collective knowledge into AI-accessible formats	"We built internal AI coding Tools trained on years of documentation... It's like having a senior developer who knows everything about our systems." (P004)

Category: Work Practices

Code	Description	Example Quote
VIBE_CODING	Iterative, conversational programming with AI until solution "feels right"	"I iterate with the AI until the solution feels right. It's more art than science now." (P001)
MULTI_AGENT_ORCHESTRATION	Managing multiple AI instances working on	"It's like running a development team where the developers never sleep,

Code	Description	Example Quote
	various aspects simultaneously	never complain, but also never truly understand." (P002)

Theme 2: Selective Adopters (50%, n=7)

Developers maintaining strategic boundaries between human and AI work.

Category: Boundary Management

Code	Description	Example Quote
TASK_DELEGATION_to_AI	Deliberate decisions about which tasks to assign to AI	"Tests are perfect for AI, repetitive, pattern-based, clear success criteria. Even if it gets edge cases wrong, it's faster to fix than write from scratch." (P006)
TASK_RETENTION_human	Conscious preservation of certain tasks for human control	"Critical path code, authentication, payment processing, data handling that's all me. AI handles the scaffolding around it." (P006)
HUMAN_OVERRIDE_capability	Maintaining ability to reject or modify AI outputs	"It's a tool, not a rule. I decide when and how to use it based on what's appropriate." (P006)

Category: Validation Practices

Code	Description	Example Quote
SELECTIVE_VALIDATION	Context-dependent checking of AI outputs	"New features get maybe 50% checking. Anything touching existing systems gets line-by-line review." (P008)

Code	Description	Example Quote
DOMAIN_SPECIFIC_validation	Enhanced scrutiny for regulated or critical domains	"Banking regulations mean some things must remain under human control... Every line of code touching money has my fingerprints on it." (P007)

Category: Skill Preservation

Code	Description	Example Quote
SKILL_ATROPHY_risk	Recognition of declining traditional abilities	"I got to a point where I was going to write some unit tests and thought 'I shouldn't need to ask AI how to do this.' That was my wake-up call." (P008)
FUNDAMENTAL_SKILL_importance	Deliberate maintenance of core competencies	"Every Friday is no-AI day. I code everything manually to keep skills sharp. A master carpenter uses power tools but remains a carpenter." (P011)

Theme 3: Conscious Resisters (14%, n=2)

Developers actively preserving traditional expertise.

Category: Resistance Philosophy

Code	Description	Example Quote
CONSCIOUS_REJECTION	Deliberate choice to minimise AI coding Tool usage	"If you're using AI to do the work, you're not learning. You're just getting output. Someone needs to preserve real understanding." (P013)
TRADITIONAL_PREFERENCE	Valuing conventional development methods	"While others become AI operators, I maintain true developer knowledge." (P013)

Category: Value Creation

Code	Description	Example Quote
EXPERTISE_REDEFINITION	Finding new value in traditional skills as others abandon them	"I've become the go-to person for reviewing AI-generated code. Ironically, resisting AI has made me more valuable." (P014)

Universal Concern (All Participants)

Category: Professional Reproduction

Code	Description	Example Quote
JUNIOR_ROLE_impact	Concern about entry-level developer skill development	"Where will juniors learn debugging when AI fixes errors instantly? The struggle is where learning happens." (P002)
MENTORSHIP_DISRUPTION	Breakdown of traditional knowledge transfer	"Juniors ask AI before asking seniors. We're losing knowledge transfer mechanisms that built this profession." (P014)

Table 2: Category to Theme Mapping

Categories	Merged Into Theme
Identity Transformation, Infrastructure Development, Work Practices	Complete Transformers
Boundary Management, Validation Practices, Skill Preservation	Selective Adopters
Resistance Philosophy, Value Creation	Conscious Resisters
Professional Reproduction	Universal Concern (crosscutting)

Theoretical Integration

The coding process was guided by two theoretical frameworks:

Actor-Network Theory (ANT) Codes Applied:

- ACTOR_ENROLLMENT: How AI coding Tools become active participants in development networks
- AGENCY_DISTRIBUTION: Task allocation between human and AI actors

- NETWORK_FORMATION: New configurations of human-AI relationships.

Affordance Actualisation Theory (AAT) Codes Applied:

- AFFORDANCE_PERCEPTION: How developers recognise AI coding Tool possibilities
- AFFORDANCE_ACTUALISATION: Differential realisation of AI capabilities
- CONTEXTUAL_CONSTRAINT: Organisational and regulatory limitations

This coding frame demonstrates the systematic progression from raw interview data through codes and categories to the three transformation patterns that answer the research question.

Appendix D: Additional Participant Quotes and Details

Extended Participant Profiles

Complete Transformers

P001 "The Orchestrator" (Startup Developer, 4 years) Extended quote on infrastructure development:

"Each markdown file is categorised, we have prompts for different contexts, error patterns we've encountered, success templates that work. It's like building a second brain for the AI. When a new developer joins, they inherit this knowledge base instantly."

P002 "The Manager" (Healthcare ML Engineer, 5 years) On domain-specific challenges:

"Healthcare isn't like building a todo app. HIPAA compliance, medical terminology, edge cases that could literally kill someone, it took three months to train our AI systems to understand these constraints. But now? It catches compliance issues I might miss."

P003 "The Enthusiast" (Enterprise Developer, 3.5 years) Full reflection on industry implications:

"What happens when every company needs 80% fewer developers? We're building tools that make our own jobs redundant. It's exciting and terrifying. I love the productivity, but I worry about the industry's future."

P004 "The Builder" (Data Analyst, Large E-commerce, 3 years) Detailed infrastructure description:

"We have 10s of thousands of pages of documentation feeding into our LLM. Every API endpoint, every data schema, every business rule from the past decade. It's like having a senior developer who's been here since day one and remembers everything."

P005 "The Transcendent" (Freelance Developer, 8 years) On outcome-focused development:

"Clients don't care if I wrote the code or AI did. They care about results. I deliver in days what used to take weeks. The code quality? Better than what I'd write manually because AI doesn't get tired or make typos."

Selective Adopters

P007 "The Boundary Setter" (Backend Developer, Payment Processor, 4.5 years) On code review challenges:

"Every PR review now I'm wondering, did they write this or did AI? There's a certain 'smell' to AI code. Perfect but soulless. I've started requiring comments explaining the human thinking behind implementations."

P008 "The Balanced" (Backend Developer, Major Airline, 3 years) Full transformation journey:

"First month with AI: amazing, 10x productivity. Second month: wait, I'm forgetting basic syntax. Third month: full crisis, couldn't debug without AI. Now I've found balance, AI for exploration, human for implementation of critical parts. Legacy code requires understanding decades of decisions. AI can't grasp that context."

P009 "The Observer" (Backend Engineer, Fintech, 5 years) On community fragmentation:

"We're losing our shared foundation. Used to be, any developer could read any other developer's code. Now? Complete Transformers write through AI abstractions. Resisters write traditionally. We're becoming different species."

P010 "The Missing Middle showed a blend of opinion and shows that adoption is spectrum.

P012 "The Strategist" (Product Owner, Telecom, 10+ years) On product management applications:

"I use AI for structure creation, user stories, acceptance criteria templates, test scenarios. But interpreting what stakeholders need versus what they say they want? That's human work. AI can't read between the lines of a rambling requirements meeting."

Conscious Resisters

P013 "The Purist" (Software Engineer, Industrial Software, 6 years) Extended philosophy:

"Every line of AI-generated code is technical debt. You don't understand it, you can't maintain it, you can't optimise it. Sure, it works today. What about in five years when the original AI model is deprecated and no one remembers why it generated that specific pattern?"

P014 "The Sceptic" (Data Scientist, Large Consulting, 10 years) On forced compliance:

"IBM mandates Watson usage. I write my solution first, then feed it to Watson for 'suggestions' I mostly ignore. It's corporate theatre. The AI is predicting text patterns, not understanding data science. My resistance has made me the go-to person for catching AI mistakes others miss."

Detailed Infrastructure Descriptions

Complete Transformer Infrastructure Systems

1. **Prompt Engineering Frameworks**
    - Hierarchical organisation (context → domain → task → variation)
    - Version control for prompt evolution
    - Success/failure pattern documentation
    - Performance metrics tracking
2. **Domain-Specific Training Protocols**
    - Industry regulation compliance matrices
    - Technical terminology glossaries
    - Edge case documentation
    - Error pattern libraries
3. **Organisational Knowledge Integration**

- API documentation ingestion
- Historical codebase analysis
- Business rule encoding
- Team convention capture

Selective Adopter Boundary Systems

1. **Decision Matrices**
    - Task criticality assessment
    - Security sensitivity scoring
    - Regulatory requirement mapping
    - Performance impact evaluation
2. **Validation Protocols**
    - Context-specific review percentages
    - Critical path identification
    - Legacy system interaction rules
    - Compliance checkpoints

Conscious Resister Preservation Systems

1. **Knowledge Documentation**
    - Algorithmic reasoning capture
    - Design decision rationale.
    - Debugging methodology
    - Performance optimisation techniques
2. **Review Frameworks**
    - AI pattern detection methods
    - Code quality metrics.

- Security vulnerability identification
- Architectural coherence assessment

Task Delegation Details

Complete Statistical Breakdown

Universal Delegation Tasks:

- Test writing: 12/14 (85.7%)
- Documentation: 11/14 (78.6%)
- Boilerplate code: 10/14 (71.4%)
- Data validation: 9/14 (64.3%)
- API integration: 8/14 (57.1%)

Universal Retention Tasks:

- Architecture design: 14/14 retained (100%)
- Security implementation: 13/14 retained (92.9%)
- Core business logic: 13/14 retained (92.9%)
- Performance optimisation: 12/14 retained (85.7%)
- User experience decisions: 12/14 retained (85.7%)

Extended Quotes on Junior Developer Crisis

P001: "We're creating a generation of developers who can build but not debug, create but not understand. It's like teaching someone to drive using only self-driving cars."

P004: "Our junior hire can build features faster than I could at 5 years' experience. But ask them to explain how it works? Blank stares. They're productive but not knowledgeable."

P007: "Traditional learning meant struggling with problems, finding solutions, understanding why things work. Now juniors get instant answers. Where's the learning in that?"

P009: "The apprenticeship model is broken. Seniors don't review junior code because AI already has. Juniors don't ask seniors questions because AI answers faster."

P011: "Every bug you struggle with teaches you something. Every algorithm you implement builds intuition. AI removes the struggle, and with it, the learning."

P013: "We're heading for a cliff. In 5-10 years, when today's juniors should be seniors, they won't have the deep knowledge needed for system architecture, performance optimisation, or debugging complex issues."

Interview Context and Reflexive Notes

Recruitment Challenges

- Initial difficulty finding Conscious Resisters
- Geographic bias toward English-speaking countries
- Company size skewed toward large enterprises.

Interview Dynamics

- Complete Transformers: Enthusiastic, often exceeded time.
- Selective Adopters: Measured, precise responses.
- Conscious Resisters: Initially defensive, then philosophical

Researcher Observations

- Surprise at depth of identity transformation
- Unexpected universality of junior developer concerns
- Infrastructure investment levels exceeded expectations.
- Cluster boundaries more permeable than anticipated.

Appendix E: Semi-Structured Interview Questions

Semi-Structured Interview Guide: AI Coding Tools and Programmer Role Transformation – Naoise Law

Interview Duration: Estimated 30 minutes per interview

Opening (3 minutes)

Introduction and Consent

- Thank you for participating in this research.
- This interview is part of my dissertation research at LSE examining the impact of AI tools like GitHub Copilot, Claude, and ChatGPT on software development
- Your responses will be kept confidential and anonymous. Once analysed, all data will be deleted per GDPR and ethical guidelines.
- May I record this interview for transcription purposes?

Section 1: Background and Context (5 minutes)

1.1 Role and Experience

Can you tell me about your current role and how long you've been working in software development?

- *Probe: What programming languages do you primarily work with?*
- *Probe: What type of projects or systems do you typically work on?*

1.2 AI Tool Exposure

Which AI coding tools have you used or encountered in your work?

- *Probe: When did you first start using these tools?*
- *Probe: How frequently do you use them (daily, weekly, occasionally)?*

Section 2: Current AI Tool Usage and Integration (5 minutes)

2.1 Implementation and Adoption

How were AI coding tools introduced in your organization or personal workflow?

- *Probe: Was this a top-down decision or bottom-up adoption?*
- *Probe: What was your initial reaction?*

2.2 Specific Use Cases

Can you walk me through how you typically use AI coding tools in your daily work

- *Probe: What specific tasks do you use them for most often?*
- *Probe: Are there tasks where you deliberately avoid using AI tools?*

2.3 Integration Challenges

What challenges did you face when first integrating these tools into your workflow?

- *Probe: How did you overcome these challenges?*

Section 3: Role and Task Transformation (7 minutes)

3.1 Daily Work Changes

How has your day-to-day work changed since you started using AI coding tools?

3.2 Skill Evolution

What new skills have you developed to work effectively with AI tools?

- *Probe: How do you evaluate AI-generated code?*
- *Probe: Have any of your traditional coding skills become less important?*

3.3 Role Perception

How would you describe your role now compared to before using AI tools?

- *Probe: Do you see yourself more as a code writer, code reviewer, or something else?*

3.4 Decision Making and Agency

How do you decide when to accept, modify, or reject AI-generated code suggestions?

Section 4: Impact Assessment (8 minutes)

4.1 Productivity and Efficiency

How have AI tools affected your productivity and work quality?

- *Probe: Can you give specific examples of time saved or efficiency gains?*
- *Probe: Have you noticed any changes in the quality of your work?*

4.2 Benefits and Challenges

What do you see as the main benefits of using AI coding tools?

- *Follow-up: What are the main challenges or drawbacks?*

4.3 Team Dynamics

How have AI tools affected collaboration and teamwork in your organization?

- *Probe: Has it changed code review processes?*

Section 5: Future Perspectives and Concerns (2 minutes)

5.1 Future Expectations

How do you expect AI coding tools to evolve in the next 2-3 years?

- *Probe: What changes do you anticipate in your role or the industry?*

5.2 Hopes and Fears

What are your main hopes or fears for the future of AI in software development?

- *Probe: Do you have any concerns about job security or skill obsolescence?*

Closing (2 minutes)

Is there anything in your experience using these AI coding tools we haven't covered that you think would be important?

Thank you so much for your time and please fill out the consent form that will be emailed to you. This has to be completed to use the content of the interview.

Thanks again and let me know if you are interested in the results.

Appendix F: AI Acknowledgement

I state that the use of Generative AI during this study was well within the line set with the MG4D7 MISDI Dissertation "Limited Authorised Use" policy. Claude was primarily used to assist me in maintaining deadlines and coherency between chapters and helped me with storing and then reflecting on my ideas for the research, I did also use tools were in the early stages of the research process for brainstorming, helping me structuring ideas, and helping me understand academic theories, Zero Generative AI was used for the data analysis and I did not upload data to these platforms. All feedback and suggestions provided by the tools were critically reviewed by myself.