CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

AI in the Loop: How AI Affects Roles, Collaboration, and Trust in Agile Software Development

Master's Thesis in Computer science and engineering

Pontus Brylander and Johan Franz

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

AI in the Loop: How AI Affects Roles, Collaboration, and Trust in Agile Software Development

Pontus Brylander and Johan Franz

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

AI in the Loop: How AI Affects Roles, Collaboration, and Trust in Agile Software Development
(Pontus Brylander and Johan Franz)

Supervisor: Linda Erlenhov, Department of Computer Science and Engineering
Examiner: Aris Alissandrakis, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Pontus Brylander and Johan Franz
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Large language model-based tools have rapidly become part of the day-to-day workflow in software engineering, with growing evidence that they affect quality and productivity on an individual level. Less, however, is studied about how these tools shape collaborative practices within agile and DevOps teams, where communication, shared responsibility, and iterative decision-making form the basis of effective work. This thesis investigates the perceived effects of these tools on decision-making, role distribution, collaborative practices, and trust in AI-generated code within agile and DevOps team environments.

A mixed-methods approach is used, combining semi-structured interviews with eleven developers across a varying range of company sizes, sectors, roles, and experience levels, and a complementary survey. The data collected through the interviews were analyzed using Braun and Clarke's thematic analysis framework, supported by Socio-Technical Systems theory and Lee and See's framework for trust in automation.

The findings indicate role changes are ongoing rather than fully completed, with participants describing AI tools being used for decision-making rather than as delegated decision-makers. AI was described as displacing informal channels of knowledge transfer through which knowledge has historically been shared within agile teams. This displacement was particularly found between junior and senior developers. AI was also described as an amplifier of existing quality practices rather than a transformer. Trust was actively constructed through manual verification, such as review and testing, with experienced users reporting more calibrated than uniform trust. The study contributes to prior work by examining AI's effects at the team level, a largely overlooked area in the literature focused on either the individual developer or the organization.

Keywords: LLM, large language model, software engineering, DevOps, collaboration, code quality, trust, thematic analysis, socio-technical systems

Acknowledgements

We would like to express gratitude to our supervisor, Linda Erlenhov, for their guidance, feedback, and continuous support throughout this thesis. Their insights have been valuable in shaping the direction and quality of this work.
We would also like to thank our examiner, Aris Alissandrakis, for their helpful comments and suggestions, which improved the overall outcome of the thesis.

(Pontus Brylander and Johan Franz), Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

The use of large language models such as GitHub Copilot, ChatGPT, Claude, and, more recently, agentic AI has grown rapidly and become part of everyday software development practice. These tools can help developers with debugging, code generation, documentation search, and creation, among many other tasks. The adoption is accelerating across the industry, while the final outcome of this change remains unknown. Productivity gains for individual developers at the cost of perceived code quality have received attention in the research by Martinović and Rozić [1], but less is known about the effects on how these tools shape collaborative practices in software teams, particularly agile and DevOps teams, where communication, shared responsibility for code quality, peer review, and iterative decision-making are the core principles of success.

The introduction of AI raises questions that go beyond individual performance: How does the presence of AI-generated code affect team discussions and peer-review practices regarding code quality? Do developers perceive roles and decision-making to change when AI is involved? And how do teams assess whether AI-assisted output can be trusted to the same standard as human-written code?

This thesis investigates these questions through a mixed-methods study that combines semi-structured interviews with developers on agile and DevOps teams and a complementary quantitative survey. The focus is on the socio-technical effects of AI adoption at the team level, encompassing decision-making, role distribution, collaborative practices, and developers' assessments of trust in AI-generated code.

1.1 Problem Description

Incorporation and use of AI tools have increased in recent years; their use ranges from a passive tool for speeding up repetitive tasks to an integral part of the collaborative decision-making process [2]. Concerns about the shift of responsibility are emerging from this rapid reshaping of the workflow regarding reliability and security [2], as well as the quality of generated code [1]. Studies have shown that developers perceive AI adoption as improving their productivity [1], and that higher perceived productivity of AI systems is associated with increased user trust in AI decision-making [3], which was found to be consistent with the earlier work on algorithm reliance [4]. However, perceived effectiveness does not guarantee correctness, and the potential for misplaced trust in AI systems underscores the need for human oversight and validation.

Despite the growing adoption of AI in software development, there is still a gap in the understanding of its effects [5]. Previous research has mainly focused on AI's influence at an individual level[1][2][6][7][8][9][10], leaving open questions on how AI affects teamwork within agile and DevOps workflows. Specifically, how role distribution and accountability shift when AI becomes part of the development process is insufficiently explored. This includes AI's involvement in practices such as peer-reviewing, discussions of code quality, and the assessment of trust in AI-generated output.

This gap suggests the need for further investigation, as software engineering is a discipline in which team communication, coordination, and shared responsibility for code quality are foundational principles of success [11][12]. If these collaborative processes are altered without a clear understanding of the consequences, it could lead teams to engage in superficial code reviews, over-reliance on AI, and the introduction of undetected bugs and vulnerabilities. Without empirical research into these topics, it becomes troublesome for teams and organizations to incorporate these tools responsibly or to establish well-informed guidelines for human-AI collaboration.

1.2 Purpose of the Study

The purpose of this study is to address the limited empirical understanding of how LLM-based tools reshape the collaborative workflows, decision-making, responsibility, and developers' assessment of code quality and trustworthiness within agile software development teams. While prior work has examined AI's perceived effects on individual productivity and code quality [1][2][6][7][13][14], the implications at the team level in agile and DevOps contexts, where collaboration is fundamental, have yet to be thoroughly explored.

For practitioners and teams working in agile environments, the study may offer insights into how AI is influencing their workflows and where caution is warranted. For organizations and managers, it can give informed insights to develop guidelines for responsible AI adoption. For researchers, it provides empirical grounding

for studying the long-term socio-technical effects of AI integration within software development teams.

1.3 Significance of the Study

This study contributes to the research on human-AI collaboration in software engineering. Most existing empirical work examines how AI tools affect the speed or quality of individual developers [1][7][15][8]. This study extends that focus by examining how AI shapes the collaborative practices underpinning agile software development.

From a practical perspective, the findings may help highlight areas where AI is subtly altering established workflows, allowing development teams and their managers to identify these shifts and ensure a responsible, deliberate integration of AI into development teams, rather than allowing practices to drift unexamined.

From a research standpoint, the study contributes qualitative empirical insights into a rapidly evolving phenomenon. Given the pace of AI tool development, documenting developers' experiences and perceptions during this transitional phase establishes a foundation for subsequent longitudinal studies. The mixed-methods design, combining thematic analysis of interviews with survey data, demonstrates the value of complementary methods for studying socio-technical change in software teams.

1.4 Research questions

RQ 1: How do software engineers perceive that LLM-based AI tools have influenced decision-making, role distribution, and their professional roles within agile software development teams?

RQ 2: How do software engineers perceive that LLM-based AI tools have influenced collaborative practices within agile software development teams, particularly peer-review practices, team dynamics, and discussions of code quality?

RQ 3: How do software engineers within agile software development teams assess the reliability and trustworthiness of code produced with the assistance of LLM-based AI tools?

2

Background

This chapter presents the necessary technical background to contextualize the study. It introduces AI technologies relevant to the research, the software development methodologies in which these tools are integrated, and the concept of trust in automated systems.

2.1 Large Language Models

Artificial Intelligence (AI) is a broad umbrella term for a variety of implementations focused on building systems that require human-like intelligence. Large Language Models (LLM) fall under this umbrella and are specifically designed to comprehend and generate human language or any other type of language, such as programming language [16]. These models are trained on a large text dataset; this text is first translated into units called tokens, which may represent words or characters. Self-supervised learning is used to model language as a distribution of probabilities over token sequences. This means that given previous tokens, it predicts the most probable forthcoming token. Modern LLMs are based on the transformer architecture [17], which dramatically improves efficiency through self-attention mechanisms that model dependencies between tokens regardless of their distance in the sequence. The self-attention mechanism assigns different weights to tokens in the input sequence, enabling contextual relationships and long-range dependencies more effectively than earlier models. In practice, training these models often includes an initial pretraining followed by fine-tuning, such as reinforcement learning from human feedback, to optimize performance and usability. LLMs trained on large source code datasets are the foundation of most AI coding tools. This is due to programming languages sharing common traits with natural languages, such as grammar, syntax, and compositional rules, which makes LLMs especially well-suited to comprehend and generate code.

2.2 Agentic AI

Agentic AI is a type of AI designed to autonomously solve a range of tasks by integrating planning and reasoning. What sets agentic AI apart from standalone models such as LLMs, which largely generate outputs for a single input, is that

agentic AI operates over multiple steps dynamically and interacts with the environment through a feedback loop. Modern agentic AI increasingly leverages LLMs for its reasoning [18]; in this setting, LLM is used to interpret goals and define actionable tasks. LLMs have become foundational to the development and evolution of agentic AI, as they have enabled more sophisticated planning and problem-solving [19]. In addition, agents are equipped with external tools, such as APIs, search engines, and code execution environments, enabling them to perform actions autonomously. A wide range of agentic AI tools used in software engineering illustrates these capabilities, including Claude Code, Cursor Agent, and Devin [20][21][22]. A defining characteristic of agentic AI is the iterative control loop the agent uses to evaluate the current state, select actions based on that state, and observe the outcome to update its behavior. This ability to adapt to multi-step processes is the key to dynamic problem-solving and error correction under changing conditions.

2.3 AI-Assisted Software Development Tools

AI tools are increasingly integrated into software development workflows, supporting a wide range of programming tasks. How developers interact with these tools varies by category. There are generally three interaction patterns commonly observed:

2.3.1 Inline Completion Tools

Inline completion tools are often integrated directly into code editors, aiming to predict and suggest code as developers type. These tools primarily operate at a level of small code snippets or individual lines, offering context-based predictions [8]. Offloading routine and repetitive tasks can accelerate productivity and possibly reduce syntax errors [6].

2.3.2 Chat-Based Tools

Chat-based tools allow developers to interact with AI systems through conversational language. Unlike inline tools that suggest solutions passively within the editor, chat-based allows for a broader range of capabilities. They can be used to explain unfamiliar code, answer technical questions, or generate code from examples and descriptions [9]. This interaction enables a more flexible approach to solving issues, particularly for debugging and reasoning about code.

2.3.3 Agentic Tools

Agentic AI tools move beyond single-step interactions by operating autonomously in multiple steps within a given scope or environment [18]. Rather than suggesting individual lines or responding to single prompts, it decomposes larger tasks into smaller sub-tasks, executes them through its external tool capabilities, and adapts dynamically based on observed outcomes [18]. This enables end-to-end workflow automation in software development.

Table 2.1: Overview of AI tools mentioned in this study

Tool	Category	Description
GitHub Copilot (Inline)	Inline code completion	AI-powered autocomplete
GitHub Copilot Chat	Conversational assistant	Chat interface integrated in the IDE
ChatGPT	Conversational assistant	General-purpose LLM chat
Codex	Agent	Coding agent
Claude	Conversational assistant / Agent	Available as a chat interface and as a coding agent
CodeRabbit	Automated code review agent	AI-driven tool that reviews pull requests
Internal LLM	Conversational assistant	On-premises chat interface, typically based on commercial models

2.4 Agile and DevOps Practices

Agile and DevOps methods are fundamental in modern software development, both emphasizing collaboration, iterative progress, and continuous improvement. The iterative process in agile practices works by dividing tasks into short cycles, known as sprints, enabling quick responses to changing requirements [11]. DevOps complements this process as the bridge between the development of code and the operations for deploying it in order to streamline the software delivery [23]. This includes handling continuous integration and continuous delivery (CI/CD) processes throughout the software life cycle and facilitating scalability [24]. The DevOps role aims to reduce silos between developers and operations personnel, cultivating a culture of shared ownership of code and deployments, resulting in faster, more reliable release cycles and improved system stability [23]. Early foundations of agile practices can be traced to the work of Nonaka and Takeuchi [25], who described product development as involving overlapping responsibilities and continuous multidimensional learning. This concept of "multilearning" allows teams to develop skills and knowledge across multiple sectors and aligns closely with agile principles of cross-functional development and collaboration.

Collaborative workflow is central in both agile and DevOps, and teams contribute during multiple stages of the development process. Common practices include daily stand-up meetings, collective sprint planning, and code discussions through peer reviews [26]. Peer-review processes play an important role in maintaining code quality and enabling collective decision-making within the team, where code changes are evaluated not only for correctness but also for maintainability and alignment with the larger project, facilitating knowledge sharing among team members [12].

2.5 Trust in Automated Systems

Trust in automated systems is a concept that encompasses the attitude, willingness, and appropriateness to rely on an automated system to achieve a goal, particularly in situations of uncertainty [27]. The development of trust is mainly based on emotions [27], including irrational feelings that cannot be explained. Furthermore, trust can be understood as a spectrum, where misuse and disuse depend on the mismatch between trust placed in a system and its actual capabilities [28]. A three-layered trust model has been synthesized from empirical literature [29]. This model distinguishes between **dispositional trust**, representing an individual's general tendency to trust automation regardless of context or specific system; **situational trust**, which is influenced by the immediate context; and **learned trust**, which is built from experience with a specific system.

A framework for automation misuse, disuse, and abuse has previously been introduced [28], in which misuse of a system is referred to as over-reliance on automation without sufficient thorough verification and can lead to unexpected errors. Disuse, by contrast, refers to under-reliance, where an automation is ignored or rejected even though it has beneficial capacity because of false alarms. Automation abuse,

however, is the careless implementation that fails to consider the consequences of human performance. In the context of AI tools in software development, misuse would be compared to developers implementing AI-generated code without sufficient review. Disuse refers to developers avoiding AI tools despite their potential benefits, whereas abuse would be associated with organizations or managers failing to provide sufficient guidelines or training for their use.

A previous study demonstrated experimentally that developers using AI tools for code creation introduced more vulnerabilities compared to those without AI assistance, while simultaneously reporting a higher confidence in the security of what they produced [15], which is a clear example of mismatched trust. This "illusion of correctness" presents a challenge of trust as the output from AI tools often appears syntactically correct while introducing hidden bugs or flaws that are difficult to detect in a casual review, unlike traditional automated systems, where failures often are directly visible and identifiable, such as alarm ringing or a machine stopping working.

While previous studies have mainly focused on trust at the individual level, these findings may also carry implications for teams. If individual trust is mismatched, it may undermine collaborative practices in agile development and mutual confidence, as varying calibration of trust towards AI tools may introduce asymmetries in review practices and accountability.

3

Related Work

Literature relevant to this thesis is reviewed to position it well within the academic field.

3.1 Adoption of AI and Developers Productivity

LLM-based AI tools have been rapidly and widely adopted in software development teams, with Martinović and Rozić [1] reporting that nearly all surveyed developers use them in their workflow. The introduction of AI tools among professionals is transitioning from experimental use to operational use, although concerns about output reliability suggest the shift remains incomplete, as evidenced by persistent challenges such as hallucinations, data drift, and non-deterministic output [30]. De La Cruz et al. [2] suggest that the ongoing adoption of AI is emotionally charged, with some developers offloading their tasks to AI, while others resist due to concerns about its potential effects on cognitive skill development and the loss of professional identity [2]. At the same time, AI adoption is associated with productivity gains in previously time-consuming tasks such as prototyping and problem-solving. However, these benefits come with hidden costs: developers describe AI outputs to appear fluent and syntactically correct, yet logically flawed, often masking faults and vulnerabilities that may lead to extra time spent fixing these issues. As speed increases, so does the overhead of ensuring code quality.

The reported productivity gains are substantial: Cui et al. [7] reports a 26.08% increase in productivity, with considerable variation across companies when completing tasks with access to AI tools. In addition, less experienced developers showed higher adoption rates and greater productivity gains, suggesting AI may act as a leveling mechanism, disproportionately benefiting them. This is further supported by qualitative evidence suggesting that less experienced developers rely more on AI tools, using them for navigation, exploration, and solution discovery rather than accelerating tasks they already understand [8]. More experienced developers, by comparison, use AI tools to accelerate work they already know how to implement, limiting the tool's potential contribution. However, productivity benefits are not universally reported. Choudhuri et al. [5] found no statistical differences in productivity between participants using ChatGPT [31] and those using traditional resources, when examining undergraduate software engineering students. The group

using ChatGPT reported increased frustration stemming from hallucinations, incorrect guidance, and an inability to comprehend context-dependent problems.

3.2 Roles and Workflows in AI-Assisted Software Development

AI tools are affecting software developers beyond productivity gains, as developers are seeing a change in the nature of their day-to-day workflows. Khojah et al. [9] conducted an observational study of 24 professional engineers using ChatGPT over one week in their work, and found that rather than using the AI tool to generate ready-to-use artifacts, participants most commonly used the tool to receive guidance on how to approach and solve tasks, using it more as a virtual colleague than a code generator. This suggests the tools are being used in a more interactive, iterative manner. Building on this, research on GitHub Copilot found that interactions with AI tools vary depending on the developers' prior knowledge, where the tool's role in the workflow depended on what the developer already knows, accelerating familiar tasks for experienced users, while others used it more exploratively [8]. This differential is further supported by Ziegler et al. [6], which demonstrates that acceptance rates of AI tool outputs vary significantly across the developer population. Less experienced developers report greater perceived benefits, including improved code quality, whereas more experienced developers are less inclined to attribute such improvements to AI tools, though they report greater benefits in other dimensions, particularly a reduction of effort on repetitive tasks. Together, these findings raise an important question about how AI affects the traditional hierarchy of expertise within teams. If AI tools narrow the performance gap between junior and senior developers, it may warrant a reconsideration of role distinctions, decision-making authority, and the distribution of responsibilities across experience levels. Furthermore, the comparatively lower acceptance rates and skepticism among senior developers may introduce friction within teams, particularly in contexts where norms around the appropriate use of AI tools are still under negotiation.

Bruhin [30] and De La Cruz [2] observe a pattern of developers shifting from writing code to overseeing, reviewing, and validating AI-generated output. Bruhin [30] reveals a transition beyond individual tasks, with a structural reorientation across roles in software engineering becoming manifest, specifically in reduced demand for traditional software developers. These roles are expected to be replaced by a growing need for business analysts who can bridge the gap between technical and business requirements. De La Cruz et al. [2] report that developers are increasingly evolving into roles centered on validating and reviewing code when collaborating with AI tools, managing the output rather than writing code from scratch. Together, these findings reflect a fundamental change in what it means to be a software developer, where the core competency shifts from producing code to evaluating code produced by an external system. In addition, this shift is accompanied by engineers adapting their workflows to AI tools, and the tools in turn adapting to how they are used. However, this co-evolution risks widening the performance gap among developers,

as high performers are likely to benefit disproportionately from AI assistance, while low performers may struggle [30].

3.3 Effects of AI on Code Quality

Although the productivity benefits of AI tools are well documented [7], the impact on code quality is less straightforward. Martinovic and Rozić [1] find that AI-generated code frequently falls short in quality dimensions, including readability, maintainability, and accuracy, which are critical components to long-term software health. This is supported by Liu et al. [13], which conducted a large-scale evaluation of over 4,000 AI-generated programs and found that nearly half exhibited maintainability deficiencies, including those that were functionally correct. These findings underscore the distinction between functional correctness and production-ready code, indicating that passing tests does not necessarily mean that the code meets the standards required for long-term maintainability, an area where AI appears particularly prone to neglect.

These quality dimensions extend into security, where the consequences are significantly more severe. Pearce et al. [14] systematically evaluated GitHub Copilot [32] across security-relevant scenarios [14]. A particularly concerning finding was that approximately 40% of top suggestions contained vulnerabilities [14], meaning that insecure code was being presented as the best recommendation. The authors noted this as an elevated risk for less experienced developers, as they may be more inclined to accept top suggestions with less critical scrutiny [14]. Similarly, Perry et al. [15] extended this with a human-subjects experiment and found that developers using AI assistants wrote significantly less secure code with an elevated false sense of security about it. Together, these findings establish that AI tools can introduce security risks that developers might struggle to identify.

Verifying AI-generated code is particularly difficult as code produced, even though incomplete or incorrect, often appears plausible on the surface [30][15]. Furthermore, De La Cruz et el. [2] emphasize the importance of strategic evaluation and careful decision-making when incorporating AI-generated code, since traditional quality assurance relies on human verification and reasoning. As AI code breaks this assumption, no human author can be consulted about the intent behind coding decisions, as the review shifts from assessing a colleague's intent to reverse-engineering a system's output.

3.4 Team Practices and Collaboration with AI

Collaboration is one of the core pillars within agile and DevOps teams, where practices such as peer review serve not only as a quality gate, but as a mechanism for knowledge transfer, and shared awareness, including alignment of standards within teams [12]. The ritual of daily standups functions as a coordination to maintain a shared understanding of the current status of blocking tasks or progress of the

system as a whole [26]. Pair programming, commonly described through the roles of a driver and navigator, or more informally, pilot and copilot, facilitates real-time knowledge sharing between developers with different levels of expertise and experience [33]. Furthermore, this practice is inherently social, depending on continuous human interaction, communication, and the establishment of a shared context of the task at hand. AI-assisted development tools such as GitHub Copilot [32] position themselves as assistants in the development process, adopting the "copilot" terminology, which is suggestive of collaborative work practices [8]. However, this analogy is only partially aligned with traditional pair programming practices. A human navigator contributes explicit reasoning that can be articulated, challenged, and negotiated [12]. In contrast, Wang et al. [10] found that users evaluating AI-generated suggestions often had to rely on inefficient manual validation methods and personal intuitions, rather than on informed reasoning about the model's decision-making process, due to insufficient support for evaluating the quality of the suggestions. Furthermore, Imai [34] investigated whether GitHub Copilot [32] can serve as a substitute for human pair programming found that, although productivity increased, measured in terms of lines of code added, the resulting code quality was lower compared to that produced through human pair programming. This was evidenced by higher rates of later code deletions. These findings suggest that, despite productivity advantages, AI tools do not replicate the knowledge-sharing and quality assurance functions that human collaboration provides.

3.5 Trust and Reliance in AI-Assisted Software Development

Trust in AI tools directly affects how developers interact with and rely on them. Yu and Li [3] found that higher perceived effectiveness of AI is associated with increased trust, while a simultaneous lack of transparency in the AI's decision-making process may produce discomfort that inhibits trust and reliance. Castelo et al. [4] demonstrated that trust in algorithms is task-dependent, with users showing greater resistance toward algorithmic judgment on subjective tasks compared to objective ones. These findings are relevant to software development, which encompasses both objective elements, such as whether code compiles or passes tests, and subjective elements, such as architectural decisions, readability, and adherence to conventions set by the team or organization. This suggests that developers may selectively trust AI output, being more inclined to trust it for routine, well-defined tasks while remaining skeptical of suggestions that require contextual understanding. Regarding AI-assisted code generation, Wang et al. [10] found that existing tools provide insufficient support for evaluating output quality, forcing developers to either accept suggestions on faith or invest considerable effort in manual verification. This problem creates an environment where calibrating trust is difficult; developers may lack the signals needed to distinguish reliable output from unreliable output. The consequences of miscalibrated trust are further backed by the findings of Perry et al. [15], where developers using AI assistants produced less secure code while reporting higher confidence in its security. This demonstrates a gap between perceived and

actual trustworthiness.

Developer experience levels further moderate this dynamic. As noted in section 3.2, acceptance rates vary significantly by experience level [6], with less experienced developers showing higher acceptance. This may have implications for trust, as Pearce et al. [14] noted that high-confidence scores on vulnerable code may pose a higher risk for less-experienced developers who might not have enough expertise to question suggestions. Bruhin [30] observed that the co-evolution between developers and AI tools might risk widening the performance gap as developers with high evaluation skills benefit more from AI assistance, while those who lack such skills may develop uncalibrated trust. This dynamic is consistent with the patterns of automation misuse and disuse predicted by Parasuraman and Riley [28]. Together, these findings indicate that trust in AI is not uniform across a team but varies with individual experience and skill, potentially creating asymmetries in how team members evaluate and rely on AI-generated output.

3.6 Positioning of This Study

A limitation of the literature reviewed in this chapter is a pattern of focus in which the effects of AI tools are studied either at the level of an individual software developer or at the level of an organization. The effects on agile and DevOps development, and how team dynamics within these teams remain largely absent. Benefits, such as developer productivity, are measured on an individual level [7][6], and code quality is inspected through static code analysis tools, tests, or automated metrics [14][13] or through developers' individual perception [1][15]. Role and workflow changes are primarily described in terms of individual developers' practices [9][8] or through the lens of an organizational structure [30], and analyzed from an individual human-AI co-evolutionary angle [2]. Trust is likewise examined as an artifact of the individual developers' interaction with AI tools [3][10]. What remains largely unanswered is the question of the inherently social and collaborative nature of software development. In agile and DevOps settings, roles are not isolated but are built through shared practices [11]. These practices involve collective code ownership, mutual accountability, and iterative decision-making. If team members differ in their adoption and acceptance of AI tools, this may disrupt established patterns of shared decision-making and accountability in ways not observable at the individual level. Furthermore, code quality is not a fixed property of an artifact in these environments but is negotiated and shaped through team consensus [11] and mechanisms such as peer review [12]. The purpose of this review process largely depends on communication between the author and reviewer within the framework of intent, context, and alternative thought processes. When AI tools and generated code are introduced, this communicative dynamic may be weakened. If no human author can be consulted about the reasoning behind coding decisions, the review shifts from a collaborative exchange to an evaluation of outputs that may have been produced with varying levels of scrutiny [14]. Finally, trust and acceptance in AI tools have been found to vary with individual experience levels [6][14]. How these individual differences in trust calibration come at the cost of collaborative quality, and

whether one developer's trust level affects other team members' perceptions of their contributions, remains to be investigated.

4

Theory

4.1 Thematic Analysis as an Analytical Approach

Understanding how AI has influenced social interaction within teams requires a theoretical lens that can account for both technological capabilities and human social processes. To ensure that the information gathered was not projected, a semi-structured interview was used [35]. The framework that further builds on semi-structured interviews and their analysis is Clarke and Braun's thematic analysis [36].

Core concepts from this framework guide the conduct of this thesis. Semi-structured interviews provide flexibility while maintaining focus on the relevant topic [35]. This allows for a predetermined interview guide while remaining open to unexpected themes that we construct from conversation. Furthermore, the thematic analysis provides a structured analysis process. The process follows a six-step analysis: 1. Familiarizing yourself with your data, 2. Generating initial codes, 3. Searching for themes, 4. Reviewing themes, 5. Defining and naming themes and 6. Producing the report [36]. Through this framework, a well-structured analysis can be conducted.

Coding with thematic analysis principles means that citations are condensed into shorter versions of the original statements. With this, reviewing themes and codes in relation to citations is highly important to ensure that what is portrayed mirrors what participants said.

The thematic analysis framework is applied in this study through both a practical and analytical lens. The research questions are exploratory and are centered on the perceived impact on the team and the individual. This framework helps with the analytical structure of the non-measurable answers. Furthermore, this provides flexibility for emerging patterns as LLM-based AI tools are not well established in the literature, as mentioned in chapter 3. An inductive approach is therefore more appropriate than other preexisting categorizations of the data.

Braun and Clarke's six-step procedure offers analytical advantages, as it is well documented and can be revisited at different stages if needed [36]. Through their process, analysis of codes from transcriptions can be provided, and patterns through grouping can be found. Through these patterns, it is possible to examine how

decision-making, collaboration, and trust within software development teams are formed.

4.2 Socio-Technical Systems

To understand the impact of AI on social interactions, collaboration, and decision-making, this study focuses on its effects in workplaces. Socio-Technical Systems (STS) Theory, founded by Trist and Bamforth, describes how technology impacts the workers' teamwork [37]. The conclusion is that new technology cannot be implemented without considering employees; otherwise, negative effects could occur. These foundations will later be evolved by many others, including Trist and Bamforth, leading to our understanding of the impact of new technologies on the employees. Bostom and Heinen refined this framework by studying Management Information Systems (MIS) and arguing that without proper trust and understanding, employees' social relations can be negatively affected [38].

The main idea of the STS theory is to recognize that the two groups, technological subsystems and social subsystems, must coexist and work together rather than in opposition [37]. The technological subsystem includes technology, tasks, and physical work arrangements. In this study, the technology subsystem comprises the LLM-based AI tools used by developers and the workflow characteristics of agile and DevOps teams. The social subsystem comprises the developers themselves and the collaboration practices through which they interact. Following the principle of joint optimization, this study examines how these two subsystems interact rather than treating them in isolation.

This study investigates how social interaction has been affected within a software development team. To evaluate this, an understanding of the team and the individual is critical. The interaction with AI in software development teams occurs at both the collective (team) and individual (developer) levels, depending on usage. It is therefore vital to understand both.

STS Theory provides the lens through which observation of team-level dynamics can be understood. The justification of this usage is based on three main points:

1. Independence of Technical and Social Systems: AI tools have flooded the online and workflow market. These tools are widely used and embedded in software development teams, influencing individuals and their social interactions. STS theory is built on this interdependence, making it a great fit for understanding collaboration, trust dynamics, and decision-making processes.

2. Alignment with Agile and DevOps Contexts: Software development teams working in agile and DevOps inherently embody socio-technical principles through collaboration, rapid iteration, and integrated workflows. STS focuses on joint optimization of technical and social systems, which aligns neatly with our study.

3. Focus on Emergent Outcomes: STS theory highlights that organizational

outcomes come from interaction between new technology and social arrangements rather than from subsystems independently. This perspective is highly important for understanding how AI-driven changes in work environments affect team collaboration and decision-making.

4.3 Trust in Automation

While the STS theory provides an understanding of the relationship between humans and technical and social subsystems, it does not explain why individuals rely on or do not rely on a technology. To address this gap of individuals reliance, the insight from Lee and See's framework will be implemented [27]. To understand when and why a developer trusts or does not trust AI, this framework can help us better understand in depth.

Lee and See describe trust as: "the attitude that an agent will help achieve an individual's goals in a situation characterized by uncertainty and vulnerability." [27]. This definition is relevant to discussions of AI, as an individual's trust in what is produced affects how it is used and to what extent. Knowing why a developer trusts or does not trust something they cannot see or fully understand is important.

The appropriateness of trust is a core part of this framework and is divided into three dimensions: calibration, resolution, and specificity [27]. Calibration refers to a person's level of trust in the device's capabilities. If a level of trust exceeds its abilities, it is categorized as misuse; the opposite is called disuse. Resolution is described as the degree to which a judgment of trust differentiates among the system's varying levels of capability. Specificity is whether the trust involves part of the system or the whole system, and whether this changes over time or remains constant.

Finally, this framework treats trust as a closed-loop process. Trust is affected by whether or not developers can trust the automation, and in turn, reliance provides the foundation for how the trust is updated over time [27]. That is, if AI is reliable and always works, trust increases; if it does not work as the user wants, trust decreases. The framework acknowledges that a multitude of surrounding factors can affect this process, such as individual differences, organizational norms, and cultural differences [27], which connects to the previous section.

Lee and See's framework is relevant to RQ3, which focuses on the reliability and trustworthiness of LLM-based AI tools. These dimensions, calibration, resolution, and specificity provide a solid analytical foundation for this study. These dimensions can help assess whether developers' trust correlates with the capabilities of the tools they use, the processes they follow, and their purpose. Given the framework, a valid justification can be drawn from the information collected from our participants.

Lee and See's framework has a focus on individuals, but it can, as in this case, also provide insight into team dynamics as the interaction with automation, emphasizing context and feedback dynamics, can apply to a social setting as well. Complementing

the previous frameworks, STS and thematic analysis, and creating a unified image of this study.

5

Methods

5.1 Research Design

The study followed an empirical software engineering research approach with a mixed-methods design to investigate and evaluate the impact of LLM-based tools as technological artifacts [39].

A qualitative case study design was employed through semi-structured interviews using an interview guide presented in Appendix: B. The interview guide was derived from the research questions and divided into sections. Each section was designed to elicit responses relevant to the research questions, while remaining open and flexible to accommodate perspectives not anticipated. A first draft of the guide was tested with four test participants to verify that questions were unambiguous and likely to trigger responses useful for answering the research questions. The interviews were conducted on software developers who work on agile and DevOps teams and have incorporated AI into their daily workflows [40][41]. The interview data were analyzed using qualitative coding techniques to identify themes related to team decision-making, perceptions of roles, collaboration, code quality, and trust in AI-generated code [36].

To complement and contextualize the qualitative data, a Likert-scale survey was also administered, the questions and answers shown in Appendix: C. After collecting the data, descriptive statistics were used to identify quantitative results [42]. The survey was distributed to both the case-study participants after the interviews and to additional participants outside the case-study. The purpose was to collect quantitative data to identify patterns in developers' perceptions of workflow changes, collaboration with AI within their teams, and confidence in AI tools across different teams and industries.

5.2 Participants and samples

The study included 11 participants in the qualitative case study and 24 participants in the survey. The survey was distributed to both the case-study participants after interviews and to additional participants outside the case-study. As a subset of the survey consisted of the interview participants, the convergence between qualita-

tive and quantitative findings may reflect this overlap rather than fully independent strengthening. Participants were eligible if they worked in an agile or DevOps environment and, in some capacity, as a software developer. Additional qualifications include interaction or observations of LLMs in the work environment.

Participant demographics in the survey and interviews were limited to software developers, including juniors and seniors. Various roles in software development were included to capture a broad range of perspectives on the adoption of AI tools across different ranges of experience.

5.3 Procedure

The qualitative interview data were analyzed using Braun and Clarke's thematic analysis framework [36]. Following the interviews, all recordings were transcribed verbatim to ensure that citations and subsequent codes accurately represented the participants. From the transcriptions, key citations were highlighted and selected from each interview to represent participants' opinions. The highlighted parts in Appendix: A.1 were selected as they were interesting or relevant for our thesis.

The highlighted parts were then condensed and summarized into versions of the participants' citations in the form of codes. The coding was done iteratively to maintain the true nature of what was said without imposing interpretation, as shown by Braun and Clarke [36]. To ensure consistency and avoid the need for subsequent inter-coder checks, both researchers coded the data together, discussing and agreeing on codes and citations in real time. Once all codes were created, they were placed into an initial grouping in Miro [43], as presented in Appendix: A.2. This grouping was done to represent broader themes so that further processing would be more manageable.

Once the initial codes were established, they were replicated for each participant to whom they applied. Each instance was labeled with a participant's p-tag identifier in a color unique to the participant. The instances were weighted by a number corresponding to how often the code appeared in the participant's transcript. Within each theme, codes were grouped in vertical lines to clearly visualize both their frequency across participants and their weight per participant as presented in Appendix: A.3.

Once all participants' individual codes were established, final sub-themes were constructed from the broader groups and connected to a main theme. After this process, everything was reviewed again to ensure that the codes were actually representing what they had been connected to. During this iterative process, codes were moved, and themes were split, edited, added, or deleted until the iterative process yielded no substantial change. An example of a final version of a main theme with sub-themes, including codes, can be seen in Appendix: A.4.

Codes not relevant to the study but still of interest were saved for possible mention in the thesis. Codes neither relevant to the study nor worth mentioning were removed based on two questions: "Does it relate to any research question?" and "Is this

relevant to this field of study?" If both answers were no, the code was removed.

Findings from the qualitative analysis were triangulated, when applicable, with the quantitative data collected from the survey to assess convergence. Where qualitative themes aligned with quantitative patterns, this strengthened confidence in the findings. Where discrepancies were found, both data sources were examined further to understand the discrepancies.

5.4 Data analysis

Initial themes felt very broad in definition and were intentionally defined that way to accommodate different views and nuances, so they could later be split into subthemes. The main techniques used were based on Braun and Clarke's thematic analysis framework [36]. Quantitative data collected from the survey were organized using Google Forms and Google Sheets. For Qualitative analysis, primary coding was done in Google Sheets and then further processed in Miro [43]. Miro allowed for visual organization and iterative refinement of codes and themes.

Several strategies were implemented to ensure the trustworthiness of our findings. Credibility was improved through triangulation of qualitative and quantitative data sources through survey results where they converge. Both researchers used collaborative coding to ensure a shared understanding and an exact transcription of all interviews, thereby supporting correct analysis. Dependability was maintained through the systematic application of Braun and Clarke's framework and by documenting the analytical process in Miro, creating an audit trail of coding decisions. The ability to confirm themes was supported by using participant citations as validation.

5.5 Limitations and delimitations

The study is subject to limitations and categorized into external, internal, construct, and reliability threats to validity.

5.5.1 Limitations

Threats to external validity concern the generalizability the results [42]. This study was conducted over half a year, resulting in a lack of longitudinal impact analysis. Following the rapid evolution of LLMs was therefore not possible, limiting the temporal scope to which the results apply. The sample size was smaller and may not be fully representative; interviewing a large number of individuals across multiple companies and countries would be great, but hard to execute due to time constraints. Factoring in additional limiting factors, such as culture and unwritten rules, is not possible because it would take more time than is available.

Participants were taken from a wide range of companies, large, small, and startups, with differing experience levels. Despite this, the number of participants in each

sector is not balanced and again not large enough to make definitive claims. Subthemes like 6.3.2.3 involve a small number of individuals and should be treated as indicative rather than conclusive.

Threats to internal validity concern the trustworthiness of the cause-and-effect relationships within the study [42]. The limited time of the research (6 months) means that a higher quantity of data, such as the number of interviews and forms answered, can be hard to acquire. The ongoing evolution of LLMs and changes to people's perceptions may lead to replication problems. Furthermore, the success of the study depended on cooperation from companies and individuals, which are not fully controllable and may have led to self-selection bias.

The following of Braun and Clarke's six-step framework [36] is structured with the added factor of both researchers coding together. Despite this, the identification and naming of themes are interpretive and can, unfortunately, miss aspects. Generating codes from participants' transcripts can miss nuances that the participants did not intend. Even if the code represents what is said, it cannot always capture the participants' intent. The use of citations in the Results section helps mitigate this issue for readers and provides a more in-depth understanding.

Threats to construct validity concern how interviews and form questions are interpreted [42]. It is not possible to guarantee that what is asked is what is answered. Balancing open questions while steering the conversation in the right direction is important, as answers are needed to the questions, but leading participants to an answer would compromise the result. Different individuals interpreted questions differently; to mitigate this, follow-up questions and clarification during interviews were used. Construct validity was a concern and was addressed through careful word choice and test interviews, where it was possible to identify ambiguous questions. These flaws were then revised before the data collection began. For survey questions, they were created in parallel to best represent our interview questions.

The semi-structured interview allows participants to express their concepts on their terms. This has a side effect: while it is positive for construct validity, it also leaves room for interpretation when they use words like "trust" and "responsibility". Thus, even when clarification is provided during interviews, participants have different internal definitions of words. Another threat to the construct validity of our trust findings is social desirability bias. Cautious, calibrated stances toward AI are the publicly endorsed professional position in 2026, particularly within software engineering. Participants speaking to thesis researchers may have presented views more suitable to this endorsement than their actual behaviors during deadline pressure would reflect. Our findings should therefore be read as evidence on how software developers describe and reason about AI trust, not necessarily as evidence about how they behave in an isolated real-world situation.

Another threat specific to this study is based on the different LLM-based models used, as each participant did not use the same models. As seen in the interview in Table 6.1, a multitude of AI tools are used. The LLM-based AI tools differ in

modeling, interaction patterns, and intended use cases. This shapes participants not only by AI in general but also by which specific tool they use. Findings related to perceptions of role changes, descriptions of trust, or discussions of AI-assisted peer review are therefore also based on the collected experiences with different tools.

Finally, there is researcher bias; both researchers are interested in this topic and may subconsciously find patterns that align with their interests. This can affect which themes are more noticeable and emphasized. Which constant work in Miro and reflection of transcriptions, this was somewhat mitigated so as to reflect participants as much as possible.

Threats to reliability and validity concern whether the study can be replicated by another study [42]. We documented the detailed methodology, process, and results. Conclusion validity when reasoning about the quantitative results was met with clear reasoning and without motive [42]. To increase reliability, questions in interviews and forms were kept independent of companies and cultures. These types of questions led to increased reproducibility and thus reliability in the study.

5.5.2 Delimitations

The study focused on software development in combination with LLMs and AI tools, disregarding use outside the software development space. The scope was limited to the context of software engineering and thus excluded the use of LLMs in other fields.

The literature review focused closely on this field of study, with particular attention to studies related to socio-technical interaction, as this was one of the research aims. Based on the research gathered, most studies had focused on the individual without considering social interactions, a gap that this thesis aimed to address. The scope was further narrowed to interactions within agile and DevOps teams and the influence of AI assistance. A key objective was to gather information from prior findings and extend their results to teams.

Data collection primarily consisted of interviews and surveys, focusing on social constructs and the perceived influence of collaborating with AI, while disregarding ethical questions. Ethics were discussed in broader terms in relation to the thesis as a whole, but were not included as a research dimension. The conceptual boundary focused on practical workflow integration and trust-based collaboration, and how they affect the team.

5.6 Ethical Considerations

Established ethical guidelines were followed for the qualitative research on human participants, conducted in this study. Attention was emphasized on informed consent, anonymity, responsible data handling, and voluntary participation. Before each interview, participants were verbally informed about the purpose of the study,

the procedures, data handling, and their rights as participants. The participants gave their consent orally before the interview began, and for the survey, an equivalent statement was presented as a checkbox before they could proceed. Participation in the study was entirely voluntary, and participants were informed that they could decline to answer any questions or withdraw at any time without reason or negative consequences. To protect participants' identities, a coding scheme was implemented. Interviewees were assigned codes P1-P11, and survey responses were reported only in aggregate. In addition, company names or other identifying details were removed or generalized in transcripts and quotations. Audio recordings were stored locally on the authors' personal computers and were not uploaded to third-party services. After the thesis is examined and approved, all raw data will be deleted. Data was processed in accordance with the EU General Data Protection Regulation (GDPR), and no special categories of personal data were collected. A subset of the interviewees and survey respondents had prior professional connections to the authors. To mitigate the risk of bias or socially influenced responses, participants were reminded of their anonymity and of the study's independence from their employment relationship before the session began. The authors also reflected on this connection during the analysis to avoid bias toward familiar expressions and perspectives when constructing themes.

6

Results

6.1 Interview Results

From the interview results, 5 main themes were created. Each main theme represents an aspect essential to this study. An additional theme, important but not essential to our thesis, was also created. There were 11 participants in the interview and 24 respondents to the survey. The interview guide is presented in the Appendix: B and the results from the Survey questions: C.

Table 6.1: Overview of interview participants.

ID	Role	Team	Tenure (yrs)	AI Tools
P1	Senior Developer	10-15	5	Copilot, ChatGPT
P2	Junior Developer	5-10	0.5	ChatGPT
P3	Senior Developer/DevOps	10-15	4	Copilot
P4	Junior Developer	10-15	1.5	Copilot
P5	Scrum Master	5-10	0.5	Internal LLM
P6	DevOps Engineer	3-4	5	ChatGPT
P7	Senior Developer	2-3	0.5	Claude, Codex
P8	Senior Verification Engineer	10	4	Internal LLM
P9	Senior Developer	2	10	Copilot, ChatGPT
P10	Senior Developer/DevOps	20	2.5	Copilot
P11	Senior Developer/DevOps	15	4.5	ChatGPT, Claude, CodeRabbit

6.2 RQ1: Decision-Making and Role Distribution

The results connected to RQ1 indicate that the impact of AI on software developers' roles is perceived as ongoing rather than completed. While the tools are increasingly integrated into everyday workflows, their adoption is not expected to alter the fundamental roles or responsibilities. Participants instead describe the primary change as how tasks are performed, with AI acting as a supporting tool for both efficiency and decision-making.

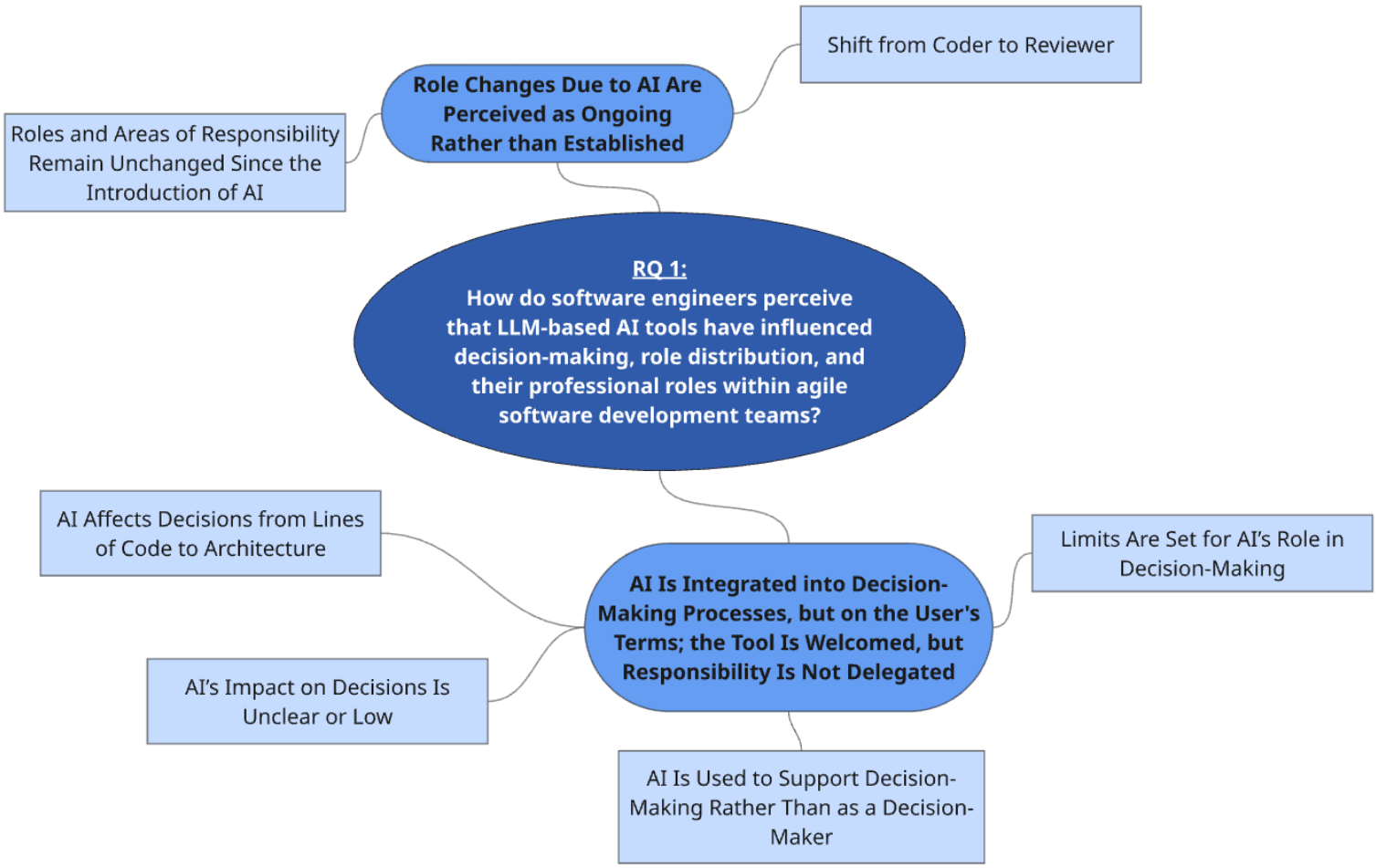


Figure 6.1: Thematic map connected to RQ 1

The thematic map was created to represent the connections to RQ1. It represents the two main themes that we constructed in relation to RQ1, along with the respective sub-themes.

6.2.1 Role Changes Due to AI Are Perceived as Ongoing Rather than Established

Across the interviews, participants described the adoption of AI tools as still in progress rather than complete. Participants reported that their formal roles and re-

sponsibilities remain unchanged, though they stated that day-to-day work processes have changed. Participants described their roles as having actively moved toward a code-reviewer role, although others expected this shift in the future. How this shift is being adopted was perceived to vary across experience levels, with different patterns of use observed between junior and senior developers.

6.2.1.1 Roles and Areas of Responsibility Remain Unchanged Since the Introduction of AI

Roles and responsibilities were described by as unchanged, although others noted that their day-to-day work processes had shifted since the introduction of AI tools. The distinction was that, although AI altered or enhanced how they performed their tasks, it did not affect their formal roles or responsibilities.

> *"It's more that AI acts as a kind of assistant. My job and my tasks have remained the same"* (P1)[1]

> *"Regardless of AI or not, my areas of responsibility haven't changed. However, AI solutions have, of course, helped me become more efficient in my role, which in turn has led to an expanded role in several ways."* (P2)[2]

These accounts illustrate a distinction between the definitions of the participants' formal roles and the shift in how they perform their tasks. While P2 described their role as "expanded" through greater efficiency, this was framed as an extension in the ability to perform tasks within the same role rather than a change to the role itself. Remaining participants described clear changes in their roles and responsibilities, and their accounts form the basis of the following sub-theme.

6.2.1.2 Shift from Coder to Reviewer

Participants reported having begun or nearly completed a shift from manually writing code to reviewing AI-generated output, while others reported expectations of a future shift in software development. The interview data is broadly reflected in the survey as seen in figure 6.2. Notably, no one selected "strongly agree," suggesting that even among those who noted a shift, it is not yet fully established. Together, these results illustrate an ongoing transition that is not fully completed.

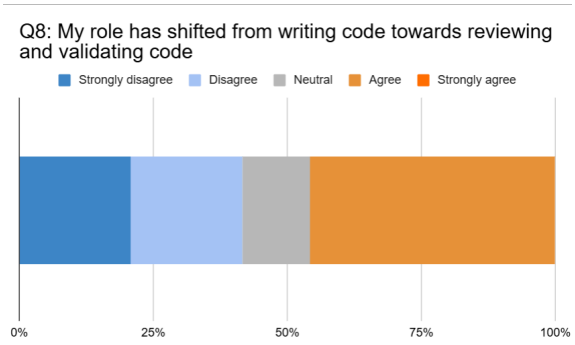


Figure 6.2: Shift from coder to reviewer

[1] "Det är mer att AI är lite av en hjälpreda. Mitt jobb och mina uppgifter har stannat den samma."

[2] "Oavsett AI eller inte, har mina ansvarsområden inte förändrats. Men sen så har ju AI-lösningarna såklart hjälpt mig att bli mer effektiv i min roll, vilket har lett till en utökad roll på flera sätt."

Of the interview samples, P7 described the highest degree of transition:

> *"Personally, I don't think I've written a single line of code since November or December. I use a combination of Claude Code and Codex"* (P7)[3]

Other participants had not reached a similar degree of transition but described a similar direction:

> *"I think it's increasingly about trusting the code and focusing more on validation."* (P4)[4]

P4's account frames the shift as both ongoing and expected, where validation is becoming the central task for developers. This sub-theme captures a pattern in which the perception is that the developer's role is shifting towards validating AI-generated output, even though this transition has not yet occurred in practice.

6.2.2 AI Is Integrated into Decision-Making Processes, but on the User's Terms; the Tool Is Welcomed, but Responsibility Is Not Delegated

This theme encompasses decision-making and the role of AI in it. From the data, we constructed a clear pattern in how participants implemented AI; it suggests they did so on their terms. Participants made decisions with some consultation from AI, but AI was not the decision-maker.

6.2.2.1 AI Affects Decisions from Lines of Code to Architecture

The results indicate that AI has influenced decisions across a wide range of areas. Participants describe AI being used to support programming, architectural, and strategic decisions. This was presented well by P2, who describes AI usage in their team:

> *"My colleagues in my team and I agree that we think it is a good tool for making decisions. Both technical and architectural, but also strategic decisions if you want."* (P2)[5]

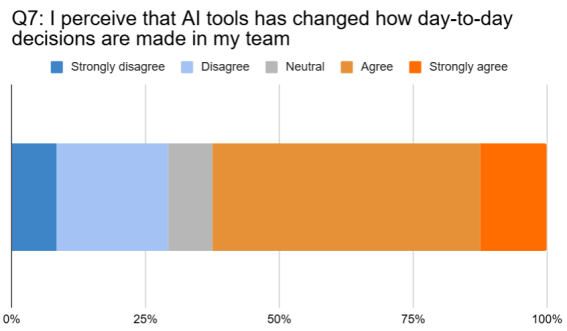


Figure 6.3: I perceive that AI tools has changed how day-to-day decisions are made in my team.

[3] "Personligen tror jag inte jag skrivit någon rad kod sedan november eller december, jag använder mig av en blandning av Claude code och Codex."

[4] "Jag tror det är mer och mer att man litar på koden och fokuserar mer på validering."

[5] "Jag och mina kollegor i mitt team är överens om att vi tycker att det är ett bra hjälpmedel för att fatta beslut. Både tekniska och arkitekturmässiga, men också strategiska beslut om man så vill."

Furthermore, participants mentioned that AI has been used to fill knowledge gaps in areas where individuals are less proficient, where the capabilities of AI exceed those of the developers, and this was described as influencing coding decisions that were made. P1 described this when using AI for generating SQL queries:

> *"It's great for writing advanced SQL queries that I would never have been able to create by myself"* (P1)[6]

These results were reflected in the survey, as shown in Figure 6.3, where a majority of the respondents (62.5%) perceived that AI tools have changed how decisions are made, but not all share this view.

6.2.2.2 AI's Impact on Decisions Is Unclear or Low

Uncertainty about whether AI has affected the team's decisions is a recurring pattern in the interviews. Participants reported that AI's effect on their decisions was either low, unclear, or nonexistent. P1 mentioned that the suggestions made by team members, leading to a decision, were unclear as to whether they stemmed from AI:

> *"It's difficult to know where they got their ideas from, whether they have been chatting with the AI or googled, I don't know"* (P1)[7]

Participants also perceived that LLM-based tools did not influence decisions at all. P4 mentions this when discussing the implementation of AI and its influence on decision-making:

> *"Nothing was really affected. We discussed it, but nothing was really affected."* (P4)[8]

These accounts illustrate limited visibility into colleagues' AI use, which obscures whether AI has shaped team decisions. AI's influence on the team is described as either invisible or insubstantial, in contrast to the more direct and open effects it has on decisions, as described in the previous sub-theme.

6.2.2.3 AI Is Used to Support Decision-Making Rather Than as a Decision-Maker

Participants described AI as a support for decision-making, with the developer responsible for issuing the verdicts. P3 describes this distinction:

> *"I get suggestions on how to solve things, so I use it all the time, as a basis for decisions."* (P3)[9]

[6] "Den är fruktansvärt bra att skriva avancerade SQL-frågor som jag aldrig skulle kunna göra själv."

[7] "Det svårt att veta var de har fått sina idéer från. Om det är för att dom suttit och chattat med en AI eller googlat, det vet inte jag."

[8] "Inget påverkades riktigt. Vi diskuterade det, men inget påverkades egentligen."

[9] "får ju förslag på hur man kan lösa saker, såsätt använder jag det hela tiden, som ett underlag för beslut"

Participants further describe how AI may suggest solutions the developer had not considered. P5 describes this scenario:

> *"AI tends to find obscure functions or implementations that I personally would never have thought of, ones that can make code surprisingly more efficient or improve security in ways you'd never have considered."* (P5)[10]

Another aspect that was reported is how AI accelerates development, leading to more ideas being explored. As a result, deciding which path to take was easier because they had more information in less time. The majority of participants use AI to aid them in decision-making, whether for coding or other group decisions. An example from P5 describes how AI can aid in decision-making:

> *"If you are discussing a problem and two people each have an idea, both seem to work about equally well or something. And you don't really know if either of them has any flaws in them, then it can often be the case that you send both ideas to an AI and ask it to look for pros and cons"* (P5)[11]

Across these accounts, AI was positioned as an input to decision-making rather than its source. It was used to broaden the options considered and accelerate the path to a decision, while the act of deciding was reported to remain with the developer or the team.

6.2.2.4 Limits Are Set for AI's Role in Decision-Making

Participants echoed the same sentiment about where responsibility lies when using AI in decision-making. There is a clear limit; whether or not AI has helped to make a decision, the product is fully the participant's responsibility. This is stated clearly by P6:

> *"Documentation or code, ultimately you are the one responsible, you can never blame an AI agent"* (P6)[12]

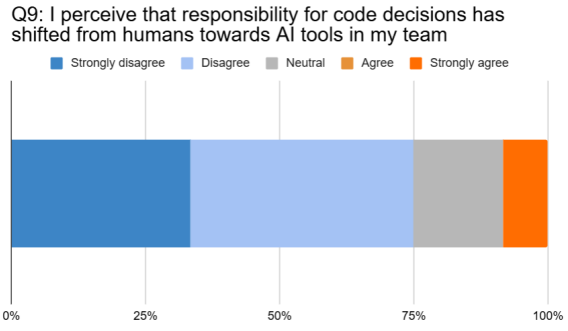


Figure 6.4: Responsibility for decisions

The limits of whether AI can replace human decisions are almost fully uniform across the accounts where P7 expressing the wish for AI replacing human decision making

[10] "AI har en tendens att kunna hitta obskyra funktioner eller implementeringar som jag personligen aldrig hade tänkt på som kan effektivisera kod förvånansvärt mycket eller öka säkerheten på något sätt man aldrig tänkt på."

[11] "Om man diskuterar om ett problem och då två personer har var sin idé, båda ser ut att fungera ungefär lika bra eller så. Och man vet inte riktigt om någon av dem har någon brist i sig då kan det någorlunda ofta i alla fall bli att man skickar in båda idéerna till en AI och ber den leta pros and cons"

[12] "Dokumentation eller kod så är det i slutändan man själv som har ansvaret, det går aldrig att skylla på en AI agent"

in certain areas:

> *"Yes, like, absolutely start replacing certain decision-making functions that involve making good decisions given x data."* (P7)[13]

The responsibility for code decisions is reflected in the survey, as seen in Figure 6.4, where a majority (75%) of the respondents did not perceive responsibility as having shifted towards AI. A comparison with Figure 6.3 highlights the distinction between decision-making and responsibility for those decisions: While most of the respondents agree or strongly agree that AI tools have changed how decisions are made, the majority disagree or strongly disagree that responsibility for code decisions has shifted from humans to AI. This contrast reinforces the qualitative theme that AI is integrated into decision-making processes while responsibility remains with the developer.

[13] "Ja typ absolut börja ersätta vissa beslutsfunktioner som handlar om att ta beslut på ett bra sätt givet x data."

6.3 RQ2: Collaborative Practices and Team Dynamics

The themes in this section indicate that the introduction of AI tools is perceived to be reshaping collaboration patterns and dynamics within software development teams. However, this is not entirely uniform; it is reported to redefine when and why collaboration occurs rather than fundamentally altering its nature. Parts of everyday human-to-human communication are shifting toward human-AI interaction, most evident in the growing tendency to consult AI before or instead of colleagues. Regarding code quality, the results suggest AI to be less of an independent driver of quality and rather an amplifier of existing conditions.

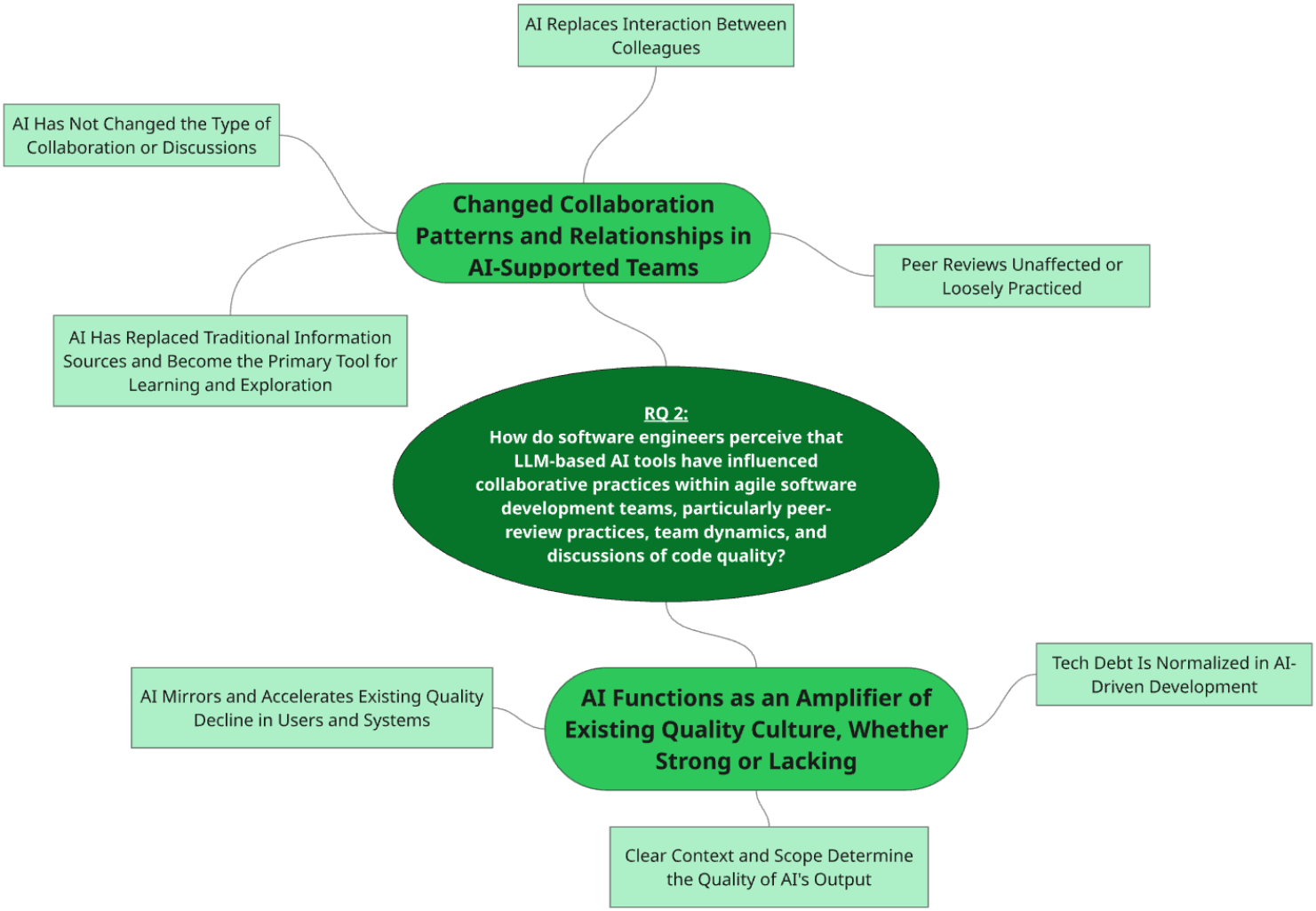


Figure 6.5: Thematic map connected to RQ 2

The thematic map was created to represent the connections to RQ2. It represents the two main themes we created in relation to RQ2, along with their respective sub-themes.

6.3.1 Changed Collaboration Patterns and Relationships in AI-Supported Teams

The results in this theme suggest that AI's effect on team collaboration is non-uniform. AI is reported to have substantially displaced human-to-human interaction in team consultations and reliance on traditional sources of information. The common pattern across the sub-themes is that AI has changed when and with whom collaboration occurs more than the collaboration itself.

6.3.1.1 AI Has Not Changed the Type of Collaboration or Discussions

Participants reported that AI tools have not affected the way collaboration or discussions take place within their team environment. A related view was that, despite AI's expanding role in their workflows, it could not replace human-to-human discussions regarding deeper insight or contextual understanding. A participant, P9, raised a specific concern that AI is unable to push back when needed and instead tends to affirm the user's statements.

> *"I don't want to ask it 'am I right about this' or 'is this a good thing?' because I see it as a risk that it will say it's a good thing regardless."* (P9)[14]

This indicates that AI has been integrated into collaborative practices without displacing them. However, while this distinction of AI as a complement to human interaction rather than a substitute is not universally maintained, as the next sub-theme shows.

6.3.1.2 AI Replaces Interaction Between Colleagues

Contrary to the previous sub-theme, participants described changes in interactions, in which human-AI is increasingly substituting for human-to-human discussion and collaboration. The patterns created within this sub-theme are grouped into three related categories: consulting AI instead of colleagues, reduced communication between colleagues, and changes in how junior and senior developers interact.

The most reported change was that participants consult AI before or instead of colleagues. Participants described consulting AI tools instead of colleagues, where three further described consulting AI as a first step:

> *"There are many examples where AI can fill that function of being able to answer a question that otherwise would have had to ask someone. Or had to, but out of lack of other quick means to turn to when you're stuck on something."* (P2)[15]

[14] "Jag vill inte fråga den, har jag rätt i det här så? Eller är det här en bra sak? För då ser jag det som en risk att den kommer säga att det är en bra sak oavsett."

[15] "Det finns många exempel där AI kan fylla den funktionen att kunna svara på en fråga som jag annars hade varit tvungen att ställa. Eller varit tvungen, men man hade av brist på andra snabba medel att ta till när man sitter fast med någonting."

> *"It's a bit individual, but I do think you probably start with the AI, and if it doesn't solve the problem, you go to a colleague."* (P6)[16]

This pattern was reciprocated in Figure 6.6, where a majority of the respondents (58.3%) agreed or strongly agreed that they first turned to an AI instead of a teammate when unsure. This in contrast to those who disagree or strongly disagree. This correlates with the interaction between colleagues decreasing, as seen in these results.

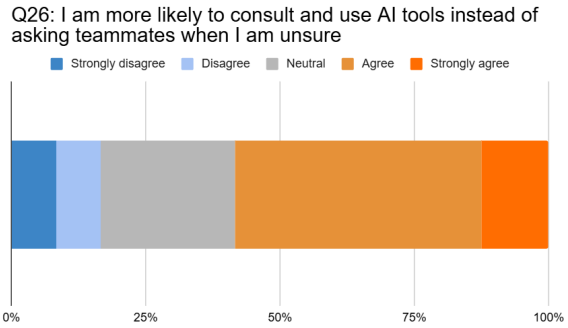


Figure 6.6: Consulting teammates' relation to AI

This shift was further associated with reduced communication among colleagues, as participants reported a perceived decrease in discussion, and two specifically reported fewer discussions of implementation details since the adoption of AI tools. P11 captured this:

> *"People have become a bit more self-sufficient. For better or for worse, there's actually less sharing."* (P11)[17]

The change was framed in two ways. On the one hand, describe AI as a means to free colleagues from being interrupted with simple questions, allowing them to work uninterrupted:

> *"Maybe it's the case that you get help from colleagues when it's truly needed, but before, you got help and took up their time for small things that AI can help with now."* (P4)[18]

On the other hand, noted reduced interaction between junior and senior developers as junior developers appeared to prefer AI. P10 described that the choice may be influenced by how approachable the senior developers appear, suggesting interpersonal dynamics play a role in the shift:

> *"If the reaction is good and the senior developer is easy to talk to and open, then the junior is probably more comfortable going to the senior and asking. At the same time, I can imagine that you probably don't want to be too much of a bother and ask a lot of questions all the time and feel like you're disturbing them and taking up a lot of their time."* (P10)[19]

[16] "Det är lite individuellt men det tror jag nog absolut att man börjar hos AIn och om den inte löser problemet går man till kollegan."

[17] "Folk har blivit lite mer självgående. På gott och ont, så det är ändå mindre sharing faktiskt."

[18] "Det kanske är så att man får hjälp av kollegorna när det verkligen behövs, men innan fick man hjälp och tog deras tid för småsaker AI kan hjälpa med nu"

[19] "Om reaktionen är bra och senioren är lätt att snacka med och öppen så är junioren nog mer bekväm att gå till senioren och fråga. Samtidigt kan jag tro att man vill nog inte vara för jobbig

Across these accounts, AI was described as replacing a portion of everyday interactions among colleagues in which consultation and knowledge sharing previously occurred. This was either framed as a benefit in time savings for senior developers or as a concern about junior developers bypassing the contextual knowledge of the codebase held by their colleagues, and this framing involved interactions among colleagues, in which consultation and knowledge sharing had previously varied across participants.

6.3.1.3 AI Has Replaced Traditional Information Sources and Become the Primary Tool for Learning and Exploration

A strong pattern was the reported displacement of exploration through traditional sources of information by the introduction of AI tools. Participants reported that AI has replaced sources such as Stack Overflow, search engines, and documentation as the go-to resource of knowledge when finding information or solving tasks:

> *"You get the right solution, or a solution that works, faster. When you otherwise would have had to go back to other references, maybe academic ones, or ask around in various online forums, or phone a friend."* (P2)[20]

Beyond the use for looking up information, participants described AI as a tool for tackling advanced tasks that were described as either blocking, due to exceeding the competence level of the user, or requiring significant time investment:

> *"I sat with it for a long time myself trying to write that algorithm, but never really managed to. I probably would have gotten it if I had kept going, it's not an impossible algorithm. But that's where we asked ChatGPT. The earlier models couldn't do it, but the latest model spat out something that actually worked."* (P9)[21]

Learning through AI was specifically emphasized for juniors. Participants described AI as useful for less-experienced developers when learning the codebase, enabling context-specific questioning and explanations:

> *"I think AI has been useful for those who have come in. I think they use it a lot to understand the codebase."* (P1)[22]

och ställa en massa frågor hela tiden. Och känna att man stör och tar mycket av deras tid."

[20] "Det går snabbare att komma fram till rätt lösning eller en lösning som funkar. När du annars hade behövt gå tillbaka till andra referenser, kanske akademiska eller fråga runt i olika online-forum eller ringa en vän."

[21] "Den satt jag själv med länge och försökte skriva den algoritmen, men lyckades faktiskt aldrig riktigt. Jag hade nog fått det om jag fortsatt så, det är ju ingen omöjlig algoritm. Men det var ChatGPT där som vi frågade. De tidigare modellerna klarade inte det, men den senaste modellen spottade ut sig någonting som faktiskt funkade."

[22] "Jag tror AI har kommit till nytta för dem som har kommit in. Jag tror de använder det jättemycket för att förstå sig på kodbasen."

AI was described as having taken on the role of multiple sources, such as documentation, search engines, and online forums, or as consolidating the expertise of more experienced colleagues. Accumulating these into one single point of access for information, exploration, and learning.

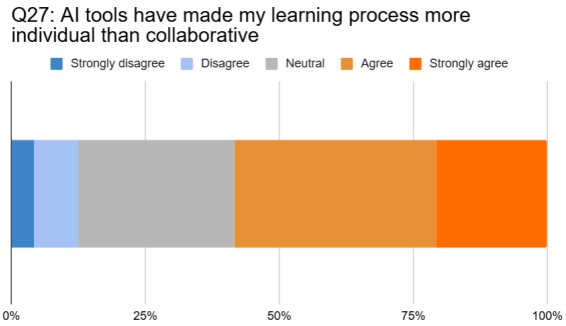


Figure 6.7: Learning processes more individual

The results from the interviews are reflected in the survey as seen in Figure 6.7. The shift towards AI as a way of learning and exploring can be seen when a majority of the respondents agreed (37.5%) or strongly agreed (20.8%) that AI tools had made learning processes more individual rather than collaborative.

6.3.1.4 Peer Reviews Unaffected or Loosely Practiced

In regards to peer review practices, a mixed picture was reported. Participants described no perceived change in peer review practices since the introduction of AI tools. P2 captured this well by noting that apart from productive benefits, the process continued largely as before:

> *"I wouldn't say that the process as a whole has changed, but rather that AI comes as an addition and aid to become more productive."* (P2)[23]

Furthermore, P4 also described a lack of peer review, framing it as a process that had not been implemented even before the introduction of AI tools. Together, these accounts suggest that existing weaknesses in peer review practices have remained largely unaffected:

> *"The code reviews were more that you trusted each other. There wasn't a whole lot of review on everything you pushed in."* (P4)[24]

Participants further described seeing a potential for AI in peer review. Individuals working in start-up environments had begun implementing AI-assisted reviews informally. However, the experiences highlighted different dynamics that described how the introduction of an automated review tool led to a decline in human engagement with the peer review process.

> *"We were very lax about it in the team before, and then when someone added CodeRabbit for automatic code review, people stopped caring entirely, because then reviews were coming in."* (P11)[25]

[23] "Jag skulle inte säga att processen i sin helhet har förändrats, utan snarare att AI kommer som ett tillskott och hjälpmedel för att bli mer produktiv."

[24] "Code reviewsen var mer att man litar på varandra. Man hade inte supermycket review på allt man pushade in."

[25] "Vi var väldigt slappa i teamet på det innan, och sen när någon la till Code Rabbit för

Another participant (P7) described a similar informal adoption in which code was reviewed by the AI rather than by team members, framing it as a choice driven by the need for speed in a startup context. This suggests that peer review remains largely human-driven for most teams in the sample, yet where AI has entered the peer review process, it has done so informally.

6.3.2 AI Functions as an Amplifier of Existing Quality Culture, Whether Strong or Lacking

Participants described AI's effect on code quality within their teams as conditional rather than uniform. Whether the quality was degrading, improving, or maintained was reported to depend on the context provided to the AI, the developers' competence in evaluating the generated code, and the existing code quality culture reflected in the codebase.

6.3.2.1 AI Mirrors and Accelerates Existing Quality Decline in Users and Systems

Participants reported that the risk of AI-assisted development is inversely related to the developer's own knowledge for reliably evaluating the output. Others framed this as a competence dependency, in which AI may cause problems if in the hands of developers least able to evaluate its output. P6 captured this:

> *"On one hand, I believe that if you trust the AI too much and do things you do not fully understand, it can cause problems in some situations. On the other hand, you might test things you are inexperienced in and develop your skills."* (P6)[26]

Depending on roles, the approach towards AI varies. P11 described the difference in how junior developers tend to approach AI more uncritically compared to senior developers, with the consequence of important decisions, such as the accumulated technical debt or the cost over time, being bypassed in the process:

> *"I would say that the more junior people are in the work environment, the more blindly they trust it. And they have a harder time understanding the consequences(...) I would say junior developers skip these decisions to a greater degree, things they maybe should have brought up with a senior colleague. Even if it works functionally."* (P11)[27]

The dependency between code quality and the user's competence was further re-

automatisk code review, då slutade folk bry sig fullständigt, för då kom det in reviews."

[26] "Dels kan jag tycka att om man litar på AIn för mycket och gör saker man inte helt har koll på vilket kan ställa till det i vissa situationer å andra sidan kanske man testar saker man inte har full koll på och utvecklas"

[27] "Jag skulle säga att ju mer juniora folk är i arbetsmiljö, litar man mer blint på det. Och man har svårare att förstå konsekvenserna(...) Jag skulle säga att de mer juniora i teamet skippar de här besluten i lite större utsträckning, som de kanske skulle ha tagit med typ en seniorkollega. Även om det här funkar rent funktionsmässigt."

inforced, explicitly framing AI's role in code quality as a function of the user. P4 stated this perception clearly:

> *"I think a good developer would be able to maintain or improve the code quality, but an unskilled developer would possibly degrade the code quality."* (P4)[28]

This framing was echoed by P7, who relocated the source of bad code quality away from the model:

> *"The models definitely produce better code, then they can also produce worse code, but then I think it is a user error"* (P7)[29]

Across the samples on code quality, the introduction of quality problems was primarily attributed by participants to users' deficiencies and model misuse. AI was primarily viewed as a multiplier of these existing issues, enabling developers who were previously producing technical debt to snowball their output at a greater volume.

6.3.2.2 Clear Context and Scope Determine the Quality of AI's Output

Participants noted that AI struggles with complex tasks and further highlighted that the AI's output is conditional on the clarity of the provided context. If the context is too complex or too large, AI struggles to understand what should be done, and the quality drastically reduces. P10 framed this nuance:

> *"So I think the more context you give it. The better it can be."* (P10)[30]

This limitation was described in the context of large or inconsistent codebases, which participants noted as a barrier to using AI effectively in their work. P10 further illustrated this limitation through a concrete example:

> *"There are so many people that have worked in it without really thinking about the structure or really caring about it, it was more: 'If it works, it works'. We do not really care about the quality, so to speak. We talked once about trying to use AI to see if it could help us restructure and get a better codebase. But it was not possible to use AI for it because it was too complex."* (P10)[31]

Beyond size, unique solutions that did not follow industry standards were described

[28] "Jag tror att en bra utvecklare hade kunnat bibehålla eller förbättra kodkvaliteten, men en dålig utvecklare hade kunnat försämra kodkvaliteten."

[29] "Definitivt att modellerna producerar bättre kod, sen kan dom producerar sämre kod, men då tycker jag det är ett användarfel."

[30] "Så jag tror ju mer kontext du ger den. Desto bättre kan det bli."

[31] "Det är så många som har jobbat i den och man har inte riktigt tänkt på strukturen och brytt sig, utan det var mer: "funkar det så funkar det". Man bryr sig inte om kvaliteten om man säger så. Vi pratade en gång om att vi ska försöka använda AI och se om den kan hjälpa oss strukturera om och få en bättre kodbas. Men det gick inte att använda AI för att det var alldeles för komplext."

as limiting the AI's ability. These team-specific conventions and solutions in the codebase were reported as requiring a developer to bridge the gap between the AI output and team standards to correctly implement them. P4 described this sharply:

> *"This is not knowledge you can google, but is shared within the company and is not completely original design patterns, but every work place have their own versions of patterns, I would say. AI does not understand this, so you need a person to implement this according to how the company works."* (P4)[32]

This sub-theme connects to the previous sub-theme by illustrating the quality of the AI's output as a function of the user's input. The accounts suggest that AI does not independently produce quality but rather multiplies the quality of the input it receives. Where context was thin, the quality of the output suffered, while in the opposite case, the output was reported to be more useful.

6.3.2.3 Tech Debt Is Normalized in AI-Driven Development

The results in this sub-theme are grounded in a smaller subset of the data, working in start-up environments. While the evidence is limited in this theme and not generalized across the data, the convergence from this data in an organizational perspective makes the analysis distinct and worth reporting in its own right. Participants reported that the accumulation of tech debt is inherent in start-up work environments. This accumulation is, however, described as being accelerated with the use of AI, as P11 described:

> *"It is also a trade-off in start-ups, it has always been the case that we only need to build sites; we can solve the tech debt later. But the problem is that when you create things 50 times faster, it accumulates very fast. The more spaghetti you already have in the codebase, the more spaghetti the LLM will produce, so it accumulates very quickly."* (P11)[33]

This view was shared by P7, however, with another framing of the accumulation being offset by a similarly increased rate of remediation:

> *"But if the tech-debt increases at the same rate as the cost of solving it decreases, then it evens out, you might generate 3x tech-debts, but you can solve it 3x faster, so we are in the same spot."* (P7)[34]

[32] "Detta är inte något man kan googla och följa en manual och implementera utan kunskap som delas inom bolaget och är inte helt orginella mönster, men varje arbetsplats har sina egna versioner av mönster skulle jag säga, och AIn fattar inte detta så man behövde en människa som implementerade enligt hur företaget jobbar"

[33] "Men sen är det också en avvägning att göra i startups, så det har alltid varit att vi bara behöver bygga sidor; vi får lösa tech-debten sen. Men problemet är att när man skapar grejer 50 gånger snabbare så ackumulerar det extremt fort. Ju mer stök och spagetti du redan har på kodbasen, desto mer stök och spagetti kommer LLM att bevara, så det ackumulerar väldigt snabbt."

[34] "Men om tech-debten ökar i samma hastighet som kostnaden av att lösa det minskar, så det 'evens out', man kanske genererar 3x tech-debts men man kan lösa det 3x snabbare, så vi är i

The participants converge on the view that tech debt is an expected but manageable cost rather than a quality failure. Whether this suggests a sustainable strategy or defers the problem onto later maintainers, as P11 framed, that someone eventually will have to "deal with the poop sandwich"[35] captures an unresolved dimension to this perspective.

samma spot."

[35] "Sen får man någon som har köpt det där lösa bajsmackan."

6.4 RQ3: Trust and Reliability

The results in this section indicate that trust among developers was described as not a dichotomous condition but a dynamic, actively managed process. Rather than expressing trust and distrust as absolutes, it was described as something continuously calibrated based on verification practices, experience, and context. Trust shifted from the AI itself to the developer's ability to validate the output.

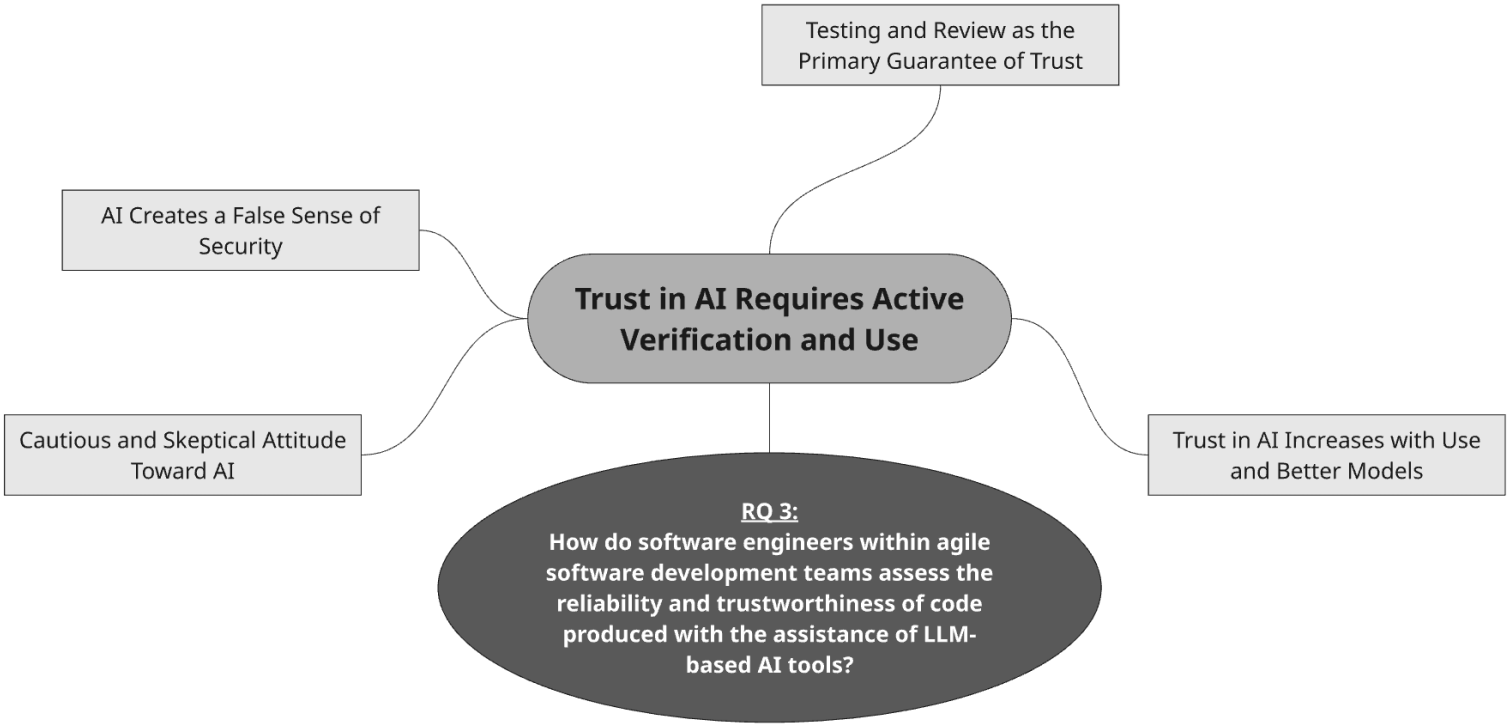


Figure 6.8: Thematic map connected to RQ 3

The thematic map was created to represent the connections to RQ3. It represents the one main theme that we created connected to RQ3, with the respective subthemes.

6.4.1 Trust in AI Requires Active Verification and Use

Trust in AI was not described as a static property across the samples, but instead described as something actively constructed through verification and experience. Rather than using absolute terms such as either trusting or distrusting AI, participants offered descriptions of reasons not to blanket-trust AI, and explained how trust is a more differentiated and context-specific calibration process. Through this process, testing and reviewing AI-generated output was found to be the primary mechanism for verifying its trustworthiness.

6.4.1.1 Cautious and Skeptical Attitude Toward AI

In general, the participants described their stance toward AI as deliberately cautious. Participants framed it as an ongoing transition where domains appropriate for trusting AI are still being mapped out, rather than blindly assumed. P8 described this directly:

> *"We are exploring right now because we do not want to use AI in situations where it does not give us anything, so we are exploring where we can use it and where it is appropriate to use it."* (P8)[36]

P9 reported a similar position where it was emphasized that openness to new technology should not necessarily be confused with uncritical adoption:

> *"You should not use it just because you can, it feels like there is a lot of hype around it."* (P9)[37]

This caution was further expressed in broader skepticism toward new technology. P5 noted a cyclical pattern in colleagues' trust towards AI over time:

> *"It feels like there was a period right at the beginning when people trusted it blindly, and then people became quite skeptical, and now it feels like people trust it blindly again."* (P5)[38]

Skepticism toward AI was particularly pronounced in relation to test generation, which the participants described as unreliable.

> *"It generates tests that pass things even though they should not."* (P3)[39]

The cautious stance toward AI was not generalized distrust but rather a case-by-case approach that demanded evidence on whether it was contextually appropriate and useful.

6.4.1.2 AI Creates a False Sense of Security

Beyond the stated caution, participants reported that concerns about misplaced confidence when generating a high volume of code were mistaken for quality. P11 illustrated this in the context of using AI for testing:

> *"That you think something is well tested, just because you get a bunch of tests spat out by the LLM. But then you look at it and all five of these tests tested exactly the same thing."* (P11)[40]

[36] "Vi utforskar just nu, för vi vill inte använda AI i situationer det inte ger oss nåt så vi utforskar var vi kan använda det och var det är lämpligt att använda det."

[37] "Man ska inte använda det bara för att man kan, det känns som det är så mycket hype kring det."

[38] "Det känns som det var en period där precis i början när folk litade blindt på det och sen blev folk ganska skeptiska och nu känns det som att folk litar blindt på det igen."

[39] "Den genererar tester som går igenom trots att den inte borde det."

[40] "Att man tänker att något är väl testat, bara för att man fick en massa tester utspottade av

Another concern identified by participants was the perceived rhetorical confidence AI showed in its language, as well as its tendency to agree with the user to convince them of its correctness, rather than actually being correct. This disproportion between rhetorical confidence and actual correctness was reported to be a risk if the user were not sufficiently skilled in the subject to identify the discrepancies. P4 captured this risk sharply:

> *"The less you know, the less you can validate the AI. It is definitely a big risk. I do not think the AI tries to be right, just to convince you that it is right. It is easier to be fooled when you are ignorant."* (P4)[41]

P11 further described the inclination of AI to capitulate to the users' opinions and suggestions rather than maintaining a position under pushback, which made it less trustworthy to be used as a sounding board:

> *"It is very difficult for modern LLMs to have any form of confidence level in what they are doing. That is why you get that, oh, of course you are right, when you have said something that you have been a bit pushy about. It is very easy for it to turn its coat with the wind."* (P11)[42]

Across the accounts, it was not AI being wrong that was considered dangerous, as this was largely expected by the participants, but rather that it was structured to discourage detection when wrong.

6.4.1.3 Testing and Review as the Primary Guarantee of Trust

Given the risks participants were describing, testing and code review were consistently described as the foundation of trust in AI-generated code. Participants framed testing as a mechanism through which AI-generated code ultimately could be trusted. P10 described how the verification through testing practically absorbs the output from the AI into the developer's own:

> *"If I use AI, I always double-check. I do check and test it properly. So it almost becomes as if I wrote that code myself."* (P10)[43]

The view that trust is based on one's own review process was the most widely shared position across the samples, articulated by the participants, where P4 stated it the most directly:

> *"I do not think you trust AI code. If you are in a position where you trust*

LLM:n. Men så kollar man på den och bara: "Alla de här fem testerna testade exakt samma grej."

[41] "Ju mindre du kan, desto mindre kan du validera AI:n. Det är definitivt en stor risk. Jag tror inte AI:n försöker ha rätt, bara övertyga att den har rätt. Det är lättare att bli lurad när man är ignorant."

[42] "Det är väldigt svårt för moderna LLM att ha någon form av confidence level i det de ska göra. Det är därför man får den där, oh of course you're right, när man har sagt något som man har varit lite dryg med. Den är väldigt lätt att vända kappan efter vinden."

[43] "Använder jag AI så dubbelkollar jag alltid. Jag kollar ju och testar av det ordentligt. Så det blir nästan som att jag skrivit den koden."

> *AI code, then you have not reviewed the code. If you review the code, then you do not trust the AI code, but rather that you yourself have done a good review of the code. So the code is code."* (P4)[44]

From this view, the question is not whether AI code can be trusted but rather a question whether the surrounding verification practices are adequate. Participants echoed that manual oversight remains necessary. Trust is not granted to the AI, but something built on the developer's own verification processes.

6.4.1.4 Trust in AI Increases with Use and Better Models

Although the participants described being broadly cautious about AI-generated code, they also reported that their trust in AI's capabilities and its ability to be correct has evolved over time. From this, we constructed two drivers: The improving capabilities of the models themselves and the amount of accumulated experience the developers had with AI.

The participants reported that successive AI model generations have expanded their support and technological capabilities, leading to higher trust in AI. P2 described that an increase in trust towards AI is tied directly to both the expanding model capabilities and their own increasing fluency with the tool:

> *"It can handle more and more context and can produce better results. There, my trust has increased because it has developed. My trust has become greater because I have used it more and can use the tool better."* (P2)[45]

On the AI-using side, participants reported that repeated use increased trust in the tool, with P7 framing it as an incremental process in which familiarity has grown, but uncertainties remain about how to use the tool effectively. Very few of the respondents reported a lack of increase in trust over time, while a large majority represented an increase in trust from slightly to extremely, as shown in Figure 6.9. The most analytically significant aspect of this growing trust came from P8, which described trust as evolving in more nuanced and differential ways the more it is used:

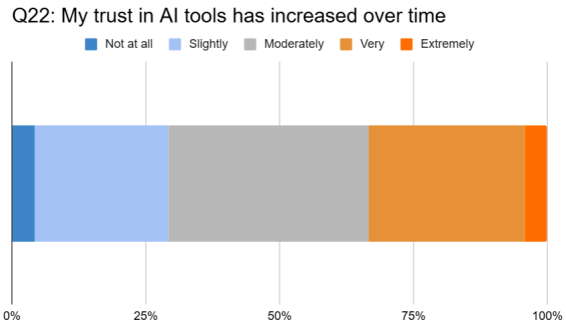


Figure 6.9: Increased trust over time

> *"The trust or the limitations become more clear the more you use AI, and*

[44] "Jag tror inte man litar på AI-kod, om man är i en sits att man litar på AI-kod, då har man inte granskat koden, om man granskar koden då litar man inte på AI-koden utan att man själv har gjort en bra granskning av koden, så koden är kod."

[45] "Den kan hantera mer och mer context och kan producera bättre resultat. Där har min tillit ökat eftersom den har utvecklats. Min tillit har blivit större eftersom jag har använt den mer och kan bättre använda verktyget."

> *therefore trust decreases in some areas the more you use it, but there are certainly also some areas where trust increases the more you use it. It feels like people who have not used AI much have very great trust in it, while those who have used AI know its limitations and do not trust it in cases where it cannot be fully trusted."* (P8)[46]

This reframes the relationship between the use and trust. Rather than experience appears to calibrate trust rather than increase it monotonically. Experienced users trust AI in some domains but less in others, whereas inexperienced users tend to trust it indiscriminately. Aligned with earlier sub-themes, this positions the cautious stance toward AI initially described not as a lack of trust but rather as its mature form. This form is granted selectively, supported by verification and ongoing use.

[46] "Tilliten eller begränsningarna blir mer tydliga ju mer man använder AI känner jag och därför sänks tilliten på vissa områden ju mer man använder det men säkert också vissa områden där tilliten ökar ju mer man använder det, det känns som folk som inte använt AI så mycket har väldigt stor tillit för den medans dom som använt AI vet dens begränsningar och litar inte på den i fall där det inte går att lita på den helt."

6.5 Additional themes outside of the RQ scope

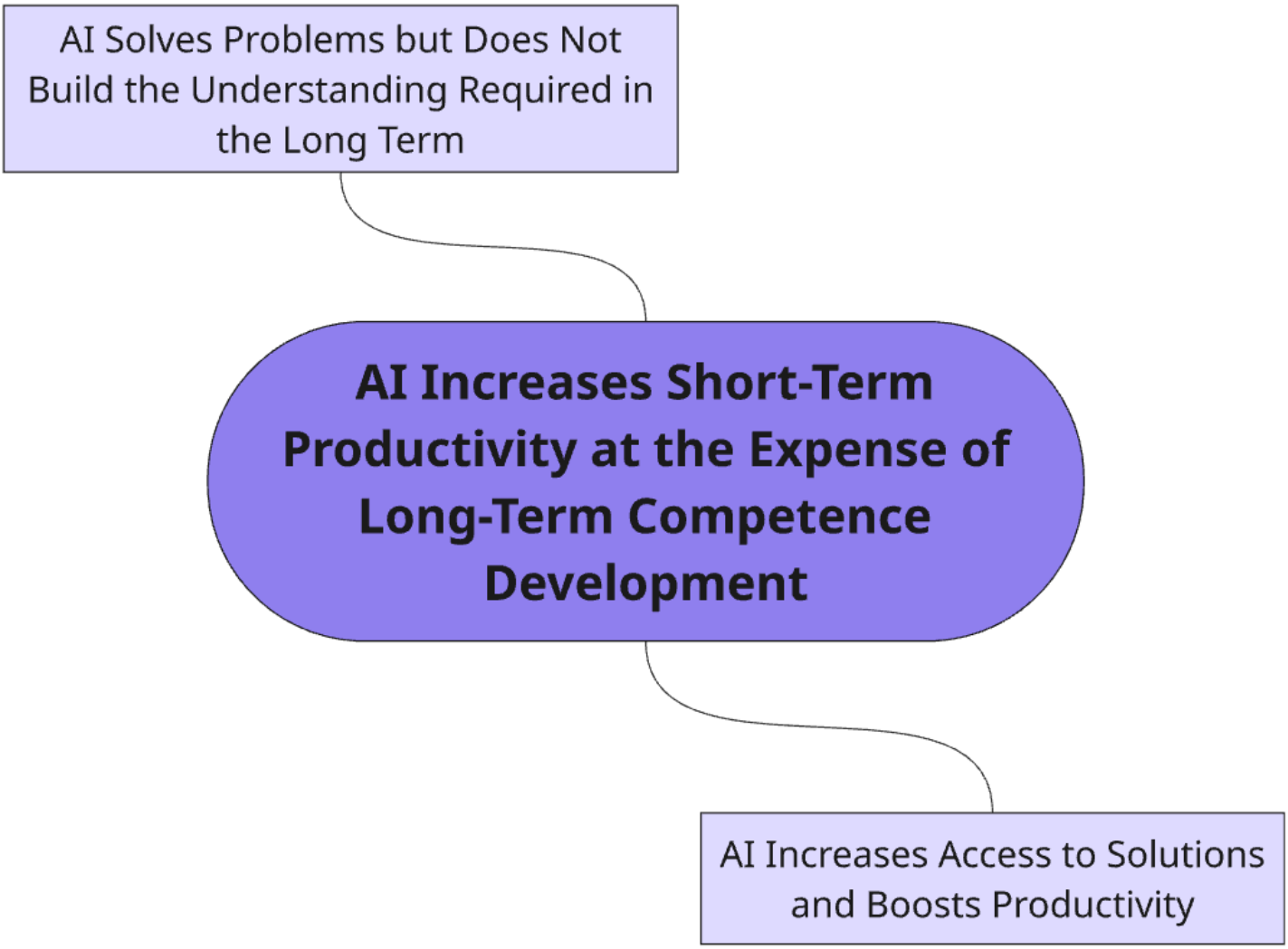


Figure 6.10: Interesting themes not connected to RQ

6.5.1 AI Increases Short-Term Productivity at the Expense of Long-Term Competence Development

We constructed this theme from the interview data, as it provides context that helps to better understand how the participants think. Participants noted that, from their perspective, productivity has increased, but at the expense of personal development. They also discuss that AI has made access to solutions easier.

6.5.1.1 AI Solves Problems but Does Not Build the Understanding Required in the Long Term

Participants reported an increased risk of solving problems without understanding. Furthermore, a sense of quick but shallow understanding regarding learning was also mentioned. The risks involved with this were concerns about over-reliance on AI. P2 gives an example of this risk:

> *"If you have relied on AI to a high degree when you have done something, there is a very high probability that it will be more difficult to debug the code yourself.".* (P2)[47]

Participants raised a concern that also touches on the long-term development of a broader understanding of the codebase. P4 and P11 described how relying on AI can reduce understanding of the codebase and lead to a loss of familiarity with the system's workings. This concern was especially evident in P11, who emphasized the need for junior developers to build knowledge and contextual understanding over time:

> *"Juniors become productive much faster, but it takes longer for them to gain a fundamental understanding of the important design principles. For example why things lead to nastier bugs, and the costs of technical debt become harder to grasp. You simply end up further from the code.".* (P11)[48]

Across these samples, AI was described as enabling participants to solve problems faster than they could understand them. Participants noted that this gap might be invisible in the short term, surfacing later when debugging or reasoning about the code that was not fully grasped, without the AI becoming necessary. The strongest concern was best voiced by P11 around junior developers, who perceived the absence of struggle through experimentation as bypassing the learning through which intuition and architectural understanding are built.

6.5.1.2 AI Increases Access to Solutions and Boosts Productivity

An overall positive perspective on AI is evident, with participants reporting increased productivity when using it. The increased speed when accessing information was consistently mentioned by the participants, describing that documentation, browsing forums, or consulting colleagues could be done faster with AI. P3 captures this mentality well when saying:

> *"It's more common to check things out in AI. It's a little faster than*

[47] "Om du har förlitat dig på AI till en hög grad när du har gjort någonting så är det en väldigt stor sannolikhet att det blir svårare att själv felsöka koden."

[48] "Det går mycket fortare för juniorer att bli produktiva, men det tar längre tid att få grundläggande förståelse för de viktiga designprinciperna. Varför saker leder till jobbigare buggar och kostnaderna av tekniska skulder blir svårare att förstå. Man kommer längre ifrån koden helt enkelt."

> *before when you googled around and didn't always get an answer. In AI you get an answer right away.".* (P3)[49]

Productivity gains were reported by the participants. These benefits were described in unison with the previous theme and did not contradict the results seen. The trade-off that AI entails is something participants see and do not brush aside. They note that short-term productivity is evident, but long-term effects remain a concern.

[49] "det är vanligare att man kollar av saker i AI. Det går lite fortare än förr när man googlade runt och inte alltid fick svar. I AI får man ju något svar direkt. "

7

Discussion

7.1 Discussion of Findings

7.1.1 RQ1: Decision-Making and Role Distribution

The results indicate that integrating LLM-based AI tools into the workflow has affected decision-making and participants' perceptions of their roles. From this result, we can see that the shift is still in development and not yet fully established.

From the results, we can see that formal roles and responsibilities are unchanged, while a smaller number of participants report that some shift has occurred or will occur in the future. These changes were described as moving toward more code review rather than coding, a shift noted in recent literature [2][30]. Bruhin's results and analysis make a point that traditional development tasks are becoming less in demand, and the restructuring of validation and oversight is a part of the AI integration [30]. Our study has made similar findings, where individuals note that validation is important when using AI. Furthermore, our results show that this transition is not always clear, but that many have described changes in their daily work without changes in their formal roles. This suggests that role changes are slow to adapt to AI, and this gap between practical and formal role change can have consequences that extend beyond methodological interest. Performance evaluations, hiring criteria, career progression frameworks, and training budgets in most organizations are calibrated against formal role definitions. If the practical work of software development has shifted toward validation and review, while formal roles continue to describe code production, organizations may be measuring developers against criteria that no longer reflect their daily work. Read through a socio-technical lens [37], this is a textbook case of technical-subsystem change outpacing social-subsystem adaptation. The tools and tasks have changed, but the roles, norms, and recognition structures around them have not. The risk of joint optimization failing here may possibly manifest in mismatched performance reviews, unclear accountability, and undervalued work. The gap between practical and formal changes warrants further study.

Decision-making on users' terms is observed by Khojah et al. [9], who describe how developers use AI as a virtual colleague and for decision support rather than as a decision-maker. This corresponds with some of the participants we interviewed, who

describe AI as support for double-checking, exploring extra alternatives, and filling knowledge gaps. This, while still retaining ownership of the produced code and the final decision. There is a strong sentiment among our participants that they retain ownership, which suggests they set an active boundary. This is reinforced by Figure 6.4, where a large majority disagree that the responsibility has shifted from them to AI.

Participants have shown that AI is consulted for input on decisions, but not delegated the authority to be the final decision-maker. In the context of Lee and See's Trust in automation framework [27], they would classify this as functional specificity: trust is directed at AI for specific input examples, such as alternative solutions or information, rather than at the system as a whole being a decision-maker. The participants' critical stance against pure copy-paste without validation is a limitation that can be characterized as a deliberate calibration of trust to specific functions. This can be contrasted with some participants and other work showing overtrust in AI-assisted contexts [15][14]. This can be interpreted as a cautious stance to AI, where over-reliance is less prominent than others have observed.

Organizational AI decision-making has seen an increased amount of implementation, but with a stance that humans remain in control [44]. Our results extended this observation by examining the uncertainty around when and how AI is used. In the aspect of not knowing if or when AI has been involved in a decision. Participants report that they do not know when colleagues use AI or how it has affected team decisions. This suggests that, while individuals in this study have decision-making limitations due to AI, team-level transparency of influence is lower. From an STS stance, this can itself lead to misalignment within the team, as communication in agile and DevOps teams relies on parallel work and a shared understanding of decision-making [11][12].

A related, underexamined dynamic appears from participants' uncertainty about colleagues' use of AI. P1's account of being unable to tell whether teammates' suggestions originated from their own reasoning, a Google search, or an AI tool points to a problem of provenance invisibility in team decisions. When the origin of a proposed solution cannot be traced, the meaning of the team decision becomes ambiguous: a discussion in which one participant implicitly speaks on behalf of an LLM is structurally different from one in which all participants reason from experience, but the difference is not visible to the team. This complicates accountability and the iterative decision-making that agile and DevOps practices depend on. The literature on AI in teams has not yet addressed this issue in depth, and our findings suggest it warrants further attention.

Results indicate that LLM-based AI tools have influenced decision-making and begun reshaping roles, but are currently in transition, with unclear effects. Day-to-day practices have shifted, but accountability for developers has remained. Our findings are consistent with prior literature on individual role shifts [2][30] and decision-making[44], with extensions regarding how decisions are treated and the asymmetry between official roles and practical implementation.

From RQ1, we find that LLM-based AI tools have started reshaping how developers work, but without altering their formal roles, shifting day-to-day practices towards reviewing and validation rather than creating their own code. We find that AI is integrated into decision-making, but across a range from individual lines of code to architectural decisions, serving as additional input rather than as a decision-maker. We find that accountability and responsibility remain with the developer and are not delegated to the AI.

7.1.2 RQ2: Collaborative Practices and Team Dynamics

Our findings suggest that AI tools have changed the collaborative practices and code quality outcomes in software engineering teams. One consistent point was that AI has begun to displace human-to-human interactions in various scenarios. One example was the first point of contact when encountering problems with their tasks. Participants reported consulting AI before or instead of asking a colleague for help, and reported that AI had replaced traditional sources of information, such as forums, search engines, and documentation. It is possible to frame this as a productivity finding: developers now have faster access to information and answers, with fewer interruptions for senior developers. However, if read through the socio-technical lens of Trist et al. [37], our findings may be interpreted with different significance. Knowledge in agile and DevOps teams flows not only through official structures but also through informal social moments where developers interrupt one another to ask questions that may lead to broader knowledge and context being shared as a side effect. Junior developers in particular benefit from these moments by being exposed to the reasoning of more experienced colleagues [12][33]. When these moments are replaced by AI, the technical subsystem absorbs an important mechanic that was previously incorporated across the social subsystem. The existing literature on agile collaboration aligns with this knowledge transfer interpretation, where formal peer interaction designed to get answers is described as a function for shared awareness, understanding, and the alignment of standards within a team [26][12]. The distinction between explicit and implicit knowledge is useful when analyzing the effects of AI. AI tools handle explicit knowledge such as documented APIs, industry-standardized patterns, and well-defined algorithms. What is lost when junior developers consult AI rather than senior colleagues is largely implicit: the unstated reasoning behind architectural decisions, awareness of which parts of the code are fragile and should be met with caution, and the historical context for why certain patterns are avoided. This is precisely the knowledge that pair programming and peer review have historically transmitted [12][33]. AI's substitution for these informal interactions, therefore, does not preserve the full content of what is being replaced; it preserves the explicit layer while weakening the implicit. Concerns about this shift were, for the most part, not expressed by the participants. It was instead framed as beneficial by several, particularly due to the time-saving aspects for their colleagues. Three participants noted that the interaction between junior and senior developers has been reducing, where one participant suggested that interpersonal dynamics, particularly how approachable the senior developers appeared, may be a mediating reason for instead turning to AI. This nuance extends the findings of

Barke et al. [8], who reported that less experienced developers rely more on AI to navigate and explore solutions, while focusing on the technologically driven advantages from an individual perspective. However, if juniors face a lower social cost in relying on AI than a senior colleague, the resulting loss of interaction through which knowledge has historically been transmitted could be affected.

When it comes to code quality and the factors participants perceive influencing the production of high- or low-quality code, the AI tools were described as amplifying the quality practices already in place rather than transforming them. Code quality was reported to depend on three factors that all preceded the introduction of AI tools: the clarity of the context provided to the models, the developer's competence in evaluating the output, and the existing quality culture within the codebase. Where these factors were met, the code quality was perceived as high, and where they were not, the AI was described as exacerbating existing weaknesses. If read through a socio-technical lens [37], the AI tool, which in this case is the technical subsystem, does not appear to produce perceived quality by itself but in cooperation with the social subsystem. This cooperation is achieved through team norms, context sharing, quality expectations, and review. Poor quality was shifted from the model to the developer, where it was described as a function of the user rather than a property of the tool. This has methodological implications: evaluations of AI's effect on code quality conducted at the artifact level through static analyzers, test pass rates, or vulnerability scans risk attributing to the tool what is in fact dependent on its surrounding socio-technical configuration, the same model can produce substantially different outcomes across teams, and tool-level evaluation cannot capture this variance.

We did not find that peer reviews were affected since the introduction of AI tools. This could analytically suggest that the amplifying quality is neutral with respect to collaborative review practices. Some participants, however, described peer review as loosely or never implemented before the implementation of AI. The reported stability can be difficult to interpret; it may reflect a genuine absence of impact, or it may reflect that AI has little to influence due to the prior lack of implementation. This nuance is something not clearly stated by the participants, and it can be hard to draw exact conclusions from it.

One point P11 raised was that the introduction of an AI-assisted peer-review tool reduced human engagement in the code-review process. This aligns with the automation-misuse described by Parasuraman et al. [28], as the presence of automated reviews appears to substitute for rather than complement human verification. It is further consistent with the literature on automation-complacency: when an automated system reliably produces output, human operators tend to disengage from the task, even when oversight is still required [28]. P11's account suggests that peer review tools may be vulnerable to this dynamic in a way that traditional static analysis tools are not, because their output reads as real review comments rather than machine-generated checks. Whether this is a feature of AI-assisted review tools generally or a contingent outcome of how they were introduced in this specific team cannot be determined from a single account; however, the pattern is theoretically predictable

rather than anomalous and thus warrants further empirical attention.

A particularly interesting finding concerning the relationship between AI usage and technical debt, although one that warrants contextual caution due to being connected to start-up environments. The participants in this subset framed technical debt as inherent to the development rather than introduced by the AI tool. P11 described how poor structure in a codebase tends to sustain further degradation, suggesting that "spaghetti code" produces more "spaghetti code." The increased development speed enabled by AI tools may accelerate this accumulation of technical debt. P7's offered a more nuanced perspective, contrasting with this by characterizing AI not only as an amplifier of tech debt accumulation but also as a facilitator of faster remediation. This viewpoint suggests that AI did not increase acceptance of technical debt but rather reshaped perceptions of its associated costs and maintenance. In both accounts, AI is positioned as a multiplier of existing conditions rather than a cause.

The interpretation of the amplifier has implications for how future evaluations of AI's effects on code quality are conducted. Measuring at the artifact level, whether through static analyzers or test pass rates, risks attributing to the tool what is really dependent on the developer's configuration and the tool's cooperation. These findings suggest that the same model can produce different outcomes depending on the surrounding practices and circumstances that tool-level evaluation alone cannot capture. This is consistent with the joint optimization principle in socio-technical systems [37], where optimizing both the social and technical subsystems yields a more effective outcome than either could achieve alone.

From RQ2, we find that AI has influenced when and with whom collaboration occurs rather than altering the collaboration itself. This interaction is most visible when human-to-human collaboration is replaced with AI, with respect to quick information gathering and turning to AI as a first place to ask. We find that this change particularly impacts the informal knowledge shared between juniors and seniors. We find that AI acts as an amplifier for teams' existing quality culture rather than the main source of code quality; this is dependent on the developers' competence, preexisting standards already integrated, and the provided context.

7.1.3 RQ3: Trust and Reliability

Our findings indicate that participants actively calibrate their level of trust. This trust is regulated through verification, AI experience, and ongoing recalibration. Participants' trust in AI was never absolute; it was neither pure trust nor pure distrust. Rather, trust was established through the user's own knowledge and ability to verify the output. This makes trust less of an attitude and more a product of interaction among the developer, the AI, and verification practices.

This pattern is consistent with Lee and See's framework of trust in automation [27]. The cautious mindset depicted by our participants, as seen in subsection 6.4.1.1, resembles a template still not fully formed or mapped out. Lee and See's framework

would describe this as functional specificity, as trust is granted to specific parts rather than the whole system. P8 describes this pattern when saying:

> *"We are exploring right now because we do not want to use AI in situations where it does not give us anything, so we are exploring where we can use it and where it is appropriate to use it."* (P8)[1]

The skepticism was described not as a generalized stance but a localized one, granted or withheld depending on the task. Exploratory coding and knowledge lookup were domains where trust was reported to be higher, while test generation and architectural reasoning were domains where it was lower.

The subtheme discussed in 6.4.1.2, related to a false sense of security, connects to concerns referenced in prior literature. Perry et al. [15] demonstrate that individuals are producing less secure code but with greater confidence in AI tools. While Pearce et al. [14] discussed that AI tools might suggest top-ranked code solutions that are vulnerable. Our participants reported a related concern where P11 described how generated tests can appear thorough when, in reality, there are just a lot of them, and P4 framed the underlying risk as a function of the user's own knowledge.

Reviewing and testing emerged as the primary means of building trust, as seen in subsection 6.4.1.3. Across the sample, participants described verification as effectively integrating AI output into their own work. P10 framed the reviewed AI code as practically equivalent to code they had written themselves, and P4 went further in collapsing the distinction of verified code simply as: "the code is code"[2], which suggests a more fundamental reframing than calibration alone. Lee and See's framework [27] positions trust as a property of the human-automation relationship, in which the human assesses whether the system's capabilities warrant reliance. The participants describe something different than this: trust is not granted to the AI but constructed through the developer's own verification process. Once code has been tested and reviewed, P4 and P10 treat it indistinguishably from their own code. The question of trusting AI ceases to be meaningful and is replaced by the question of whether one's own verification was adequate enough. This relocates trust from the human-machine boundaries to the human's relationship with their own practice. We suggest this as an extension of Lee and See's framework when applied to skilled professional contexts, where humans retain the capacity to verify automation output, in contrast to automation contexts where humans must rely on the system because verification is impractical, such as autopilots and medical alarms.

P8 mentions that "The trust or the limitations become more clear the more you use AI," which underscores that usage amount affects trust as a non-static increase rather than a process differentiation. This is consistent with Lee and See's dimension of resolution, in which they argue that users distinguish among varying levels of

[1] "Vi utforskar just nu, för vi vill inte använda AI i situationer det inte ger oss nåt så vi utforskar var vi kan använda det och var det är lämpligt att använda det."

[2] "så koden är kod"

system capacity [27]. There were some indications that less-experienced users trust AI more than experienced users do. This is reflected by Ziegler et al. [6] and Pearce et al. [14] on the increased risks of less experienced developers and extends this by suggesting resolution depends on general technical experience, and in addition to this, accumulated experience with AI.

Our findings suggest that caution should not be interpreted as distrust but rather as a form of needed verification for trust. The adoption of LLM-based AI tools requires experience and practice to understand what can be trusted and what should not. There is contrast to this in studies on overtrust and miscalibration, and it may reflect agile and DevOps teams' requirement for verification. If the verification process is the basis of trust, then the output of AI matters less.

From RQ3, we find that developers' trust in LLM-based AI tools is not a fixed attitude but rather an ongoing process of validation, with testing and reviewing being the main methods of building trust. We find that after verification, developers treat the code produced as their own, shifting trust from the AI to their ability to verify and thus their responsibility. Furthermore, we find that experience calibrates trust in contrast to increasing trust, where experienced users have a more selective trust in AI, with less experienced users trusting AI over a broader range.

There is a caveat we need to address for this interpretation. The calibrated trust reading depends on participants' own verification practices and their stated caution towards AI. This pattern can therefore be observed through genuine trust calibration, founded on experience with AI, or through participants reflecting professional norms and views. These two are not mutually exclusive, as a calibrated stance can be personally true and still reflect professional norms. The main issue would be in data collection, distinguishing the two views. Observational studies of how participants perform under deadline pressure would be needed to distinguish between stated and enacted calibration. This study can claim reasoning from participants regarding AI and trust in calibrated terms, but whether their behavior matches the descriptions they made is a question for future empirical studies.

7.2 Future Work

This study observed how software engineers work with LLM-based AI tools within Agile and DevOps teams. For our next steps, two primary directions to pursue have been identified.

The first direction is to conduct a more in-depth, larger-scale study to get a better understanding and broader picture of the current state of AI in the workplace; more time for interviews and surveys would be highly important. A longitudinal study following the same groups would provide a better understanding of role shifts, social interactions, and actual AI usage. This would enable formally recognized changes that could not be established from this study. There is also merit, depending on the study, in interviewing outside Sweden to get a broader perspective from multiple

countries.

A Second direction to follow is from the anecdotal findings in section 6.5.1, where participants noted learning and short-term productivity at the expense of long-term competence development. Although this is not central to our study, it was a theme that was often brought up and therefore warrants further investigation. Observations from participants, such as P7, indicated that reliance on AI makes debugging harder, and other participants noted that juniors risk solving problems without understanding, which can have long-term effects that this study could not encompass. Future studies could examine this through empirical and longitudinal research, following juniors' development as they advance in seniority.

8

Conclusion

This thesis set out to investigate how LLM-based AI tools have influenced Agile and DevOps teams, as perceived by developers. Where the focus is on decision-making, role distribution, collaborative practices, team dynamics, and developers' assessment of trust in AI-generated code. Through a semi-structured interview with 11 participants, a complementary survey using Braun and Clark's thematic analysis, and interpretation guided by socio-technical system theory and Lee and See's framework, this work contributes to research on team dynamics and social interaction, where previous studies have largely focused on the individual.

Our findings indicate that LLM-based AI tools within the software development team are best understood as an ongoing transition rather than a completed transformation. Turning to how AI tools have affected decision-making and role distribution within teams (RQ1), participants report that formal roles have not changed, while some indicate changes in responsibilities. The day-to-day practices have changed in meaningful ways. Observations from participants indicate that reviewing and validating code has become a larger part of their work than writing code from scratch. This is a trend others expect to see in the future as well, with more and more time spent on reviewing and validating. AI was described as increasingly influential in decision-making in a wide range of areas. Participants made a clear statement about retaining ownership of the code, even when AI was involved in the creation. The LLM-based tools were welcome as consultants, but accountability for the product remained with the developer.

Regarding collaborative practices and team dynamics (RQ2), our results suggest that collaborative practices have changed since the introduction of AI. Participants mention that AI has replaced traditional information sources, including documentation, search engines, online forums, and, increasingly, the first place they turn when encountering problems, rather than a colleague. This interaction is one in which information and knowledge were traditionally shared between individuals. While there was mention of AI relieving seniors of questions from juniors, others mentioned that this harmed the relationship between seniors and juniors. Peer review practices were largely reported as unchanged, but existing weaknesses persisted even after AI was introduced. AI was mentioned as a replacement for peer review rather than complementing developers. Since the introduction of AI, code quality has not been seen as a producer of good or bad code, but rather as an amplifier, where

AI scales with existing quality and developers' competence. The context, clarity, and standards provided by the developer were the main factors in determining code quality.

In response to trust and reliability (RQ3), our results show that trust in AI was neither absolute nor static but a construct shaped by validation, experience, and ongoing recalibration. Participants expressed caution regarding AI and had selective trust based on experience, with specific areas considered more trustworthy than others. The foundation for trust in AI was established through testing and review, with some noting that once this was done, the distinction between human and AI code became irrelevant, as validation had occurred. Participants also noted that experience did not always increase or decrease trust; rather, it varied across different areas, with juniors tending to trust AI more than seniors. The cautious stance described was better understood as a mature form of trust.

Across our three RQs, our analysis constructed a consistent pattern. LLM-based AI tools in the context of this study did not appear to autonomously transform software developers; they amplified and accelerated their workflows and reflected the team's current competence. The result shown, whether positive or negative, depended more on the configuration of the surrounding socio-technical system and less on AI's actual capabilities. Current quality cultures were reflected, and strong quality cultures were reinforced while weak ones were amplified. Experienced developers took a more cautious stance toward AI and were less inclined to place blind trust, whereas less experienced developers were more likely to do so. Teams with an existing culture around peer review were reported to be less affected by AI. The amplification dynamics that AI introduces can be seen through a socio-technical lens, reinforcing that technical and social aspects must be integrated for optimization, and that the effects of AI cannot be evaluated in isolation but rather in relation to the human practices and team norms it interacts with.

The contributions this thesis provides are twofold. From a research perspective, previous work had mostly focused on the individual, whereas this study evaluates team dynamics. Previous literature has examined individual developers' or organizations' interactions with AI; this study presents a new perspective on socio-technical dynamics at the team level. From a practical perspective, this study offers insight for practitioners, teams, and managers to build a foundation for how AI affects workflows and social interactions. Where attention to certain areas could be warranted, such as safeguarding informal collaborative practices where information and knowledge have traditionally been shared, they may be in danger.

Future work should focus on longitudinal studies that capture the evolution of the aspects covered in this study. Furthermore, future studies should encompass a larger-scale operation that focuses on multiple organizations and their teams. The effects on short-term efficiency versus long-term competence development are an important focus of upcoming studies in this field. As AI integration increases in software development practices, empirical studies should focus on capturing team collaboration to ensure that AI adoption is as successful as possible and not something humans

will endure in the future.

Bibliography

[1] B. Martinović and R. Rozić, "Perceived impact of AI-based tooling on software development code quality," *SN Computer Science*, vol. 6, no. 63, Jan. 2025.

[2] E. De La Cruz, H. Le, K. Meduri, G. S. Nadella, and H. Gonaygunta, "Redefining the programmer: Human-AI collaboration, LLMs, and security in modern software engineering," *Computers, Materials & Continua*, vol. 85, no. 2, pp. 3569–3582, Sep. 2025.

[3] L. Yu and Y. Li, "Artificial intelligence decision-making transparency and employees' trust: The parallel multiple mediating effect of effectiveness and discomfort," *Behavioral Sciences*, vol. 12, no. 5, p. 127, Apr. 2022.

[4] N. Castelo, M. W. Bos, and D. R. Lehmann, "Task-dependent algorithm aversion," *Journal of Marketing Research*, vol. 56, pp. 809–825, 2019.

[5] R. Choudhuri, D. Liu, I. Steinmacher, M. Gerosa, and A. Sarma, "How far are we? the triumphs and trials of generative AI in learning software engineering," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, ser. ICSE '24, 2024, pp. 1–13.

[6] A. Ziegler, E. Kalliamvakou, X. A. Li, and A. Rice, "Measuring GitHub Copilot's impact on productivity," *Communications of the ACM*, vol. 67, no. 3, pp. 54–63, feb 2024.

[7] Z. K. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz, "The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers," *Management Science*, 2026.

[8] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, 2023.

[9] R. Khojah, M. Mohamad, P. Leitner, and F. Gomes de Oliveira Neto, "Beyond code generation: An observational study of ChatGPT usage in software engineering practice," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1819–1840, 2024.

[10] R. Wang, R. Cheng, D. Ford, and T. Zimmermann, "Investigating and designing for trust in AI-powered code generation tools," in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1475–1487.

[11] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," https://agilemanifesto.org/, 2001, accessed: 2026-04-08.

[12] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*. IEEE Press, 2013, pp. 712–721.

[13] Y. Liu, T. Le-Cong, R. Widyasari, C. Tantithamthavorn, L. Li, X.-B. D. Le, and D. Lo, "Refining ChatGPT-generated code: Characterizing and mitigating code quality issues," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, p. 116, 2024.

[14] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," *Communications of the ACM*, vol. 68, no. 2, pp. 96–105, 2025.

[15] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. ACM, 2023, pp. 2785–2799.

[16] X. Yan, Y. Xiao, and Y. Jin, "Generative large language models explained [ai-explained]," *IEEE Computational Intelligence Magazine*, vol. 19, no. 4, pp. 45–46, November 2024.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[18] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[19] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, and T. Gui, "The rise and potential of large language model based agents: A survey," *Science China Information Sciences*,

vol. 68, no. 2, p. 121101, 2025.

[20] Anthropic, "Claude code," Website and technical summaries describing Claude Code as an agentic AI coding tool, 2025, https://claude.com/blog/introduction-to-agentic-coding.

[21] Cursor, "Cursor agent," Online articles and tool comparisons discussing Cursor's agent mode, 2026, see discussion of Cursor's agentic capabilities in: https://hiveoscity.com/glossary/agents.

[22] C. Labs, "Devin ai," Tool description and documentation, 2026, https://en.wikipedia.org/wiki/Devin_AI.

[23] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," *Journal of Software: Evolution and Process*, vol. 29, no. 6, p. e1885, 2017.

[24] D. Sovtić, A. Trpkov, M. Radenković, M. Jevtić, A. Labus, and L. Marković, "Methodological approach to applying devops for the development of a decentralized application," in *2025 6th International Workshop on Engineering Technologies and Computer Science (EnT)*, Sankt Peterburg, Russian Federation, 2025, pp. 1–7.

[25] I. Nonaka and H. Takeuchi, "The new new product development game," *Harvard Business Review*, vol. 64, no. 1, pp. 137–146, 1986.

[26] V. Stray, D. I. K. Sjøberg, and T. Dybå, "The daily stand-up meeting: A grounded theory study," *Journal of Systems and Software*, vol. 114, pp. 101–124, 2016.

[27] J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," *Human Factors*, vol. 46, no. 1, pp. 50–80, 2004.

[28] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Human Factors*, vol. 39, no. 2, pp. 230–253, 1997.

[29] K. A. Hoff and M. Bashir, "Trust in automation: Integrating empirical evidence on factors that influence trust," *Human Factors*, vol. 57, no. 3, pp. 407–434, 2015.

[30] O. Bruhin, "Beyond code: The impact of generative AI on work systems in software engineering," in *Proceedings of the International Conference on Information Systems (ICIS)*, Bangkok, Thailand, Dec. 2024.

[31] OpenAI, "ChatGPT," https://chat.openai.com, 2023, large language model.

[32] GitHub, "GitHub Copilot: Your AI pair programmer," https://github.com/features/copilot, 2021, accessed: 2026-04-15.

[33] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair programming," *IEEE Software*, vol. 17, no. 4, pp. 19–25, 2000.

[34] S. Imai, "Is GitHub Copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '22)*, 2022, pp. 319–321.

[35] E. Malvaceda-Espinoza and G. Flores-Pereyra, "Methodological aspects in the construction of the protocol in semi-structured interviews," *Education Commons, Psychology Commons, Quantitative, Qualitative, Comparative, and Historical Methodologies Commons, Social Statistics Commons*, 12 2025, 672 downloads since January 05, 2026.

[36] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[37] E. L. Trist and K. W. Bamforth, "Some social and psychological consequences of the longwall method of coal-getting," *Human relations*, vol. 4, no. 1, pp. 3–38, 1951.

[38] R. P. Bostrom and J. S. Heinen, "Mis problems and failures: a socio-technical perspective," *MIS quarterly*, pp. 17–32, 1977.

[39] P. Ralph, N. bin Ali, S. Baltes, D. Bianculli, J. Diaz, Y. Dittrich, N. Ernst, M. Felderer, R. Feldt, A. Filieri, B. B. N. de França, C. A. Furia, G. Gay, N. Gold, D. Graziotin, P. He, R. Hoda, N. Juristo, B. Kitchenham, V. Lenarduzzi, J. Martínez, J. Melegati, D. Mendez, T. Menzies, J. Molleri, D. Pfahl, R. Robbes, D. Russo, N. Saarimäki, F. Sarro, D. Taibi, J. Siegmund, D. Spinellis, M.-A. Storey *et al.*, "Empirical standards for software engineering research," arXiv preprint arXiv:2010.03525 [cs.SE], 2021.

[40] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[41] R. K. Yin, "Discovering the future of the case study method in evaluation research," *Evaluation Practice*, vol. 15, no. 3, pp. 283–290, 1994.

[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, 2024.

[43] Miro, "Miro: The visual workspace for innovation," 2025, accessed: 2026-05-03. [Online]. Available: https://miro.com

[44] X. L. J. Song, Yanshuo; Qiu, "The impact of artificial intelligence adoption on organizational decision-making: An empirical study based on the technology acceptance model in business management," *Systems*, vol. 13, no. 8, 2025.

[Online]. Available: https://libkey.io/10.3390/systems13080683

A

Appendix - Procedure

Figure A.1: Transcription with highlighted parts

Figure A.2: First grouping of codes

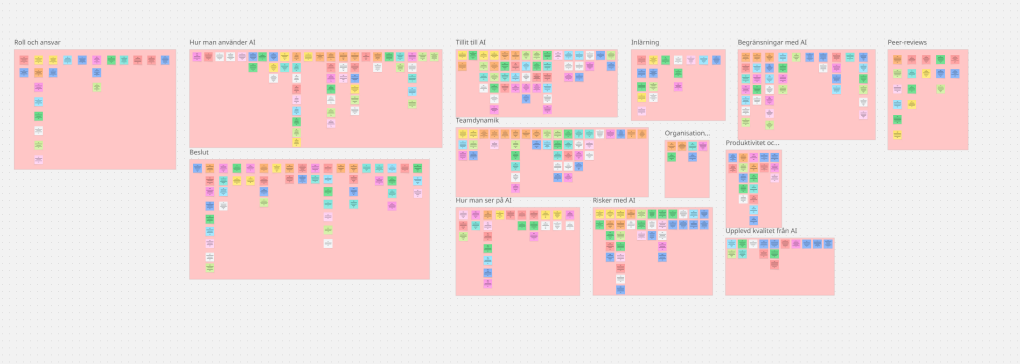


Figure A.3: First grouping of codes with participants

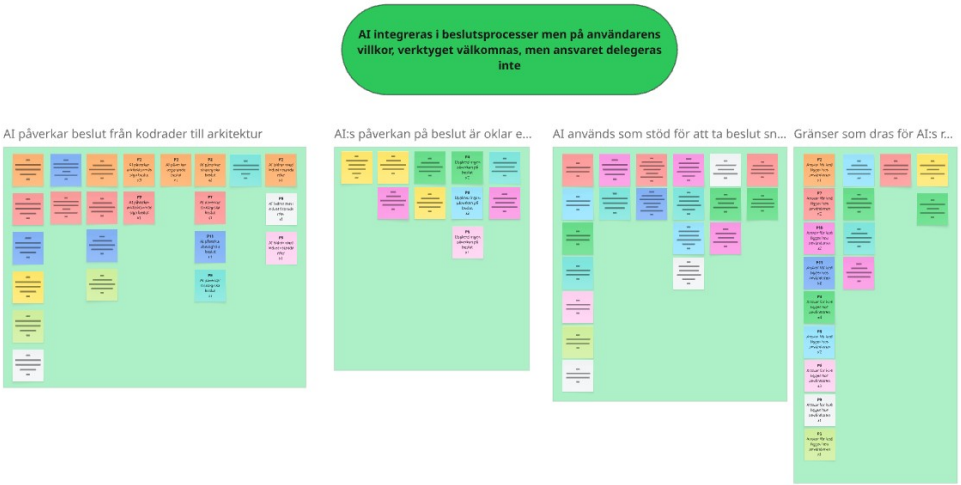


Figure A.4: Final themes and sub-themes with codes

B

Appendix - Interview Guide

This interview guide was originally conducted in Swedish, as all participants were native Swedish speakers, and has been translated into English for this appendix. The purpose of the guide is not to follow it directly but rather to be used to steer the interview in the right direction.

Section 1 – Background and Context

Can you briefly describe your role and the team you work in?

- What is your formal title?
- What are your daily tasks?
- What types of tasks do you typically work on?
- How large is the team?
- Do you work according to Agile or DevOps?
- Do you collaborate with other teams?

How long have you worked in this team?

- Have you worked in similar teams before?

Does your team use AI tools in its work?

- If yes: Are the tools officially approved by the organization? Which types of AI tools are used, and how? Have you worked in this team (or a similar one) before AI was introduced?
- If no: Why not? Is it due to company policy or personal preference?

Section 2 – Decision-Making and Roles

Has the introduction of AI affected how decisions are made in your team?

- If yes: What types of decisions? (technical design, architecture, implementation, bug fixes) Has it accelerated decision-making? Has AI suggested alternative solutions that would not otherwise have been discussed?
- If no: Why do you think it has not affected decision-making? Is AI primarily used as support for syntax or documentation rather than for proposing solutions?
- If an AI suggestion is implemented, would you consider it your decision or the AI's?

Do you feel that your role or responsibilities have changed since AI was introduced?

- If yes: In what way? More code review and less code writing? Greater focus on architecture and validation? Has productivity changed? Has your learning changed? Do you think less independently? Has communication between senior and junior developers changed?
- If no: Do you think it will change? Do you think AI has affected other roles?

Can you describe a concrete situation where AI influenced a decision or outcome in your team?

- If yes: What was the context? What did the AI suggest? Would the same decision have been made without AI? What was the outcome? How did it affect you?
- If no: Would you want AI to assist in decision-making? Is there a situation where you think AI could have helped?

Section 3 – Code Review and Code Quality

Have code reviews changed after the introduction of AI-generated or AI-assisted code in your team?

- If yes: How? Do reviewers know when code is AI-assisted? Are reviews stricter or more cautious? Is AI-generated code assumed to be of lower or higher quality? Are there extra validation steps?
- If no: Why have reviews not changed? Do reviewers know when code is AI-assisted?

How would you describe the team's general perception of the appropriateness of using AI-assisted code?

- Is it seen as helpful, risky, lazy, or efficient?
- Does the perception vary depending on seniority or experience?

- Is there disagreement around its use?
- Is there stigma around using AI at work?

How do you personally perceive the quality of AI-generated code compared to manually written code?

- In which situations is it better?
- In which situations is it worse?
- Does it depend on complexity?
- Does it introduce problems that are difficult to detect?
- Does it affect how you maintain the codebase?

Section 4 – Testing and Validation

Has the introduction of AI affected how testing is carried out in your team?

- If yes: Are tests generated by AI? Are more tests written? Has test coverage changed? Are the tests better?
- If no: Is AI not used for testing? Why do you think it has not affected testing?

How confident do you feel about tests that are generated or influenced by AI?

- In what way are you more or less confident?
- Do you review them more carefully?
- Do they require modification?
- Are they superficial or accurate?
- What does AI do that affects your trust?

Have you experienced problems — such as bugs, errors, or security vulnerabilities — that you associate with AI-generated code or tests?

- If yes: How was it discovered? What was the problem? How did it arise? How was it resolved?
- If no: Do you think it is only a matter of time?

Section 5 – Trust and Reliability

How does your team determine when you can trust AI-generated code?

- How did you arrive at that assessment?
- Does it depend on the complexity of the task?

Are there situations where you trust AI more or less?

- Why? Scaffolding or boilerplate? Repetitive tasks? Documentation? Safety-critical tasks? Larger architectural decisions?

Has your trust in AI-generated code changed over time?

- Has it increased or decreased? Why?
- Any defining events?
- Does trust increase with experience?
- How do you use AI now compared to before?

Section 6 – Team Dynamics and Social Effects

Has AI affected collaboration or discussions within the team?

- More technical discussions?
- Fewer discussions because AI has already suggested something?
- Is AI a "third voice" in meetings?
- Does AI reduce junior developers' dependence on seniors?

Has AI affected how team members explore unfamiliar parts of the codebase?

- Do people ask AI instead of colleagues?
- Does it increase learning speed?
- Does it lead to accurate understanding?
- Does it accelerate onboarding?
- Does it create superficial understanding?

Has AI changed how knowledge is shared or learned within the team?

- If yes: Less peer-to-peer learning? More self-directed learning? Are seniors consulted less? Has documentation work changed? Do people choose AI over colleagues?
- If no: Has AI never replaced asking a colleague? Why do you think that is?

Section 7 – Reflection

What do you see as the main benefits and risks of using AI in your team?

- Productivity
- Code quality
- Skill degradation
- Over-reliance
- Security risks
- Innovation
- Short-term vs. long-term risks

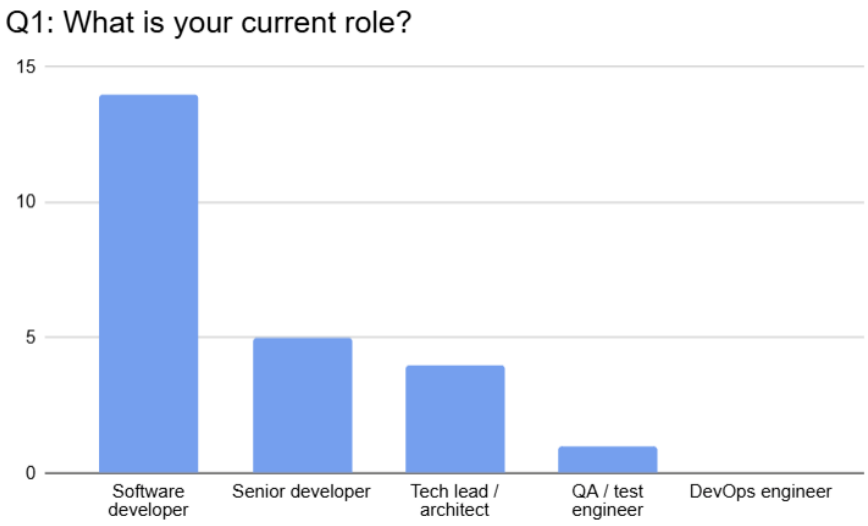
If you could change how AI is used in your team, what would you change?

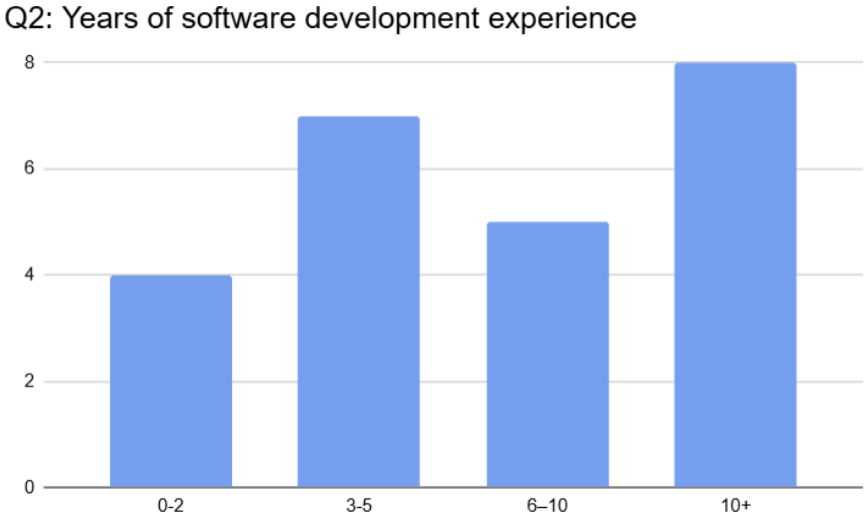
- Policies?
- Training?
- Restrictions?
- More or less usage?
- Clearer guidelines?
- Choice of tools?

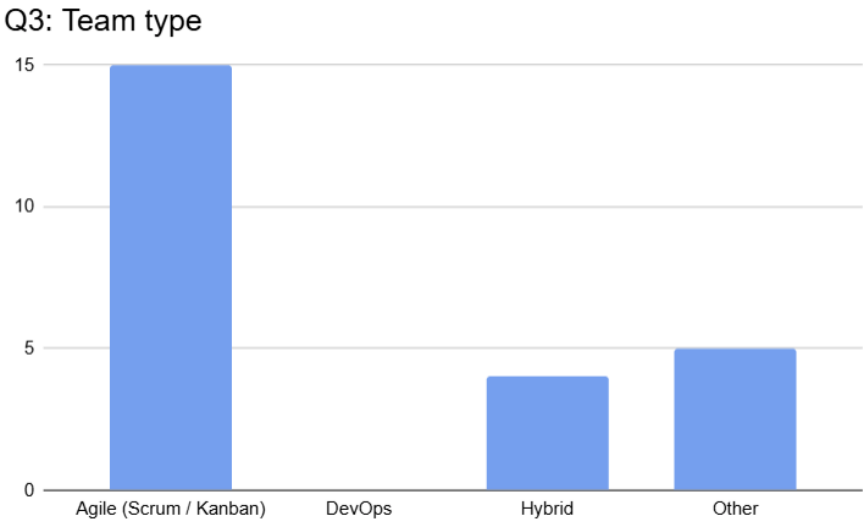
Is there anything we have not discussed that you think is important regarding working with AI in your team?

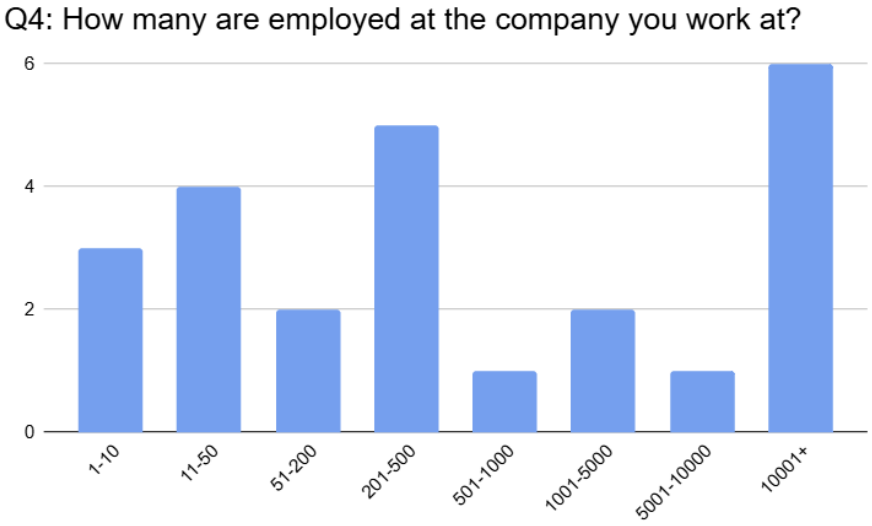
C

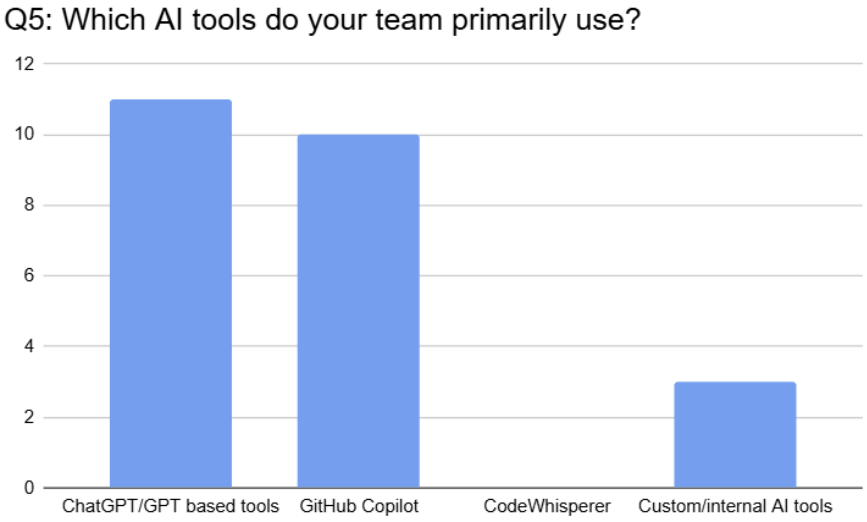
Appendix - Survey Questions and Answers

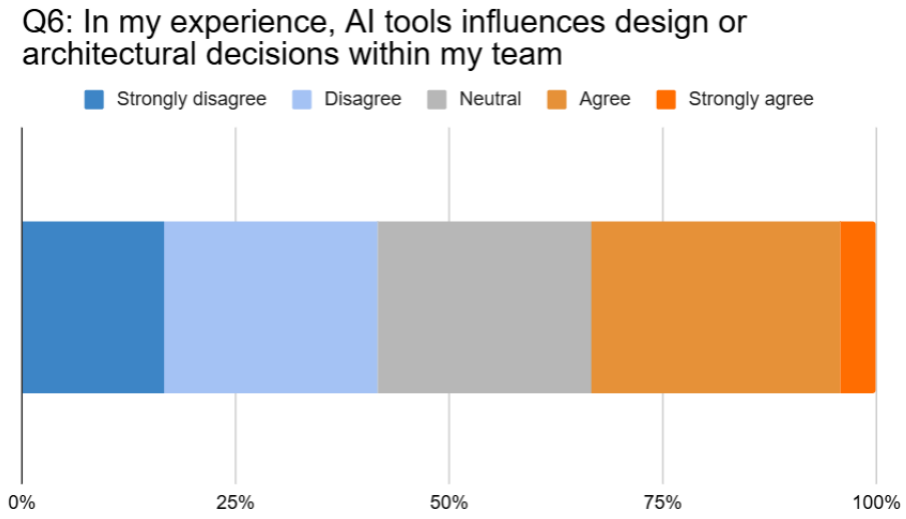


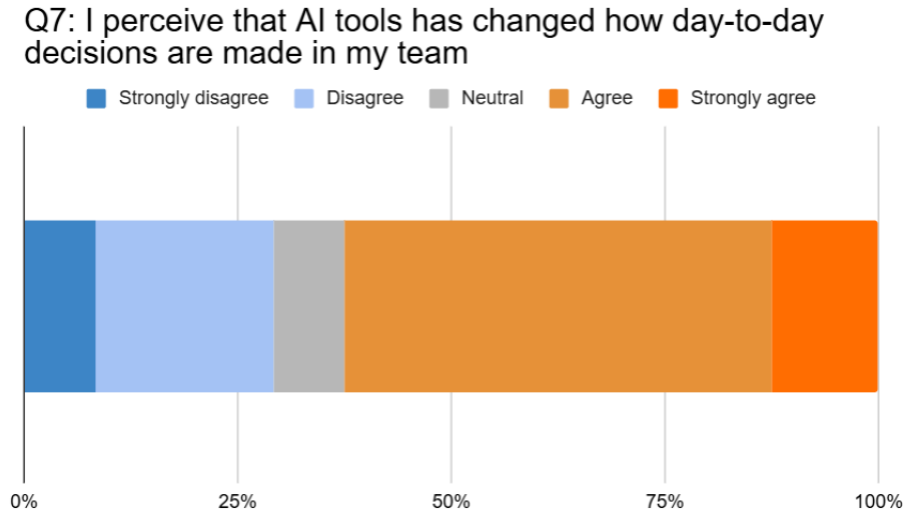


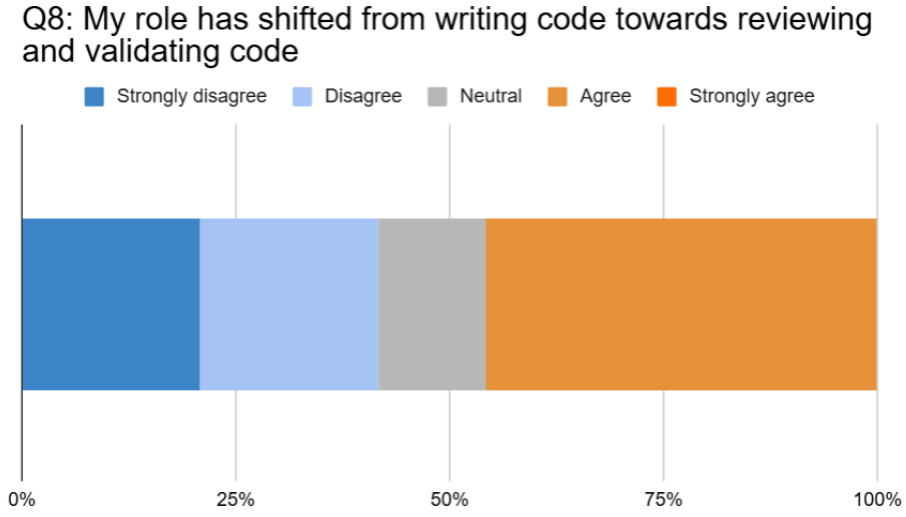

Q3: Team type
15
10
5
0
Agile (Scrum / Kanban)
DevOps
Hybrid
Other

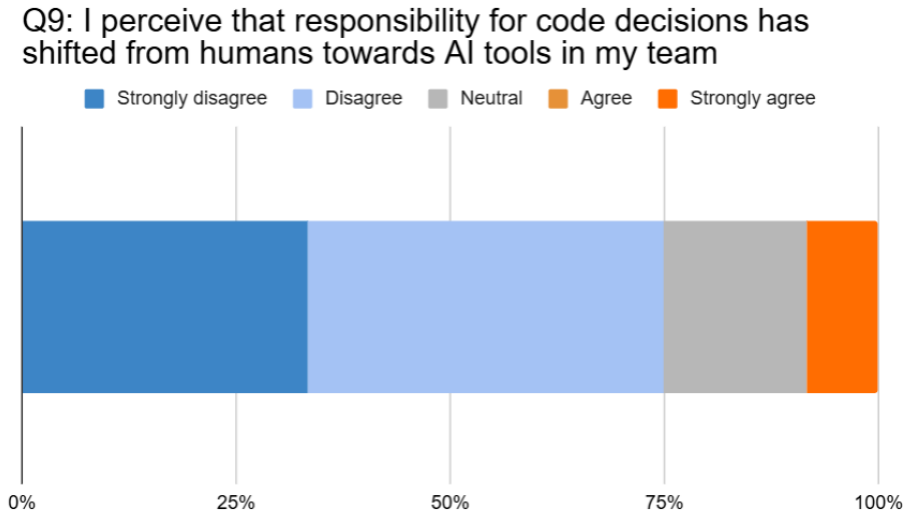

Q4: How many are employed at the company you work at?
6
4
2
0
1-10
11-50
51-200
201-500
501-1000
1001-5000
5001-10000
10001+

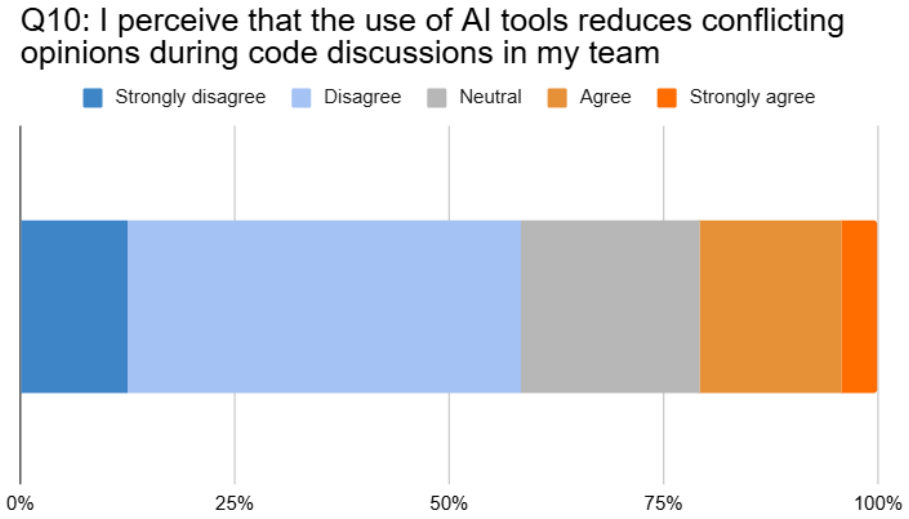

Q5: Which AI tools do your team primarily use?
12
10
8
6
4
2
0
ChatGPT/GPT based tools
GitHub Copilot
CodeWhisperer
Custom/internal AI tools

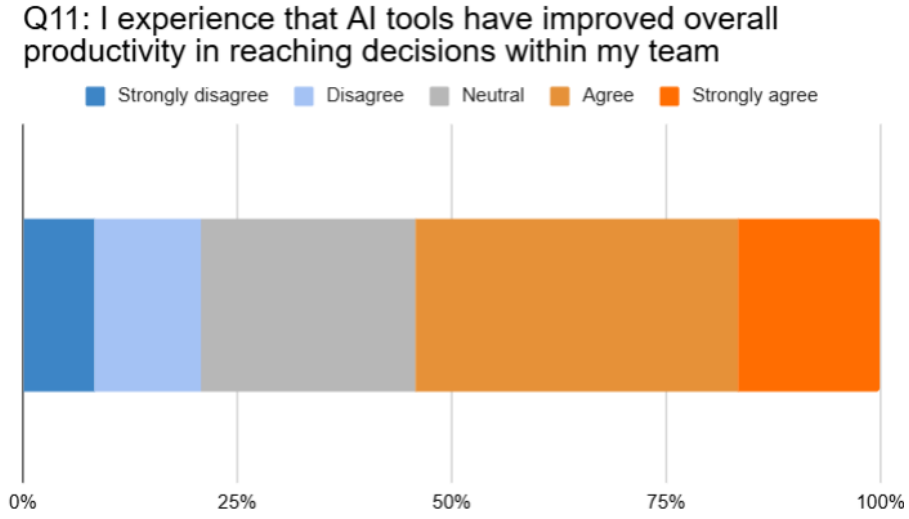

Q6: In my experience, AI tools influences design or architectural decisions within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

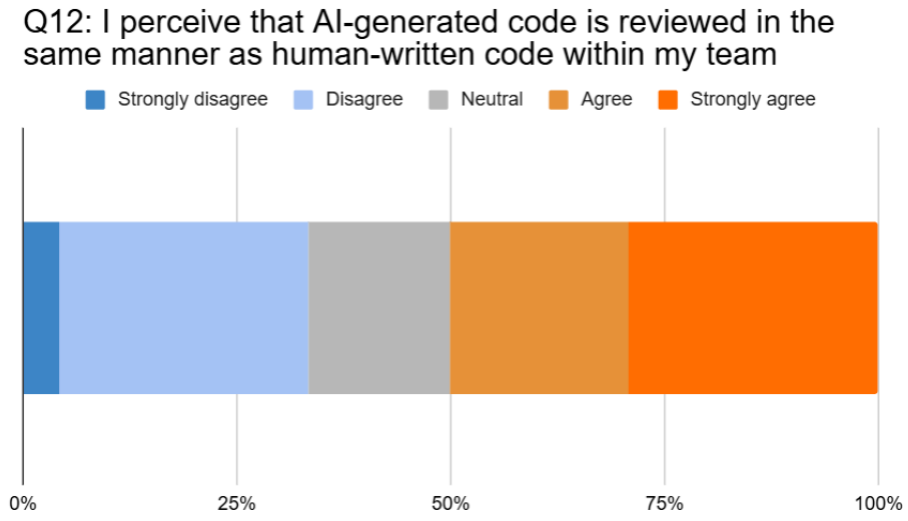

Q7: I perceive that AI tools has changed how day-to-day decisions are made in my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

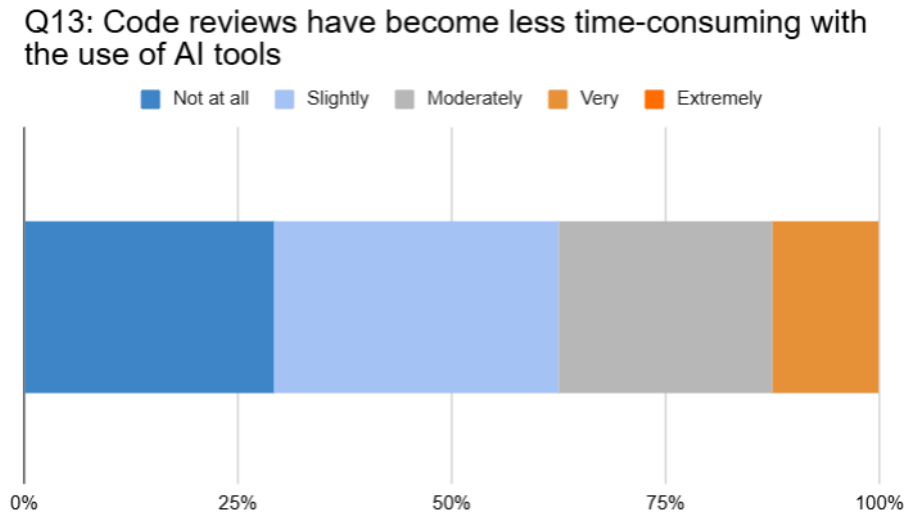

Q8: My role has shifted from writing code towards reviewing and validating code
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

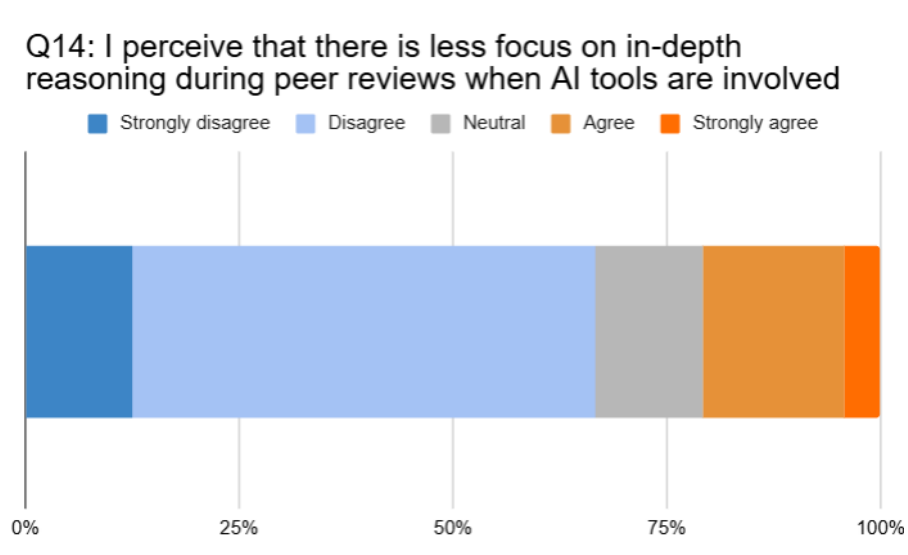

Q9: I perceive that responsibility for code decisions has shifted from humans towards AI tools in my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

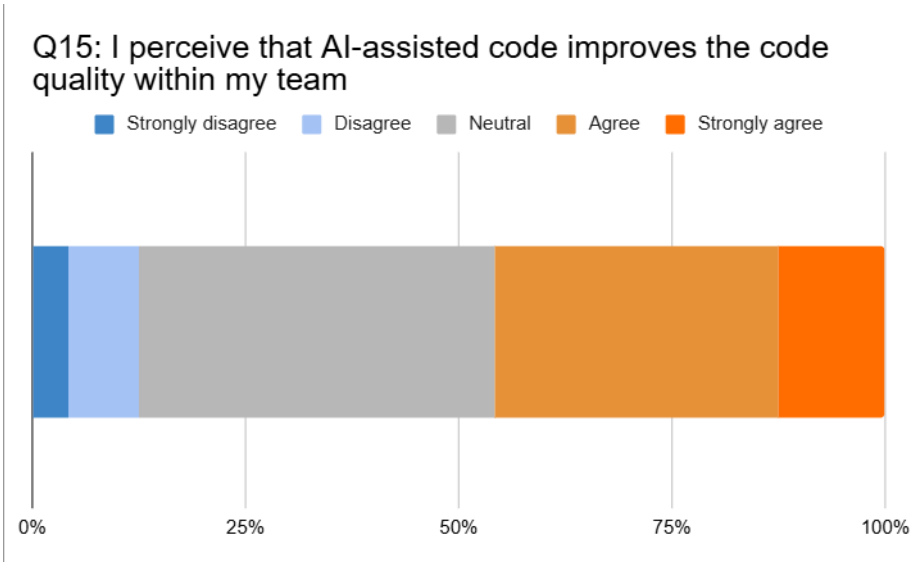

Q10: I perceive that the use of AI tools reduces conflicting opinions during code discussions in my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

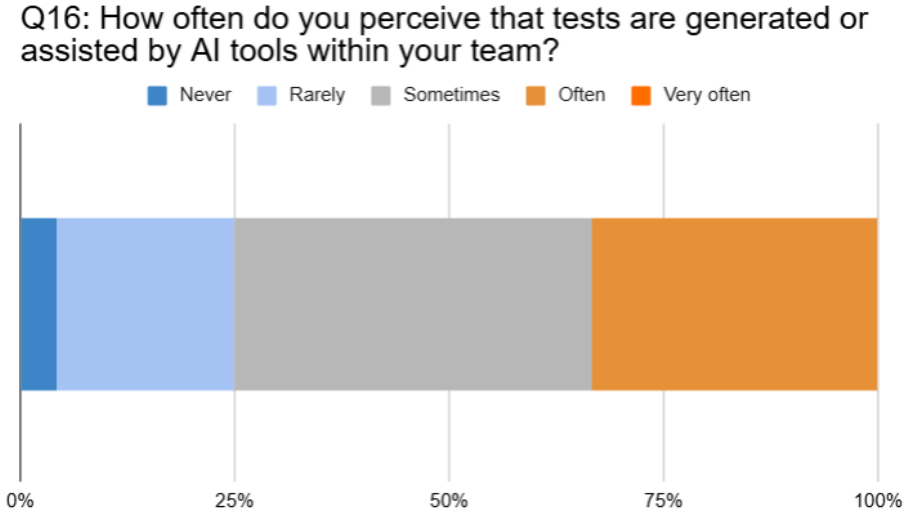

Q11: I experience that AI tools have improved overall productivity in reaching decisions within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

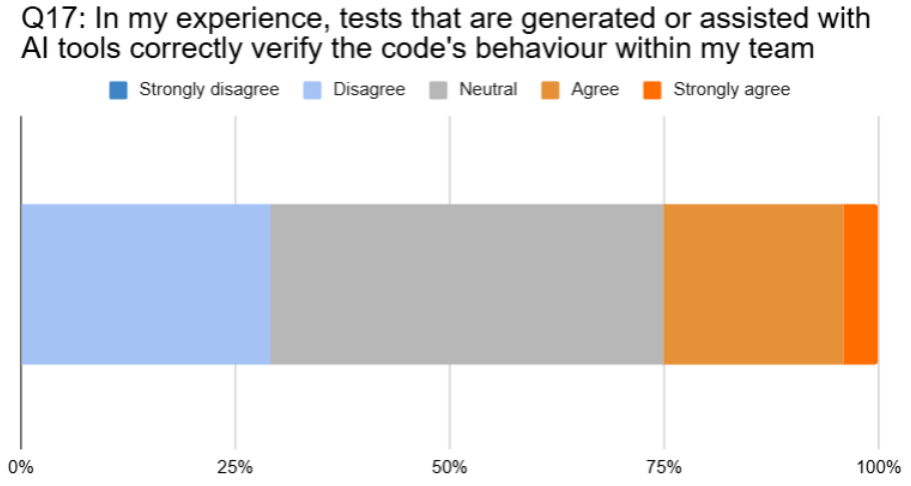

Q12: I perceive that AI-generated code is reviewed in the same manner as human-written code within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

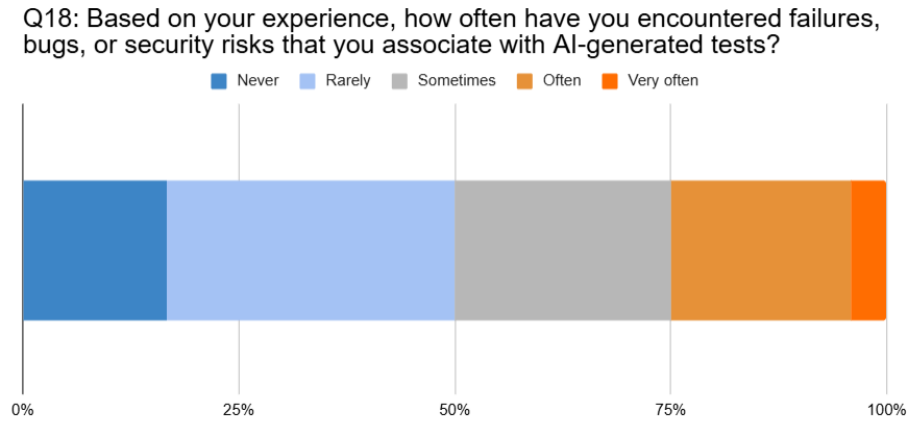

Q13: Code reviews have become less time-consuming with the use of AI tools
Not at all
Slightly
Moderately
Very
Extremely
0%
25%
50%
75%
100%

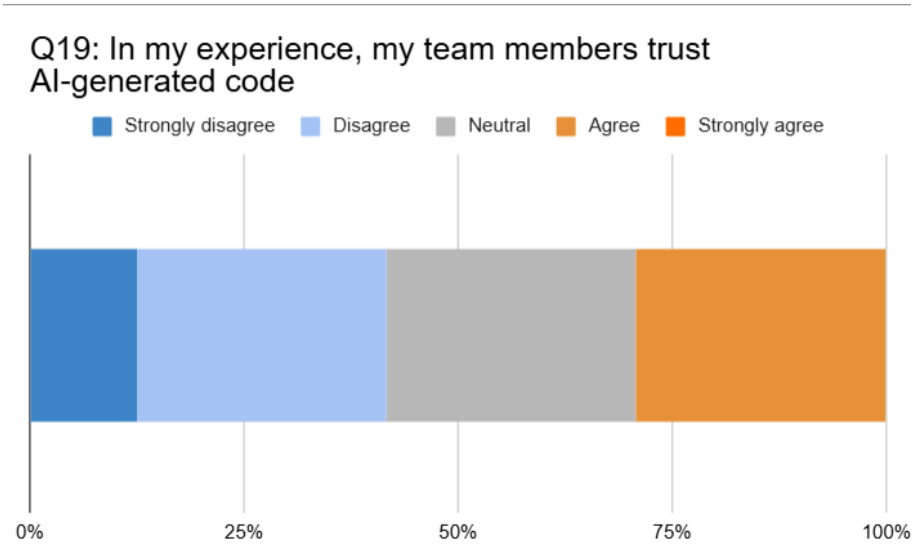

Q14: I perceive that there is less focus on in-depth reasoning during peer reviews when AI tools are involved
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

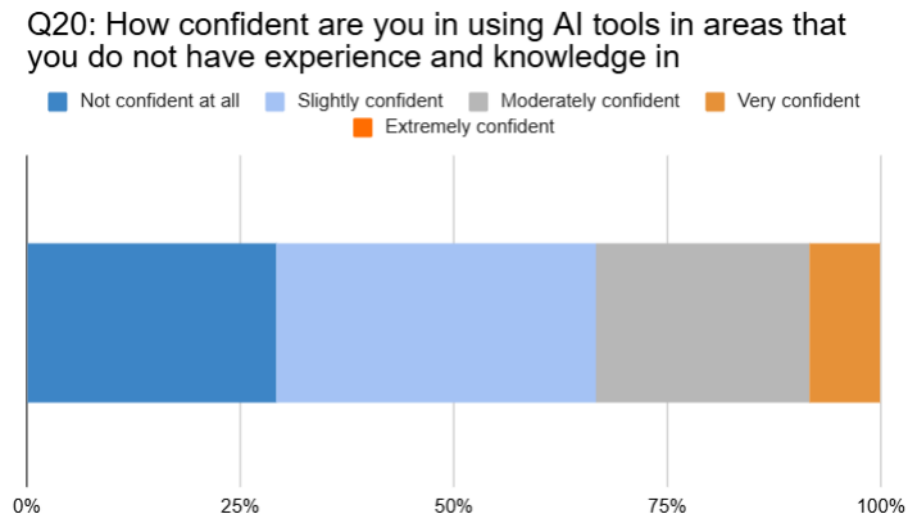

Q15: I perceive that AI-assisted code improves the code quality within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

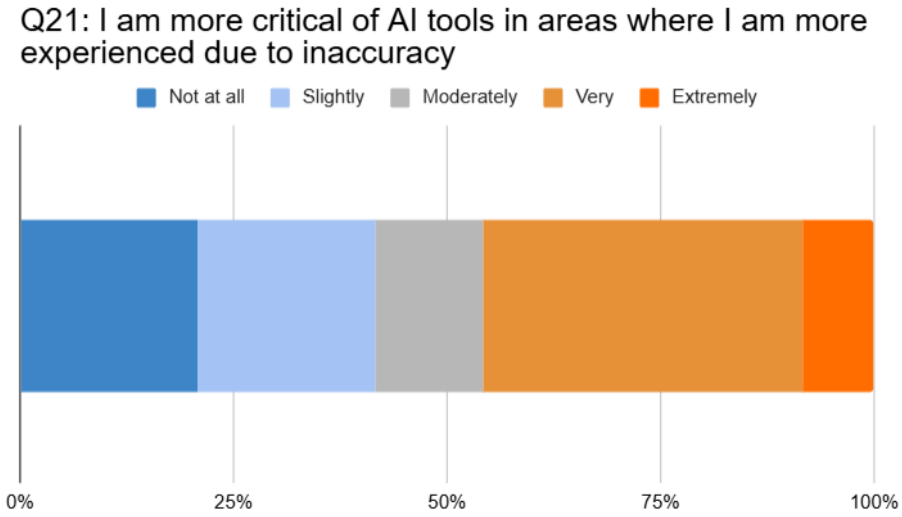

Q16: How often do you perceive that tests are generated or assisted by AI tools within your team?
Never
Rarely
Sometimes
Often
Very often
0%
25%
50%
75%
100%

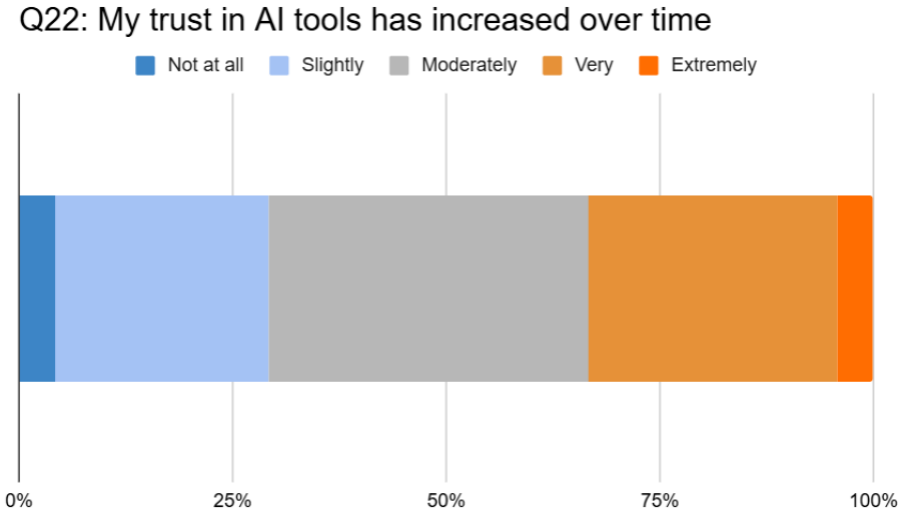

Q17: In my experience, tests that are generated or assisted with AI tools correctly verify the code's behaviour within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

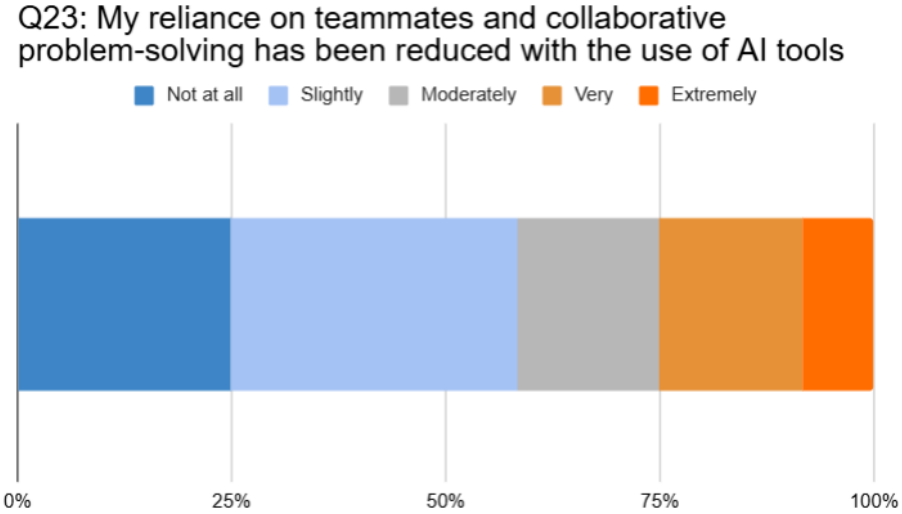

Q18: Based on your experience, how often have you encountered failures, bugs, or security risks that you associate with AI-generated tests?
Never
Rarely
Sometimes
Often
Very often
0%
25%
50%
75%
100%

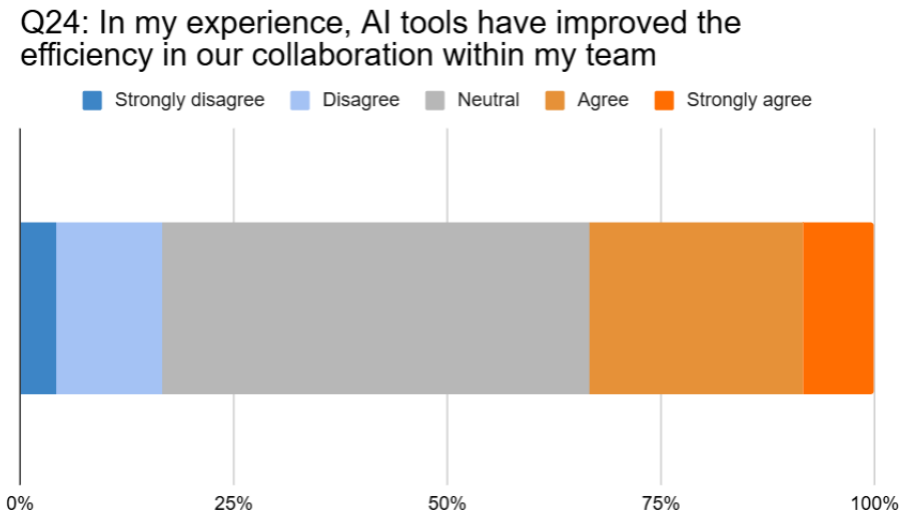

Q19: In my experience, my team members trust AI-generated code
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%

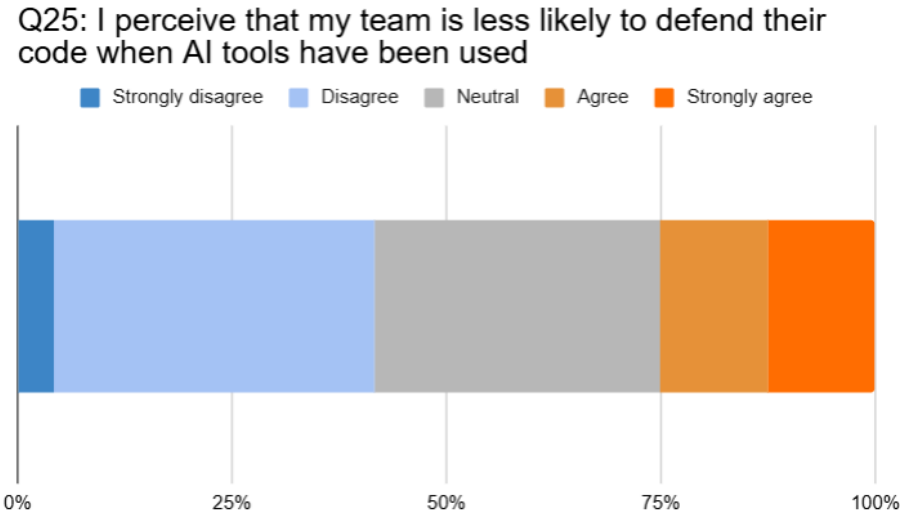

Q20: How confident are you in using AI tools in areas that you do not have experience and knowledge in
Not confident at all
Slightly confident
Moderately confident
Very confident
Extremely confident
0%
25%
50%
75%
100%

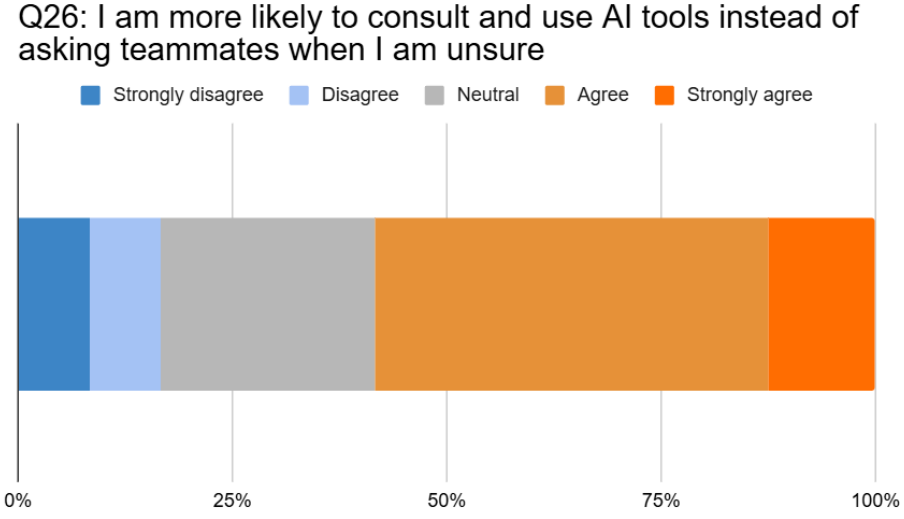

Q21: I am more critical of AI tools in areas where I am more experienced due to inaccuracy
Not at all
Slightly
Moderately
Very
Extremely
0%
25%
50%
75%
100%

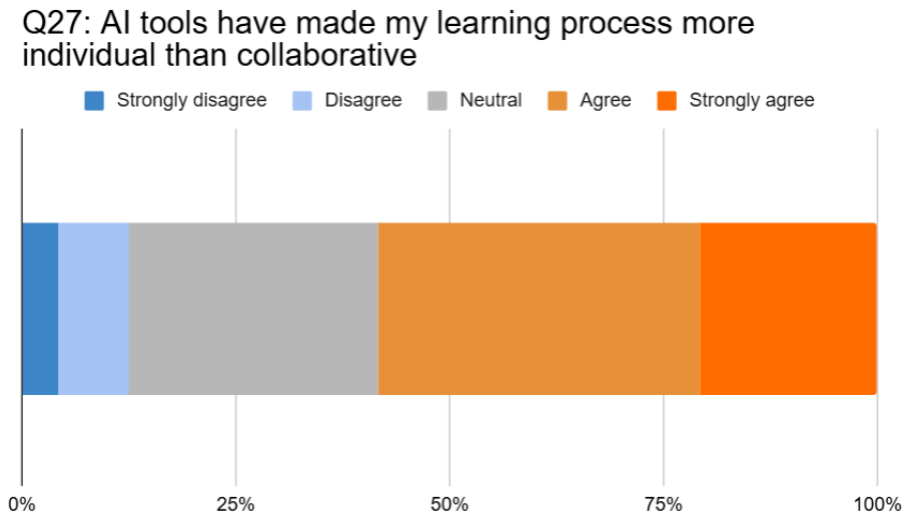

Q22: My trust in AI tools has increased over time
Not at all
Slightly
Moderately
Very
Extremely
0%
25%
50%
75%
100%


Q23: My reliance on teammates and collaborative problem-solving has been reduced with the use of AI tools
Not at all
Slightly
Moderately
Very
Extremely
0%
25%
50%
75%
100%


Q24: In my experience, AI tools have improved the efficiency in our collaboration within my team
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%


Q25: I perceive that my team is less likely to defend their code when AI tools have been used
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%


Q26: I am more likely to consult and use AI tools instead of asking teammates when I am unsure
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%


Q27: AI tools have made my learning process more individual than collaborative
Strongly disagree
Disagree
Neutral
Agree
Strongly agree
0%
25%
50%
75%
100%