CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Redefining the Developer Role

A Case Study on Human-AI Collaboration in Software Engineering at a Large Manufacturing Company

Master's Thesis in Software Engineering and Technology

Samuel Falck & Cecilia Sundell

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

Master's Thesis 2026

Redefining the Developer Role

A Case Study on Human-AI Collaboration in Software Engineering at a Large Manufacturing Company

Samuel Falck & Cecilia Sundell

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Redefining the Developer Role: A Case Study on Human-AI Collaboration in Software Engineering at a Large Manufacturing Company
Samuel Falck & Cecilia Sundell

Supervisor: Daniel Strüber, Department of Computer Science and Engineering
Examiner: Hans-Martin Heyn, Department of Computer Science and Engineering
Company Involved: Husqvarna Group, Digital Solutions
Company Contact: Andreas Rowell

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

Samuel Falck & Cecilia Sundell
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Generative Artificial Intelligence (GenAI) is fundamentally transforming software engineering. While the potential efficiency gains of these tools are widely recognized, their impact on roles and organizations in real-world contexts is only beginning to emerge. This case study investigates the integration and consequences of GenAI at a large Swedish manufacturing company through 25 interviews with participants from multiple software engineering domains. Specifically, the research examines current usage patterns, concrete use cases, and practices for human-AI collaboration. The results indicate that GenAI integration profoundly redefines developers' identities, shifting their primary roles from traditional "coders" to "code reviewers" and "software architects". Developers are moving upstream in the development cycle, assuming responsibilities traditionally held by product owners and project managers. Core competencies are shifting as the need for syntax memorization is replaced by systems thinking and communication skills. While the vast majority of participants reported individual time savings, this efficiency is largely reinvested into raising quality baselines rather than directly increasing output volume. Furthermore, developers across all domains remain reluctant to trust autonomous AI without strict human oversight. The study also reveals significant disparities in GenAI adoption, with embedded systems developers demonstrating slower adoption rates. At the organizational level, GenAI presents new opportunities, such as cognitive offloading and enhanced cross-functional communication. However, it also introduces new challenges and bottlenecks. The primary development bottleneck has shifted from *how* to build a solution to *what* to build, accompanied by a shift toward requirements engineering and an emerging "deskilling" threat that could result in a future shortage of developers with deep domain expertise. To realize the true potential of GenAI, organizations must take proactive responsibility for upskilling, unifying AI strategies, and protecting the junior developer role.

Keywords: software engineering, generative AI, human-AI collaboration, developer role, deskilling

Acknowledgements

We would like to express our sincere gratitude to everyone who has supported us through the process of writing this master's thesis. First and foremost, we would like to thank our supervisor at Husqvarna Group, Andreas Rowell, and the entire Digital Solutions department for giving us the opportunity to conduct this case study. Thank you for welcoming us, providing valuable resources and showing genuine interest in our study. Thank you as well to all the developers, managers and participants at Husqvarna Group who generously shared their time, experiences and insights during the interviews. This study would not have been possible without your contribution. We also want to express our appreciation to our academic supervisor at Chalmers University of Technology, Daniel Strüber, for his continuous guidance and valuable feedback throughout this project. Finally, we would like to thank our examiner, Hans-Martin Heyn, for his time and constructive review of our work.

Samuel Falck & Cecilia Sundell, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

Since the early 2020s, the software development field has undergone a transformative shift. The ability of generative artificial intelligence (GenAI) to "understand" and write computer code from natural language prompts has gained worldwide attention and established the technology as a cornerstone of the development process in many organizations [1]. While the potential efficiency gains are evident, a picture of how developers interact with these models and the wider implications for software developer roles and organizations is only starting to emerge.

Today, GenAI is widely used by software developers. According to the 2025 Developer Survey by Stack Overflow, 84% of respondents are using or planning to use GenAI tools [2]. As this technology is relatively new and quickly evolving, empirical research regarding human-AI collaboration in software development is of significant interest. Case studies are especially valuable in this area, as they provide insights into how these tools are used and the effect they have in real-world environments. These insights are in demand by the industry as they are transferable to similar contexts and organizations [3]. This case study investigates human-AI collaboration in software development at a large Swedish manufacturing company.

1.1 Problem Description

The usage of GenAI has quickly become a regular occurrence in software development. The ongoing transformation changes workflows, skill requirements and forms of collaboration. Developers' work is changing as GenAI is implemented in more stages of the development process.

The accelerating adoption of GenAI has been driven by the promise of efficiency, but recent empirical evidence indicates that it can correlate with a decrease in stability and throughput [4]. Even if the cognitive load of programming is reduced using GenAI, the generated code comes with increased responsibilities in other areas, such as verification and integration. Without an understanding of how GenAI shapes workflow dynamics and productivity, bottlenecks may emerge in other parts of the organization and risk degrading software delivery performance, despite the significant investment in GenAI tools. This reveals a larger gap in our understanding of human-AI collaboration, demonstrating a productivity paradox between individual developer

productivity and the performance of the organization. Many large organizations lack a framework to manage this shift, and GenAI may act as an amplifier of existing organizational friction.

Contrary to the pervasive industry narrative framing GenAI primarily as an automation tool, not all developers utilize it for code generation. Instead, many leverage these models for knowledge acquisition, error interpretation and cognitive support [5]. This discrepancy suggests that existing organizational strategies and process assumptions may be inconsistent with actual usage patterns.

This makes new empirical studies in real organizations central to understanding how GenAI affects work in practice, how different developer roles are adjusting, and what skill requirements are emerging in this new environment. There is currently a lack of case studies in real industrial environments that examine how GenAI reshapes everyday development practices, especially across a full software cycle or workflow. Existing research predominantly relies on data from student populations; consequently, there remains limited empirical evidence regarding how professional developers integrate GenAI tools into their daily workflows. Research has also yet to determine the broader consequences of GenAI on communication, task ownership, decision-making and collaboration within software development teams.

Moreover, the existing research offers no consensus on measuring changes in productivity and quality when work is mediated by GenAI. Prior studies often focus on adoption rates and microtasks, rather than the broader software development lifecycle (SDLC). Additionally, the connection between individual developer behavior and organizational-level processes remains unclear, since little is known about the cognitive behaviors developers adopt when interacting with GenAI. The lack of methodological robustness also limits the ability to evaluate the transformative potential of GenAI. Given these gaps, this study aims to investigate how GenAI is changing the software development field.

1.2 Purpose of the Study

The purpose of this study is to investigate and analyze human-AI collaboration for software development at a Swedish manufacturing company with over 10,000 employees. Specifically, the study aims to:

- Explore how GenAI is reshaping developers' workflows, identity, responsibilities and required competencies.
- Understand how developers interact with GenAI.
- Investigate implications for managers and organizations.

Software development is context-dependent. It is not just about the individual programmers, but also about technology, tools, scale, end users, organizational culture, team size, and communication. To understand how software development

works in real environments, studies that maximize context realism are needed [3].

Following the ABC framework for software engineering research by Klaas-Jan Stol and Brian Fitzgerald [3], this study prioritizes the dimension of context (C) over the precision of measurement (B) or generalizability over actors (A). As noted in the paper, "the proper place to study elephants is the jungle, not the zoo" [3]. To understand the natural behavior of developers collaborating with GenAI, it is reasonable to observe them in their "natural habitat". This study will therefore be a case study, as it allows for understanding software development in concrete and realistic settings.

While the outcomes of this study are based on a specific company, this research pays special attention to the transferability of insights to similar contexts within other development settings and large organizations. By revealing the real-world complexities and potential hidden costs of GenAI adoption, this study exposes factors that are often invisible in large-sample studies or laboratory experiments that prioritize generalizability or precision over realism [3]. Furthermore, this research contributes to the wider body of empirical software engineering knowledge regarding human-AI collaboration.

1.3 Significance of the Study

The primary theoretical contribution of this study is the prioritization of contextual realism in an industrial environment. Much of the existing empirical research regarding human collaboration with GenAI relies on data from students or isolated tasks in experiments. This research contributes to the broader body of work regarding empirical case studies exploring human-AI collaboration. It also addresses the gap regarding the productivity paradox by investigating why increased individual speed in coding may not translate to organizational productivity.

For practitioners in similar contexts, particularly software developers and software engineering managers in large organizations, this study offers transferable insights into potential hidden costs and other important aspects of GenAI integration. Consequently, the results can help organizations see beyond the prevailing simple narrative that GenAI will only bring productivity gains and understand the potential risks and how to mitigate them.

1.4 Research Questions

To address the objective of this study, it is necessary to investigate the impact of GenAI beyond simple productivity metrics. Therefore, this study aims to further empirically explore the effects of its adoption. Based on the identified research gap (see Chapter 3), the following research questions (RQ1, RQ2 and RQ3) are designed to explore the transformation of software development:

- **RQ1:** How does the integration of GenAI reshape developers' roles, workflows, responsibilities, and skill requirements?
- **RQ2:** How do software developers collaborate with GenAI during different stages of the software development process?
- **RQ3:** What organizational and managerial challenges and opportunities arise when GenAI is introduced into existing development workflows?

The research questions are designed to be interconnected and provide a multilevel analysis of the effects of GenAI adoption through a bottom-up approach. RQ1 sets the baseline for developers' roles and capabilities (micro level). RQ2 explores how these evolving roles connect to practical workflows within the SDLC (meso level). RQ3 further connects this insight to explore broader systemic consequences (macro level). This structure also aligns with the recommendations of Klein and Kozlowski [6], who argue that multilevel analysis in organizations should be done at the individual, team, and organizational levels.

2

Background

This chapter establishes the contextual foundation necessary for understanding the intersection of GenAI and software engineering.

2.1 GenAI

GenAI is a form of artificial intelligence (AI) that generates data [7]. The GenAI models currently used for generating computer code, like OpenAI GPT-5.2 [8] and similar models, are almost exclusively large language models (LLMs) utilizing the Transformer architecture.

The Transformer architecture was introduced in the paper *Attention Is All You Need* [9] from 2017 and relies on attention mechanisms, which allow models to efficiently capture global dependencies in data. This enables models to compute the relevance of each word in a long sequence to every other word. An example provided in the paper is the sentence "It is in this spirit that a majority of American governments have passed new laws since 2009, making the registration or voting process more difficult". The verb "making" and the adjective phrase "more difficult" have a strong dependency between them, but they are separated by the interrupting phrase "the registration or voting process". Previous architectures, like recurrent neural networks and convolutional neural networks, would struggle to efficiently capture this dependency, but a Transformer uses attention to create a direct connection [9].

In 2018, OpenAI introduced its first Generative Pre-trained Transformer (GPT) model in the paper *Improving Language Understanding by Generative Pre-Training* [10]. GPTs are built on the idea of unsupervised pre-training on a massive corpus of unlabeled text, followed by supervised fine-tuning using labeled data. This allowed the model to leverage vast amounts of linguistic information from raw text, thereby overcoming the problem of the scarcity of labeled data for specific tasks. This strategy immediately delivered significant performance gains across a variety of natural language understanding tasks and became the standard for modern LLMs. The first GPTs focused on human language, but later models expanded these capabilities to include code in their training data, and some models were created specifically for "understanding" and generating code.

2.1.1 GenAI for Software Development

The adoption of GenAI in software development began in 2021 with the technical preview release of GitHub Copilot [11], powered by OpenAI Codex, a version of GPT-3 fine-tuned on 159 GB of public code from GitHub[1] [12]. Unlike previous tools, Copilot was able to write entire functions from a single comment.

The release of ChatGPT in November 2022 introduced conversational programming [13]. Developers could now have a back-and-forth conversation with a model to generate, debug, or explain code, and even receive advice on architectural design. If the model made a mistake, the user could point it out and allow the model to correct itself.

Cursor, an AI-assisted IDE, was released in early 2023 [14]. While GenAI tools already existed in editors, Cursor differed by being an AI-first IDE, designed with GenAI as the primary interaction method. Through a chat window adjacent to the code, the user provides instructions via prompts and GenAI generates the code; after the code is generated, the user can review and modify it in the editor. The model has access to the entire codebase by default, enabling it to "understand" the whole system rather than just a single file. This context awareness makes GenAI significantly more effective, especially in complex architectures.

Currently, there is a growing focus on GenAI agents [15]. While previous tools acted like advisors, agents act more like autonomous employees. They are given a high-level goal and autonomously determine the necessary steps, information and tools to execute the work before finally implementing a solution.

2.2 Human-AI Collaboration

The concept of humans and computers working together can be traced back to 1960, when J.C.R. Licklider envisioned a future where human brains and computing machines would collaborate in a form of symbiosis [16]. In such a relationship, humans would "set the goals, formulate the hypotheses, determine the criteria, and perform the evaluations" while computers would handle the "routinizable work that must be done to prepare the way for insights and decisions" [16]. This contrasts with a vision of complete automation, where the computer replaces the human entirely.

This relationship has evolved from traditional human-computer interaction, where users issue explicit commands to passive tools, toward human-AI collaboration. Unlike other software, AI exhibits probabilistic behavior and a form of agency, often suggesting solutions that the user did not explicitly request. This shift requires a *mixed-initiative* approach, where intelligent systems do not act as passive tools or fully automate tasks, but instead take turns with the human in contributing to the work [17]. In software engineering, this can be observed when a GenAI model

[1] GitHub is a cloud-based platform where developers store code using the version control system Git.

suggests a code completion (computer initiative) or when a developer prompts the model for an explanation (human initiative).

As GenAI models are probabilistic, they are prone to errors. To manage this, researchers propose guidelines for human-AI interaction that emphasize the need for "efficient dismissal" and "correction" of GenAI output [18]. Since GenAI models can "hallucinate"[2] and generate incorrect code, the interface must allow developers to easily verify, accept, or reject suggestions without breaking their workflow. Effective collaboration therefore relies not just on the model's capability, but also on the interface's ability to support the human in managing the uncertainty in GenAI [18].

2.2.1 Prompt Engineering

To effectively facilitate collaboration between humans and GenAI, prompt engineering has emerged as an important skill. Prompts are the instructions given to the GenAI model to enforce rules and ensure specific qualities of the generated output. White et al. [19] argue that prompts are essentially a form of programming for interaction with the model. Rather than simple queries, effective prompt engineering often relies on *prompt patterns*. These are reusable solutions to common problems and serve as a method for knowledge transfer between developers, similar to traditional software design patterns. By using these structured patterns, developers can constrain the model's probabilistic behavior and guide it toward high-quality results. Consequently, the developer's skill set must expand to include this form of natural language programming to bridge the gap between human intent and computer execution [19].

Collaborating with GenAI also shifts cognitive effort from execution to metacognition, the ability to monitor and control one's own thought processes [20]. The interaction a user has with an AI tool can be compared to a manager delegating tasks to an employee or team, as the user must explicitly define goals and verbalize implicit knowledge. Consequently, prompting encompasses not only problem formulation, but also self-awareness, detailed task delegation and decomposition, clarifying goals, and making one's implicit requirements explicit [20].

2.2.2 Vibe Coding

As GenAI becomes more popular and effective in software development, the concept of *vibe coding* has emerged. The term describes a phenomenon where developers rely almost entirely on GenAI to write the code while focusing primarily on the "vibe" (visual aesthetics, user experience and behavior) of the final product. In some cases, individuals with no prior programming experience have started to build applications, and platforms like Lovable[3] specialize in enabling users to do so via vibe coding. Vibe coding has garnered criticism, especially regarding security vulnerabilities, as GenAI can make mistakes not immediately visible in the final product. The research

[2]In the context of GenAI, *hallucinate* means confidently suggesting incorrect information or data.

[3]Lovable is an AI-powered platform that turns text prompts into functional web applications.

team at Escape investigated more than 5,600 applications developed with vibe coding platforms, like Lovable, and found more than 2,000 vulnerabilities [21].

3

Related Work

The software developer role is currently undergoing a significant shift. As GenAI automates routine code generation, the developer's focus is moving from manual implementation towards tasks centered more around architectural oversight and high-level decision support [22]. GenAI acts as a catalyst for this change by lowering the barriers to producing software. The transformation will likely evolve human-AI collaboration as the mechanical aspects of programming become increasingly automated [23].

3.1 Usage Patterns of GenAI in Software Development

A 2023 grounded theory study of Copilot-assisted programming by GitHub and Microsoft determined that there are two main modes to the usage of GenAI assistants in software development [24]. The first is *acceleration mode*, where the developer already knows what they want to do and the assistant helps with writing solutions more quickly. The second is *exploration mode*, where the developer is unsure how to proceed and uses the assistant to brainstorm solutions.

An observational study of ChatGPT practices in software development conducted in 2024 shows that the dominant use case is consultation (62.2%), where the professional developers studied asked primarily for guidance and explanations. Artifact manipulation in generating or fixing code accounts for 31.7% of the interactions [5]. Some users also reported that conversing with GenAI helps maintain focus and increases cognitive engagement for problem-solving, compared to searching the web.

While current literature categorizes and acknowledges these types of usage modes, there is still limited empirical evidence of how they integrate into the complex workflows of large organizations. Consequently, this thesis aims to investigate these practices not in isolation, but as part of a broader human-AI collaboration system.

3.2 New and Evolved Skills to Handle the Change

As the reliance on GenAI tools grows, developers need to evolve their competencies. Technical proficiency should include "AI literacy", the ability to understand, use, and critically evaluate AI [25]. Readability, the ability to quickly parse and comprehend code, becomes more important than the speed of writing. Developers are increasingly required to identify subtle errors in syntactically correct code generated by GenAI [26]. Studies have shown that skills that make developers great today are critical thinking, creativity and problem-solving [27]. The use of GenAI can reshape the relative importance of these traits as workflow transformations influence them both directly and indirectly.

The 2025 DORA report highlights the clear skepticism toward code generated by GenAI, where 76% of developers in the survey expressed *no* to *somewhat trust* in the quality of AI-generated output [4]. Furthermore, GenAI tools have a tendency to hallucinate since they generate answers based on probabilistic patterns rather than factual understanding [28]. Incorrect answers may also be due to several other reasons. One is the inability of GenAI to understand the underlying context of the question asked. Developers have admitted to feeling misled by GenAI, as its ability to write syntactically correct code with polished explanations creates the impression that solutions are implemented correctly with functional logic, even when they are not [28]. Developers therefore need to be more vigilant in detecting those hallucinations.

Beyond the shift in technical skills, there is also a concern regarding the long-term impact on career progression for junior developers [29]. A competence gap emerges when junior developers struggle to acquire basic expertise, as the entry-level tasks are automated by GenAI. Organizations cannot rely solely on individuals' own learning, but must establish a framework for mentoring and strategic GenAI use to avoid this workforce polarization [29].

Some developers have also expressed concern that their role might be replaced, or that by using these tools and encouraging their organization to use them efficiently, they are actively contributing to their own obsolescence [30, 31].

The necessity of evolving technical skills is established, but it remains unclear how this shift will impact the developer's professional identity and self-perception of value. This study addresses this gap by analyzing the transition of the individual developer's changing skill set and responsibilities.

3.3 The Emerging Bottlenecks

Trust and verification appear as new potential bottlenecks. Interactions with GenAI introduce a new dynamic where developers must assess and validate the generated suggestions. This verification may be insufficient, and its implications for quality, productivity and team workflow are unclear [5]. As stated in the *Theory of Con-*

straints, every system has at least one constraint that will limit its total output [32]. Improvements made elsewhere are illusory and will not cause increased organizational performance. Generating code faster will hence not automatically improve delivery speed.

Rigorously double-checking suggestions through manual review takes time and effort, raising concerns about the overall cost of this verification. The productivity gained from the generated code may be partially offset by the time required to verify its output. An experimental study by Vaithilingam et al. [26] suggests that developers struggle to debug AI-generated code because they did not write it themselves and therefore lack the intuition about where potential bugs may be located. This resulted in the participants deleting the entire generated code almost half of the time, rather than attempting to fix it, as it was deemed easier than debugging. Developers also needed to constantly switch contexts between writing and reading code, which increased their mental load.

Code churn refers to the amount of code that has been written and then reverted or substantially changed within two weeks. Current trends show that the amount of code that has been "churned" has approximately doubled between 2021 and 2024 [33]. This contradicts established software engineering best practices. For example, instead of using abstraction to utilize already created code, GenAI is likely to generate the same logic in multiple places, increasing duplication and technical debt, which makes the system more difficult to maintain in the long run.

According to DORA's 2025 report, many of the downstream bottlenecks appear due to the large batch size of code that developers are able to generate using GenAI [4]. This implies that GenAI does not accelerate existing agile workflows but rather changes them to accommodate the increased output of GenAI in later phases of the development process.

Upstream bottlenecks have also been discovered in requirements engineering, as the quality of the GenAI output is dependent on the quality of the input prompt [34]. The challenge shifts from *how* a solution should be coded to defining exactly *what* to build. AI tools require unambiguous and detailed instructions to deliver correct results, which takes a significant amount of time to formulate what is actually to be accomplished.

Current studies have identified several bottlenecks but offer limited insights into the managerial strategies required to mitigate them. They also often examine these issues in isolation rather than within a complex ecosystem of large organizations. Therefore, this study will view these constraints through an organizational lens, exploring challenges and opportunities for existing workflows that likely must be adapted to leverage GenAI.

3.4 The Productivity Paradox

The gap between perceived and actual productivity represents another paradox, where individual gains do not automatically translate to team-level improvements. There is a perception-reality gap, demonstrated in some studies where users felt more productive using GenAI tools, despite the lack of a statistically significant reduction in task completion time [26, 35]. This implies that the productivity paradox is connected to job satisfaction, as the feeling of satisfaction from achievement could still increase even though efficiency and output remain the same. This further complicates how organizations measure results, Return On Investment (ROI) and Key Performance Indicators (KPIs).

However, this discrepancy can be linked to the reduction in cognitive load as the mental effort to perform tasks is decreased, making developers feel another type of benefit from using GenAI tools. According to Forsgren et al. [36], the Developer Experience (DX) is highly related to developer productivity. Their framework, called SPACE, accounts for different types of metrics such as employee satisfaction and performance evaluated as outcome instead of output. Activity, communication, and efficiency are the other main categories for included specified metrics that will, in combination with the previous, form a holistic understanding of developers' productivity. Individual speed gain can lead to degradation in overall stability later on [4]. This may change management strategies from *local optimization* (coder) to prioritize *global optimization* (the pipeline). While individual metrics may show improvements, it is important to see the overall picture and measure the entire value stream.

When individual coding tasks are accelerated, the increased development speed also creates tension with organizational risk management, simultaneously slowing down expected productivity gains by forcing organizations to introduce more extensive review processes. Managers have recently raised concerns regarding security, data privacy, and IP rights surrounding the generated artifacts and usage of the GenAI tools [37]. They are also worried about the long-term maintainability of the code, as well as its trustworthiness. They acknowledge that the generated code still requires significant human verification, increasing the expectations on employees' skills, as they must also be able to verify and take responsibility for the generated code. These additional layers are further affected by active investments in human capital, as the productivity gains are also likely to be lost due to employees' lack of experience with new tools. Therefore, there exists an increased need to allocate time and resources for learning, e.g., through workshops or other continuous learning initiatives [37].

This study is designed with this paradox in mind by addressing the tension between local optimization and global efficiency. The research questions are selected to navigate beyond simple metrics to capture a holistic understanding of the problem, and aim to uncover how organizations can bridge the gap between perceived productivity gains and actual system-level impact.

3.5 Other Case Studies

A pilot case study conducted in late 2023 examined the integration of GenAI in a large software company [38]. Participants with no prior regular use of GenAI in their work were given licenses for several tools. The participants started to use the tools to acquire new knowledge and optimize workflows by resolving technical issues, generating simple code, assisting in brainstorming, and aiding in formal writing of different documents and reports. The study found that the tools had an overall positive impact due to saving time on tasks and making knowledge acquisition easier by providing quick access to information. There were, however, challenges regarding the accuracy of responses, the need for some manual adjustments, and sensitive project data sometimes preventing the use of the tools for security reasons.

A case study at a large Brazilian company conducted in 2024 surveyed and observed developers to understand how GenAI tools impact professional software development beyond the initial hype [31]. The study found that GenAI was widely adopted for technical tasks, especially those that are code-intensive, and the overall impact was positive. There was an increase in productivity, as Jira[1] task data showed a 23% reduction in the average development cycle. The developers' experience with GenAI was positive as enjoyment of coding was increased, mental effort in repetitive tasks was reduced, and the tools were perceived as helpful for learning purposes. However, the feedback regarding the quality and reliability of GenAI suggestions was more varied, and its use in non-coding tasks, such as architecture, design and requirements, was limited. There were also concerns regarding overreliance and negative long-term effects on problem-solving skills.

A case study conducted in 2025 examined the adoption of Google Gemini at a large international IT company [39]. It found that GenAI became an integrated part of the developers' workflow and proved useful for offloading certain high-volume repetitive tasks, like test cases, code templates, scripts and bulk search-and-replace. Participants also described it as a replacement for Stack Overflow[2], as in-context prompts are preferable to searching through forums. As in the previous study, the developers reported an increase in productivity.

[1] Jira is a project management and issue-tracking tool developed by Atlassian used by software development teams to plan, track and release products.

[2] Stack Overflow is a question-and-answer website where programmers share knowledge and solve coding problems.

4

Methodology

This chapter details the methodology employed to investigate the research questions. It begins by outlining the research strategy, which utilizes a single-case study design with embedded units of analysis to capture diverse technical workflows within the organization. Following this, the data collection process is described, detailing the use of purposive sampling and semi-structured interviews to gather insights. The chapter then explains the inductive thematic data analysis applied to evaluate and code the interview transcripts. Finally, it establishes the delimitations of the study, clarifying its focus on human-AI collaboration within a complex organizational context rather than specific tool benchmarking or governance frameworks.

4.1 Research Strategy

To capture the diversity of software development within the organization, this study adopted a single-case study design with embedded units of analysis. In embedded case studies, there is a distinction between the case itself and the embedded units of analysis, which are the entities within the case that are analyzed to answer the research questions [40].

In this instance, the case was the company, and the units of analysis were the technical domains along with the managers. This allowed for a detailed examination of how GenAI integration might differ across technical contexts. The case was divided into the following units:

- **Unit A (Full-stack)**: Developers working across the entire stack, including frontend and mobile developers.
- **Unit B (Backend)**: Developers primarily focused on backend infrastructure.
- **Unit C (Embedded)**: Developers working with embedded systems.
- **Unit D (QA/Testers)**: Quality assurance engineers and testers.
- **Unit E (Managers)**: Managers and team leaders.

These units were defined using a purposive sampling strategy to achieve maximum

variation, covering diverse workflows and technical environments. For instance, Unit C (Embedded) operated under safety-critical and hardware constraints, where systems could be jeopardized by GenAI hallucinations, whereas Unit A (Full-stack) focused on user-facing rapid iteration. While some of these units may constitute a larger proportion of the company's workforce, this purposive sampling was done to support analytical rather than statistical generalization [41].

Each participant was selected with the requirement that they belonged to a *main unit* based on their primary tasks. However, because responsibilities within these units may overlap (e.g., some developers in Unit A (Full-stack) may also perform backend and testing tasks), some participants were also assigned *secondary units* if deemed necessary. These units also influenced which questions were asked, as detailed in Section 4.2.1.

4.2 Data Collection

To ensure validity through data source triangulation, insights within each unit were collected from multiple perspectives, including junior and senior developers from different teams and locations. This ensured that the data captured a broad range of viewpoints and mitigated the risk of bias toward a single group [40]. To facilitate this, individual participants were selected using a purposive sampling approach [41]. Participants were recruited through nomination by a contact within the organization, rather than volunteering, to ensure they matched the desired profiles.

The data were collected through 25 digital interviews; Table 4.1 shows the number and percentage of participants in each unit.

Unit	Number	%
A: Full-stack	9	36%
B: Backend	4	16%
C: Embedded	5	20%
D: QA/Testers	3	12%
E: Managers	4	16%
Total	25	100%

Table 4.1: Units of the participants.

The number of participants is not equal across the units. The primary reason for this is that the number of employees in each unit varies within the organization, which naturally means the number of participants also varies. Another reason is that participants in some units, especially Unit A (Full-stack), gave more diverse answers, which prompted additional interviews to gain more unique insights. To mitigate any bias based on units, the results will be presented from each unit separately where large differences in how different developers answered were observed.

The questions were tailored to specific roles (e.g., managers received questions regarding GenAI's impact on management). The specific interview questions were

established at the start of the project, and inspiration was drawn from the SPACE framework [36]. The interviews were semi-structured with both open and closed questions. The session structure followed the funnel model, starting with broad, open-ended questions and narrowing in scope as the interview progressed [40].

Sessions were recorded and subsequently transcribed using AI tools. Relying on automated transcription rather than manual note-taking allowed for a natural conversation and a focus on relevant follow-up questions. All transcripts were manually reviewed for accuracy, and all subjects explicitly provided informed consent. Confidentiality was maintained for all participants, and they remained anonymous in all reporting.

4.2.1 Interview Questions

The interview questions were designed through an iterative process in which the research questions were broken down and operationalized into discussion points. As mentioned in Section 1.4, RQ1 explores developers' roles and workflows, RQ2 investigates their collaboration with GenAI, and RQ3 addresses organizational and managerial challenges and opportunities. The purpose of breaking down and operationalizing the research questions was to ensure that each one was covered by one or more interview questions while leaving space for new insights.

A systematic mapping of the interview questions (see Figure 4.1) was performed to ensure that no relevant topics were omitted and to facilitate sorting the results and supporting later discussions. Each of the three research questions was broken down into its core components. Specifically, RQ1 was divided into role, workflow (WF), responsibilities (Resp.), and skills; RQ2 into collaboration (Collab.) and phase; and RQ3 into organization (Org.) and management (Mgmt.). The interview questions were linked to the research questions they were intended to address, marked as "X" when primarily addressing a research question and "(x)" when addressing the research question secondarily. The questions drew inspiration from the dimensions of the SPACE framework, ensuring aspects such as satisfaction and the outcomes of performance and efficiency were covered [36].

The interview guide was structured to start with broader, contextual questions about the participant's role and general usage of GenAI, for example, "*What do you think about the use of AI in software development?*". After this, the interview started addressing more complex and specific topics through questions such as, "*Have you developed any specific strategies for how you write your prompts?*". The middle of the interview (inserted after question 7) consisted of a modular section with one or two domain-specific questions suited for the specific units (including secondary units) to which the participant was assigned (see Section 4.1). For example, embedded systems developers were asked the question "*Do you feel that the AI models understand the context of your target hardware?*". The complete interview consisted of 13 open questions and one to two domain-specific questions. All interview questions can be found in Appendix A.

	RQ1				RQ2		RQ3	
Question	**Role**	**WF**	**Resp.**	**Skills**	**Collab.**	**Phase**	**Org.**	**Mgmt.**
1	X		(x)					
2					X			
3		X			X	(x)		
4		X		X	X			
5		X				X		
6	(x)	X				(x)	(x)	
7			X		X			
A1	X			(x)		X		
B1	(x)		X			X		
B2		(x)	X		(x)		(x)	
C1					(x)		(x)	
D1	X	(x)			X			
E1								X
E2				(x)	(x)			X
8	X		(x)		(x)			(x)
9				X	(x)			(x)
10					X		X	
11					X		X	
12	X			(x)			(x)	
13								

Figure 4.1: Mapping of Research Questions and Interview Questions

The interviews were conducted using a semi-structured approach, meaning the interview guide served as a framework rather than a fixed script. The order of questions was adjusted dynamically depending on how the conversation developed. If a respondent spontaneously touched on a topic intended for later in the session, this was discussed immediately to maintain the flow. To ensure the relevance and appropriate depth of the responses, follow-up questions were used in cases where the initial response was too general or did not address the research questions. For example, the question "*How much trust do you generally feel regarding the code the AI produces?*" had the follow-up question "*How much of your time goes into verifying AI code/output compared to code you wrote yourself?*". Space was also given to explore interesting side topics, provided that they contributed to the overall purpose of the study.

4.3 Data Analysis

To analyze the qualitative data from the interview transcripts, a systematic, bottom-up approach was employed to derive conclusions while maintaining a clear chain of evidence. The transcripts were coded following an editing approach (also known as inductive thematic analysis), in which codes were defined based on findings during the analysis [40]. The procedure consisted of the following steps:

1. The transcripts were analyzed line-by-line to extract codes representing answers and specific concepts, behaviors, and sentiments related to GenAI usage.

2. The codes were discussed and filtered. The conversational back-and-forth format of the interview made the transcripts easy to interpret when reading them, leading to few coding disagreements.

3. The codes were categorized by the interview questions. Because of the semi-structured and conversational format, participants sometimes mentioned something in one interview question that was relevant to another question. Consequently, codes from one question sometimes were grouped with another question.

4. The code categories were integrated into themes and sentiments that directly addressed the research questions. This process was facilitated by the codes now being grouped by interview questions, which in turn were already mapped to research questions.

5. After broader themes and sentiments were established, the analysis moved toward hypothesis generation to explain the observations and answer the research questions.

At the start of the process, the answers were grouped by the main unit of the participant; however, when the results for a question were very similar across several units, or based on the specific intent of the question, the results were combined. While the interviews focused on qualitative data, the results for some questions could be summarized as binary (yes/no) responses or on a scale of 1 to 3.

4.4 Delimitations

The study was delimited to investigating the usage of GenAI in the context of large organizations, where software development is often characterized by complex dependencies, extensive legacy code, technical debt and established procedural bottlenecks. Human-AI collaboration can take different forms depending on the size of the organization, department, or team. The study aimed to take this complexity aspect into account rather than focusing solely on coding in isolation. This meant that the study prioritized empirical breadth over in-depth technical details within individual development elements.

No comparative analysis or benchmarking was conducted between different GenAI tools. The justification for this decision was that the integration and use of specific tools were highly dependent on organizational conditions. Furthermore, the rapid pace of technological advancement led the study to focus instead on identified collaboration patterns and developers' professional identity in relation to GenAI. These aspects are often considered more stable than any tool's specific performance, an evaluation of which would require different research questions.

Ethical and legal aspects, such as security policies, intellectual property rights and data protection, were excluded from the core analysis. Although developers worked within these regulatory frameworks, an in-depth evaluation of GenAI governance fell outside the objective of this study. Legal dimensions were important but would have overextended the scope of the thesis, which focused on developer workflows.

5

Results

This chapter presents the results collected in the study. The results are based on data collected from 25 interviews conducted from February to March 2026 with different types of developers and managers to provide a broad and nuanced view of the organization. The chapter is structured around the study's research questions, which follow a progression from the individual (micro), to the team (meso), and finally to the organizational (macro) levels. First, the chapter focuses on presenting the individual developers' daily usage of GenAI, workflows, professional roles, and skill requirements. Then, the perspective is broadened to highlight how GenAI affects collaboration. Finally, the chapter shows how GenAI affects the overall organization, with emerging bottlenecks, threats and opportunities.

5.1 The Integration of GenAI in the Developer Role (RQ1)

This section shows the results of the study's first research question: "*How does the integration of GenAI reshape developers' roles, workflows, responsibilities, and skill requirements?*"

The section starts by describing how the professional role and tasks are changing, including the individual experience of GenAI adoption and the accompanying pressure. It then presents the impact on perceived productivity, with a focus on frequency of use and which tasks developers save time on. It also presents how developers perceive the reliability of GenAI. Finally, it maps out how these factors together are driving the shift in skill requirements for the developer role in the future.

5.1.1 The Evolution of the Developer Role

With the rise of GenAI, developers have started to view themselves more as "code reviewers" or "software architects" rather than just "coders", as seen in Figure 5.1. These terms represent a step away from primarily writing code manually and toward overseeing, reviewing, planning, and orchestrating the work of GenAI models. One participant in Unit A (Full-stack) even reported not having written a single line of code manually in months. Others have started to view GenAI as another higher-level abstraction in software engineering, similar to how high-level languages serve as

an abstraction for low-level languages. There is, however, a noticeable difference between units; the majority of those who still see themselves as coders are developers from Unit C (Embedded). The theme of Unit C (Embedded) lagging behind in the adoption of GenAI is repeated several times throughout the results.

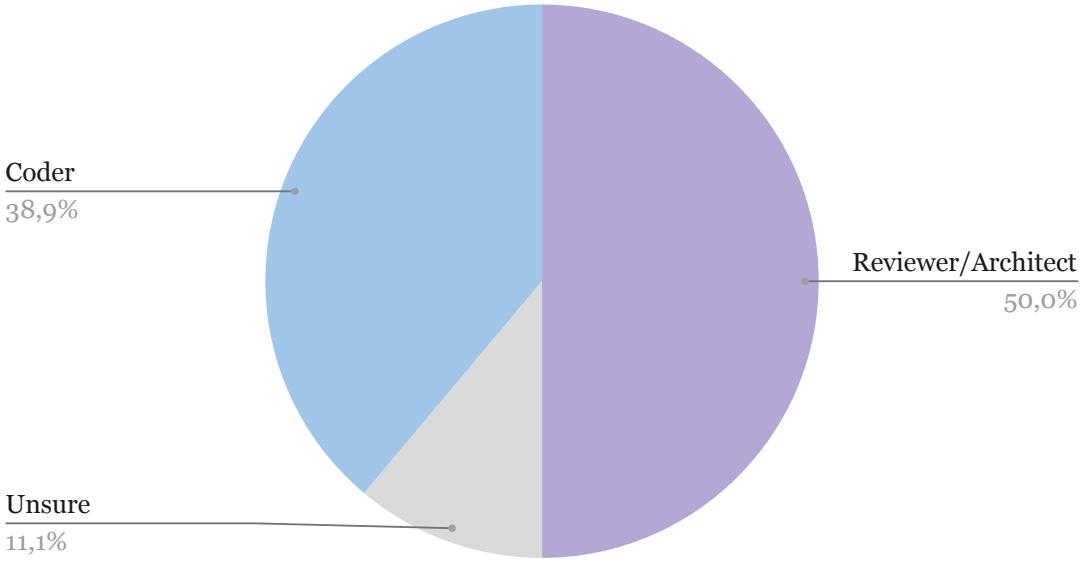

Figure 5.1: The percentage of developers primarily viewing themselves as coders or code reviewers and software architects. Unit D (QA/Testers) is excluded since they already mostly focus on review, and Unit E (Managers) is excluded because they do not work directly with code.

Because GenAI can automate much of the coding, developers are enabled to shift their focus further "upstream" in the SDLC process. This shift includes taking over tasks from product owners (POs) and project managers (PMs), and allows developers to spend more time on requirements gathering and engineering. Most participants, including managers, believe these roles will merge in the future. They anticipate a greater focus on determining what to build, as the implementation of the features or products will become less time-consuming and, consequently, not be a bottleneck in the process. GenAI also lowers the barrier to performing tasks outside of one's primary technical domain. Several participants observed that, rather than specializing in siloed roles, they are becoming generalists with responsibilities across multiple areas. One participant expressed that they are starting to view themselves "as all roles at once".

There is no clear consensus that GenAI will help lower the cognitive load of developers; the responses are mixed, with several even leaning toward the negative. Some participants believe the technology could help by eliminating the need to retain vast amounts of information simultaneously or constantly shift between tasks. Conversely, others note that stress might actually increase due to this shift and express a preference for focusing on a single task at a time.

When asked if they feel pressure to deliver more now that GenAI is available, almost

all participants responded “no”. So far, there is no clear increase in expectations from management, though some believe this will change in the future. Some participants already feel external pressure to keep pace with rapid developments across the broader industry. Additionally, roughly half of the participants in Unit A (Full-stack) expressed anxiety regarding the future, fearing that GenAI might threaten software developer roles. Junior developer positions appear to carry the highest perceived risk of disappearing.

Participants agree GenAI will not completely replace humans in software engineering. Human involvement will remain necessary because the end products are ultimately made for other humans. Current trends point toward smaller teams where the remaining developers assume a significantly broader scope of responsibility as GenAI automates much of the manual coding. However, embedded systems development currently lags behind in this evolution.

Summary of The Evolution of the Developer Role

Manual coding is decreasing, the roles of developers are shifting from coders towards code reviewers and software architects. There is a growing focus on orchestrating and supervising GenAI models that write the actual code.

Embedded systems developers lag behind in this transformation.

Developers are taking over tasks from POs and PMs.

There is no clear consensus that GenAI currently helps lower the cognitive load of developers.

Developers do not feel internal pressure to deliver more or faster from within the organization, but external pressure exists.

While the role of software developer does not seem to disappear entirely, current trends point towards smaller teams where the developers assume a broader scope of responsibility.

5.1.2 Impact on Workflow and Productivity

The integration and frequency of GenAI usage vary among the different units, but there are no participants who abstain from using GenAI entirely. The differences observed relate strictly to the frequency of usage as seen in Figure 5.2 (and potentially specific use cases), rather than absolute adoption. To categorize the usage frequency, the study uses the following defined scale:

- **Rarely:** Around three requests per day or less.
- **Often:** More than three requests per day, but for less than the majority of the day. This level of usage is often characterized by some continuous discussions with the GenAI during the day.

- **Continuously:** GenAI is used all the time or for the majority of the day. This level involves near-constant, continuous discussions with the GenAI.

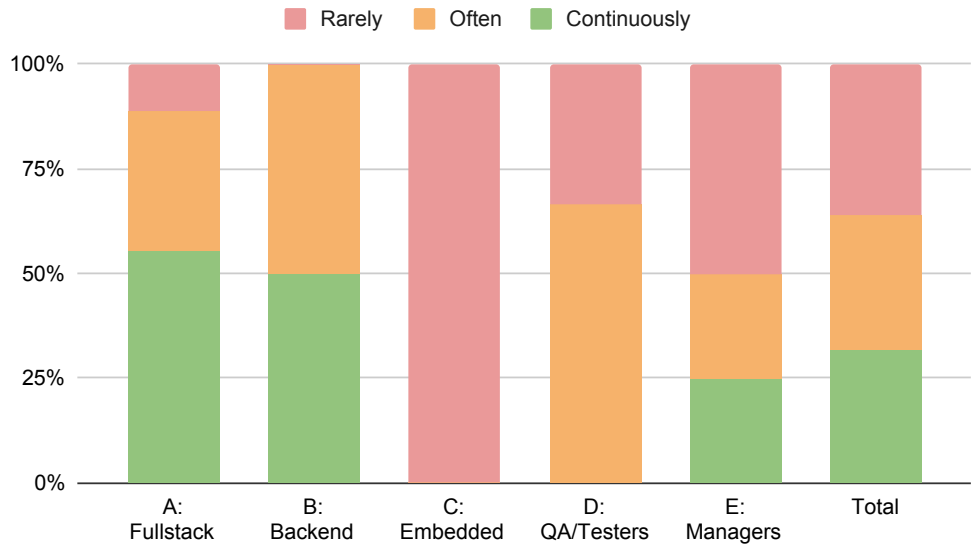

Figure 5.2: Self-reported GenAI usage frequency across units (normalized)

Overall responses across all units are relatively evenly distributed among all frequency categories. Units A (Full-stack) and B (Backend) show similar patterns, utilizing GenAI with comparably high frequency. Developers from Unit C (Embedded) report the lowest usage overall, with all participants in this unit categorizing their frequency as "Rarely". Unit D (QA/Testers) utilizes GenAI more frequently than Unit C (Embedded), but less extensively than Units A and B. Unit E (Managers) shows a mixed distribution of usage frequencies.

Time Saving	A	B	C	D	E	Total	%
Yes	7	4	5	2	2	20	80%
Unsure	2	-	-	1	1	4	16%
No	-	-	-	-	1	1	4%

Table 5.1: Perceived time saving per unit.

There is a strong consensus among developers regarding their perceived increase in productivity and time savings. As seen in Table 5.1, 80% believe that they have become more productive with the help of GenAI tools. The remaining responses show uncertainty and only one manager explicitly believes that GenAI does not save time. Managers are generally more skeptical than the developers and refrain from confirming increased productivity on the grounds that it is still too early to determine due to the lack of concrete evidence.

When asked about which types of tasks GenAI helps developers save time on, the responses have been categorized into three main areas:

1. **Automation of Repetitive Work:** The primary efficiency gains reported from all units come from automating repetitive work. This is, for example, done by generating boilerplate code, handling syntax and writing unit tests. Developers from Units B (Backend) and C (Embedded) describe how tools such as autocomplete significantly reduce the need to manually copy and paste syntax. This creates a smoother and more pleasant coding experience for them.

2. **Cognitive Offloading and Context Switching:** Developers describe GenAI as an important search engine for information retrieval and for problem-solving. They are also faster at overcoming blockers and explain that by asking GenAI instead of a colleague, they progress more quickly in the code implementation. Unit B (Backend) reports that this also saves time for senior developers since they are being relieved of answering simpler questions and therefore avoid unnecessary interruptions. In addition, GenAI helps with context switching among the developers. For Units A (Full-stack) and B (Backend), this consists of switching between tasks and being able to understand other people's code faster. Unit C (Embedded) describes it as being able to switch faster between different syntax and programming languages.

3. **Front-loading of Productivity:** Another interesting highlight is that time savings can occur early in the SDLC, rather than just in the implementation and testing phases. Units A (Full-stack), B (Backend), and D (QA/Testers) report time gains in ideation, design and planning. Unit A (Full-stack) describes using the tools to decide on architectural trade-offs and create prototypes, while Unit B (Backend) mostly uses the tools for evaluating alternative solutions and sorting ideas earlier.

Participants were later asked about what happens with the time that they reported having saved. Their responses show an interesting insight: efficiency does not lead to decreased workload as time is immediately reinvested in the development process. Unit A (Full-stack) describes that their time and focus shift towards understanding domain knowledge and the requirements. Managers also note that this means that the bar for what constitutes acceptable quality and a minimum viable product (MVP) has been raised. One participant relates this dynamic to the Jevons paradox: when a resource becomes more efficient to use, its consumption increases. The time saved from code generation is therefore used to achieve a higher quality result in the same allocated time. This marks a shift from just referring to productivity and time savings, towards a focus on more complex dimensions in maximized value creation.

Summary of Impact on Workflow and Productivity

The frequency of using GenAI is dependent on the unit, with developers in embedded systems using it the least often.

80% of participants report saving time because of GenAI; others refer to a lack of evidence of increased productivity.

Developers save time on:
1. Automation of repetitive work
2. Cognitive offloading and context switching
3. Front-loading of productivity

The time saved is generally reinvested in the development process to increase quality.

5.1.3 Trust in Output of GenAI

This section examines the level of trust developers have in the output generated by GenAI. Their perceptions are categorized into three tiers based on the perceived reliability:

- **High reliability (output requires minimal modification):** The output is consistently accurate and contextually relevant. Only minor tweaks or formatting adjustments are required.
- **Moderate reliability (output requires significant modification):** The output provides a functional foundation but requires manual editing or logic corrections to be usable.
- **Low reliability (output requires reconstruction):** The output is frequently incorrect or irrelevant. The effort required to fix the output is often equal to or greater than writing the code manually.

Figure 5.3 illustrates the varying levels of trust each unit places in GenAI. Unit E (Managers) was excluded from this analysis as they do not work directly with code.

Unit B (Backend) exhibits a quite high level of trust, yet remains somewhat skeptical about utilizing autonomous GenAI due to the associated risks. One participant noted that while the output might be 90% correct, the remaining 10% can cause significant problems, rendering it insufficient for autonomous use. Another participant highlighted that collaborating directly with GenAI is difficult and that the coding process is largely binary: either the GenAI generates the code entirely, or the developer writes it independently. Furthermore, Units A (Full-stack) and B (Backend) both agree that verifying AI-generated code often takes longer than verifying manually written code.

Unit C (Embedded) is the most skeptical, demonstrating low to moderate trust. This aligns with other findings in this study regarding GenAI's limited capabilities

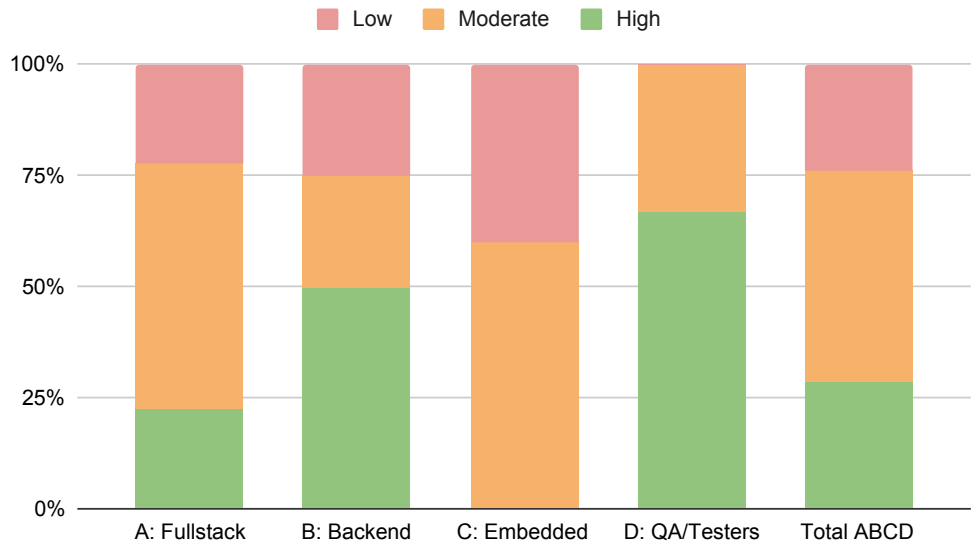

Figure 5.3: Perception of reliability in GenAI across units.

in embedded systems. Several developers reported placing less trust in GenAI when working with low-level programming, and higher trust during high-level programming. Unit D (QA/Testers) demonstrates the highest overall trust in GenAI, but even they maintain that GenAI output cannot fully replace all testing.

In conclusion, participants generally lean toward moderate trust, with confidence levels varying across different units. This consensus indicates that while GenAI output provides a solid foundation, it still requires manual review and frequent editing. All units agree that human oversight remains essential, as developers ultimately bear responsibility for the final code.

Summary of Trust in Output of GenAI

Overall, participants lean toward moderate to high trust. GenAI is viewed as providing a solid foundation, but human oversight and manual review remain essential since developers ultimately bear the responsibility for the final code.

Unit C (Embedded) is the most skeptical group, demonstrating low to moderate trust in GenAI.

5.1.4 Shifting Skill Requirements

The integration of GenAI in software engineering entails a shift in which skills are considered important for developers. This section will present the skills that the participants believe will become less or more important due to these changes.

Less important Skill: Syntax memorization

A consistent pattern across all units is that the value of syntax specialization and manual code writing is generally reduced. Developers describe that GenAI is increasingly taking over the generation of code. Therefore, the need to memorize syntax for specific frameworks or languages is decreased. They also mention that they do not need to be as skilled at formulating perfectly written code from scratch, since GenAI is good at providing boilerplate code. The threshold for working across multiple domains is also considered lowered, reducing the risk of becoming locked into a specific technology platform.

Important Skill 1: Systems thinking and code review

Developers emphasized that domain knowledge and having a holistic systems mindset have become more important. A deeper architectural understanding is required to be able to successfully and efficiently integrate the larger volumes of code that GenAI generates. The generated code pieces are to be fitted into their broader system, hence the developer also needs to be able to understand system complexity. Participants mention that this is connected to their emerging identity as "Quality Curator". The ability to analyze, read, and review code is valued more than ever, as well as critical thinking, which is important to identify logical errors and potential hallucinations.

Important Skill 2: Defining intent and clear requirements

The ability to define intent and effectively break down requirements is becoming increasingly important. Developers describe that they are starting to be expected to understand customer needs and to be able to make decisions and design choices based on the customer's underlying problems. Several responses to the interview questions show that knowledge of what to build now often outweighs knowledge of how to code it.

Important Skill 3: Efficient communication with AI and humans

The rise of GenAI is transforming software development into a more communicative practice in two ways. Firstly, the developers describe the importance of skills surrounding AI-literacy and knowing how to interact with GenAI tools, connecting to concepts like prompt engineering, context engineering and AI orchestration. Secondly, traditional communication skills are becoming more important as the developer role moves closer to the domain of product owners and focuses more on gathering requirements.

Soft skills that matter

In addition to the specified skills above, some adaptive soft skills are emphasized by some participants as having increased importance. These are the following: curiosity, learning capabilities, and being able to independently break down one's own tasks. These are also considered factors for having an easier time adapting to the AI transformation.

The collected data also highlights an interesting contradiction around future skill

profiles. While there are indicators from all units that the developer role is becoming less technical and broader, developers describe moving toward a more abstract methodology similar to vibe coding and being a "jack-of-all-trades". In contrast, developers from Unit B (Backend) argue that the developer role will move towards more specialization in order to be able to solve complex problems GenAI cannot handle. A managerial perspective reports that future developers will be required to have a strong technical understanding, but manual coding is being automated away. However, they still predict that the demand for developers who possess both a high-level abstract understanding and a deep technical niche will increase.

Finally, a notable exception from the overall trends in skill requirements is reported by Unit C (Embedded), where they feel that their skill requirements remain largely unchanged.

Summary of Shifting Skill Requirements

Skills becoming less important:

1. Syntax Memorization

Skills becoming more important:

1. Systems Thinking and Code Review
2. Defining Intent and Requirements
3. Communication
4. Curiosity, Learning Capabilities, and Task Breakdown

5.2 Human-AI Collaboration (RQ2)

This section presents the results of the study's second research question: "*How do software developers collaborate with GenAI during different stages of the software development process?*"

The section starts with a review of the development phases in which GenAI is used, followed by a mapping of more specific use cases among units. It later examines how GenAI is used in practice, focusing on the specific strategies and approaches the participants are using. Then, it highlights the technical problems and obstacles limiting the developers' workflows. Finally, the section concludes with collected thoughts and ideas from the participants on how human-AI collaboration can be improved in the future.

5.2.1 Usage Patterns across SDLC Phases

This section examines GenAI usage patterns across the SDLC. For the purpose of this analysis, the SDLC is divided into the following phases:

- **Ideation:** Exploring ideas and solutions at a high level.
  For example, using GenAI to brainstorm, formulate ideas and create prototypes.

- **Requirements Engineering:** Translating the idea into functional requirements (what the system must do) and non-functional requirements (performance, security).
  For example, using GenAI to generate user stories or define requirements from ideas.

- **Planning:** Focusing on how the requirements are to be implemented.
  For example, using GenAI to plan implementation or task breakdown.

- **Coding:** Writing source code.
  For example, using GenAI to generate code for a new feature.

- **Refactoring:** Specifically improving the internal structure of existing code without changing its behavior.
  For example, using GenAI to increase modularity and reusability.

- **Debugging:** Identifying, tracing, and resolving errors and bugs in the code.
  For example, using GenAI to identify and solve bugs.

- **Testing:** Verifying that the software works as intended.
  For example, using GenAI to generate unit tests, integration tests, or other test scenarios.

- **Documentation:** Creating guides or comments based on project artifacts.
  For example, using GenAI to generate README files or documentation.

Deployment and monitoring are not featured as a phase due to it being the responsibility of a specific DevOps engineer role, and there are no DevOps engineers as participants in this study.

Table 5.2 shows the percentage of each unit that uses GenAI during each software development phase. It also includes a column (ABCD) with the combined data for all participants except managers. This is done as the role of managers differs significantly since they do not work directly with code.

The data show that GenAI is mostly used for coding; 90% of participants, excluding managers (Unit ABCD), use GenAI in some capacity for coding. The second most common phase for these same participants is testing, at 81%. An interesting observation is that although almost all developers use GenAI for writing new code, only 38% use GenAI for refactoring old code. It can also be observed that only around half of Unit ABCD use GenAI for debugging.

The data reveal some differences regarding how different units use GenAI. Units A (Full-stack) and B (Backend) use GenAI for ideation and planning more frequently than the other units, although they differ significantly when it comes to debugging and documentation, as Unit B (Backend) uses GenAI significantly more for those purposes.

Table 5.2: The percentage of each unit that uses GenAI in each software development phase.

Phase/Unit	Total	A	B	C	D	E	ABCD
Ideation	48%	78%	75%	40%	-	-	57%
Requirements	16%	-	50%	-	-	50%	10%
Planning	48%	78%	75%	20%	-	25%	52%
Coding	76%	100%	75%	80%	100%	-	90%
Refactoring	32%	33%	50%	40%	33%	-	38%
Debugging	40%	33%	75%	40%	67%	-	48%
Testing	68%	89%	75%	60%	100%	-	81%
Documentation	32%	11%	75%	40%	33%	25%	33%

Note: For Unit D (QA/Testers), coding and testing are regarded as the same phase.

Developers from Unit C (Embedded) mostly use GenAI for coding and to some extent for testing, and less frequently for other tasks. All participants from Unit D (QA/Testers) use GenAI for testing, and a majority use it for debugging. No one in Unit D (QA/Testers) uses GenAI for ideation, planning, or requirements engineering.

Requirements engineering saw the lowest GenAI adoption rate, at only 16%, and it can also be observed that backend developers and managers are the only units that use GenAI in this phase. Half of the participants from Unit E (Managers) reported that they use GenAI for creating requirements (compared to 10% for Unit ABCD), making it the phase where managers use GenAI the most. It should also be noted that many managers do not use GenAI in any specific phase, but instead for other miscellaneous tasks, such as writing emails and translating texts. Still, it appears that managers have not adopted GenAI for the same purposes as the other roles.

Summary of Usage Patterns across SDLC Phases

GenAI is predominantly used for coding and testing, and least often in requirements engineering.

There are differences in GenAI usage between units:

- Units A (Full-stack) and B (Backend) use it more for ideation and planning.
- Unit B (Backend) uses it more for documentation.
- Unit B (Backend) and Unit D (QA/Testers) use it more for debugging.
- Unit A (Full-stack) and Unit E (Managers) are the only units using it for requirements engineering.

5.2.2 Specific Use Cases

The specific use cases for GenAI differ somewhat between the units, although there is some overlap. Participants in all units agree that GenAI can be utilized as a senior colleague with extensive knowledge. When faced with a problem they do not know

how to solve and lack the required knowledge for, developers now have the option to consult a GenAI model before asking a real colleague. This is often preferred as a first step when encountering difficulties, as it is more convenient and does not consume colleagues' time. One participant in Unit C (Embedded) even noted that they could now take on problems and tasks outside their role with the assistance of GenAI. Consequently, many participants see GenAI as a tool for learning new concepts, as it can easily provide specific knowledge by acting as a mentor that answers any questions that arise, as well as providing step-by-step guides when performing a new task for the first time. It is, however, important to note that GenAI cannot solve all problems and cannot completely replace the role of a senior colleague.

Many participants across units mention using GenAI to generate boilerplate code[1] and analyze logs. Some other overlapping use cases across the units are using GenAI as a search engine (for both code documentation and general information) and as a rubber duck[2] for debugging. However, the most common use case overall is to utilize the GenAI autocomplete feature most IDEs provide, as it is highly convenient when making small modifications, such as changing variable names.

Units A (Full-stack) and B (Backend) have started to use GenAI to evaluate architecture solutions and explore design trade-offs in the earlier development phases (see 5.2.1). Some participants in Unit A (Full-stack) mention that they sometimes provide GenAI models with the requirements and allow them to generate a plan. AI agent orchestration is also becoming more common, where the agent generates the code while the developer supervises the process.

Unit D (QA/Testers) primarily uses GenAI for generating unit tests and automating regression suites. Unit E (Managers) primarily uses GenAI for miscellaneous tasks, but some mentioned using GenAI to engage in deeper technical dialogues in areas where they lack technical expertise.

[1]Boilerplate code is code that is repeated in several places with little to no variation.

[2]Rubber duck debugging is a problem-solving technique where a programmer explains their code line-by-line to an object-like a rubber duck-to force themselves to spot logic errors.

Summary of Specific Use Cases

General use cases, across multiple units:

- Consultation
- Learning and skill development
- Search engine
- Auto-completion
- Boilerplate code generation
- Debugging

Unit-specific use cases:

- Units A (Full-stack) and B (Backend): Evaluating solutions and design trade-offs
- Unit A (Full-stack): Generating development plans based on requirements
- Unit D (QA/Testers): Unit tests and automated regression suites
- Unit E (Managers): Engage in deeper technical dialogues

5.2.3 Practices for Structuring AI Collaboration

There are significant differences in prompting strategies across the organization. It is clear that some developers have used GenAI more extensively and for a longer period than others, making them more knowledgeable about how to utilize it effectively. Some individuals are in such early stages of adoption that they are unaware of prompting strategy, while others have advanced to the point of creating standardized solutions to automate GenAI usage.

Some participants have created concrete frameworks or "recipes" for prompts to generate high-quality output. Observed strategies include:

- One participant utilizes a structured approach:
  1. Describe the goal and what needs to be done.
  2. Provide context.
  3. Describe the pattern and structure for how it should work.
- Another participant uses a similar tactic, but begins by explaining how the feature will be used from the user's perspective before delving into more technical details.
- One participant dictates prompts using voice-to-text, listing knowns, unknowns and uncertainties, while also providing their own suggestions and initial thoughts.
- Several participants assign the model a specific persona or role combined with the context and task.

- A common debugging method involves directly copying and pasting error messages to use them as prompts.
- Those utilizing an agentic coding system typically require the GenAI to create an implementation plan before initiating the coding process. This plan is manually verified.
- A frequent strategy for improving output involves instructing the GenAI to ask clarifying questions if anything is ambiguous.
- Because reviewing AI-generated code can be time-intensive, several developers have adapted their prompting strategies to ease the review process. This is done by breaking down problems into smaller parts and letting the GenAI generate code in segments step-by-step.

The most prominent trend observed is that several developers are shifting away from writing individual prompts in chat interfaces and moving toward utilizing AI agents integrated with systems that automatically provide the necessary context. Rather than manually describing every detail, developers rely on instruction files and Architecture Decision Records (ADRs). This approach utilizes a hierarchy of instruction files dedicated to describing a feature's context and outlining the conventions the resulting code should follow. ADR files are used to document the rationale behind architectural choices to ensure the model does not have to rethink established decisions.

Unit A (Full-stack) is leading this evolution with an approach called *spec-driven development*. In this workflow, developers focus on writing detailed specifications defining the exact behavior, inputs and outputs of a feature in plain text files, and then let AI agents read these files and generate the code necessary to satisfy those requirements.

Furthermore, several participants emphasized that the existing codebase should be well-structured and well-written to ensure high-quality output, as the AI agent relies on the current code as a stylistic and architectural template.

Summary of Practices for Structuring AI Collaboration

Basic prompting strategies:

- Structured frameworks (defining goals, context and patterns)
- Providing the GenAI with known uncertainties and own initial thoughts
- Assigning specific personas or roles to the GenAI
- Step-by-step generation to ease code reviews
- Requiring GenAI to generate implementation plans before coding
- Interactive prompting (instructing the GenAI to ask clarifying questions)
- Direct error message debugging

Advanced prompting strategies:

- Transitioning from chat interfaces to integrated AI agents
- Spec-driven development
- Providing automated context via instruction files and Architecture Decision Records (ADRs)

5.2.4 Current Limitations

When looking at the current limitations of GenAI in software engineering, three main themes can be observed.

Limitation 1: Generating Entire Features

A feature is a distinct unit of functionality in a larger system that solves a specific problem or allows users to complete a task (for example, push notifications in a mobile app). Current GenAI models are sometimes able to generate entire functional features when prompted, but there are often problems included, such as security vulnerabilities, suboptimal performance, or unhandled edge cases. Participants also noted that even if the feature works, the implementation is often poorly structured, necessitating time-consuming refactoring. The larger and more complex the feature, the greater the risk that the implementation fails.

Limitation 2: Context Window Limit

The context window is the total amount of information that a GenAI model can process or remember at any given time. This information is usually text, such as instructions from a developer or the existing code in a repository. If the context window limit is reached (e.g., during a long conversation or when generating a large feature), the model begins to discard prior data from its active memory. This greatly increases the risk of mistakes or hallucinations in the newly generated output.

While the described new AI collaboration strategies presented earlier generally yield better results, new challenges have emerged. One participant mentioned that large instruction files can fill up the GenAI's context window. To address this issue, the participants suggest using so-called "skills" for the GenAI models. Instead of providing the GenAI with every instruction upfront, *skills* act as on-demand capabilities. The model only retrieves and loads these specific files into its working

memory when a task requires them.

The limitation of the context window is a problem for all developers, but especially for those working with embedded systems. This limitation primarily stems from the fact that code in embedded systems is usually closed-source and highly domain-specific due to hardware constraints. When working in other domains, such as frontend development, developers typically use public frameworks and tools on which the GenAI model is already trained. In embedded systems, the context is almost completely unique. This means that for the GenAI to understand an embedded system, large parts of the code repository usually need to be included in the GenAI's context, rapidly consuming the context window. If a model lacks the required context for a task, the output often becomes highly generic, which typically fails to function within an embedded environment.

Limitation 3: Autonomous AI-agents in Production Backend Environments

Every backend developer in this study expressed reluctance to let autonomous AI agents make changes in the back end, such as writing SQL[3] statements in a production database. This reluctance stems from the high potential risks, as data can be difficult to restore. If the agent makes a mistake, critical data could be lost or live services could experience downtime. It remains unclear what it would take for developers to trust GenAI for these kinds of operations. Some even appear opposed to ever using agents in certain environments, arguing that human oversight is strictly necessary. They are, however, highly positive about using GenAI as an assistive tool for writing code, provided they retain full execution control over any changes.

Summary of Current Limitations

1. Generating Entire Features
2. Context Window
3. Autonomous AI-agents in Production Backend Environments

5.2.5 The Future of AI Collaboration

When speculating about the near future, most participants anticipate that AI agents will play an increasingly prominent role in software engineering and some predict that autonomous agents will manage entire workflows. Several participants described scenarios where agents autonomously resolve tickets, implement feedback and create pull requests.

Participants also expect that these agents will become more deeply integrated with databases and APIs through the use of MCP (Model Context Protocol) servers and Swagger files. One participant in Unit A (Full-stack) suggested enabling GenAI

[3]SQL (Structured Query Language) is the standard programming language used to store, manage, and retrieve data within relational databases.

models to pull context directly from Figma designs (UI/UX prototypes) into the codebase. Furthermore, a participant advocates for an organization-wide context repository, accessible to agents via MCP, that includes best practices, architectural patterns, skills, and security standards across all teams. This would automatically equip agents with the relevant context for their current tasks. The participants also mentioned storing project memory, including a complete history of design choices with rationale and alternatives considered, meeting transcripts, and a comprehensive record of the project's evolution. With this, AI agents could become instant experts on any project, with full awareness of historical decisions and context.

Participants in Unit D (QA/Testers) identified future opportunities for smarter debugging and leveraging GenAI to "create fake scenarios that can increase test coverage". This would allow them to focus on more strategic and critical-thinking tasks.

Summary of The Future of AI Collaboration

Most participants believe AI agents will play an increasingly prominent role.

Some participants also believe that these AI agents will become more deeply integrated with databases, APIs and other software programs. One participant advocates for an organization-wide context repository to make agents more effective.

5.3 Organizational and Managerial Challenges and Opportunities (RQ3)

This section addresses the study's third research question: "*What organizational and managerial challenges and opportunities arise when GenAI is introduced into existing development workflows?*"

The overall insights indicate that the introduction of GenAI is rarely purely a technical issue, but rather a more complex organizational transformation. This leads to a change in the leadership role from steering processes to enabling learning. The managerial challenges are presented as main themes where the evaluation of performance and potential adaptation are covered. Then, the primary results of organizational challenges and the mapping of current bottlenecks are presented. Finally, it describes the strategic and organizational opportunities for the future identified in the interviews.

5.3.1 Managerial Strategies: Performance and Skill Sharing

New requirements emerge on how organizations measure performance and manage skill development when GenAI is introduced to existing workflows. The results of this study highlight two main themes in management strategies: a delay in the adaptation of metrics and a shift towards an individual-driven learning culture.

A managerial challenge that emerges in the study is the current lag in performance evaluation. All the managers and team leaders report that they have yet to update their KPIs (Key Performance Indicators) or estimation models (e.g., story points). Development work is therefore measured in the same way as before the implementation of GenAI tools. The interviews reveal a clear divide in the opinions on whether these metrics should be revised, often characterized by unclear justifications as to why change is or is not necessary because of the integration of GenAI.

Some participants argue that existing metrics should be maintained. Their reasoning is based on the idea that performance is fundamentally about delivering customer value, regardless of the underlying tools the developers are using to achieve this. However, some other participants state that current measurement methods are rapidly becoming obsolete as efficiency increases. One manager even predicts that GenAI will be at the center of the development process within six months. Although the rest disagree, another manager points to a necessary paradigm shift for leadership, where organizations must move away from traditional quantitative metrics (e.g., lines of code or time spent) and instead move further towards measuring actual outcomes and value created.

There have also been changes in how the organization handles competence development around GenAI. Another finding is the lack of more formal, top-down training efforts such as traditional courses. Instead, knowledge accumulation is characterized by a bottom-up approach, which is almost exclusively driven by the developers' own curiosity combined with collegial learning (e.g., peer-to-peer). Several managers admit that a significant responsibility for learning new tools is thus placed on the individual, something that in some cases is even expected to happen during the employees' free time. As a consequence of this learning culture, the manager's role shifts to acting more like an enabler where leadership focuses on creating time for exploration. Examples of some initiatives include "tribe days", where teams can build and evaluate GenAI demos, and the establishment of internal GenAI discussion forums for knowledge exchange. A growing opportunity has also been identified in using GenAI tools themselves as pedagogical support to facilitate this individual-driven learning process.

Summary of Managerial Strategies: Performance and Skill Sharing

1. Delay in the Adaptation of Metrics
2. Shift towards an Individual-driven Learning Culture

5.3.2 Organizational Challenges and Bottlenecks

Challenge 1: Knowledge loss and the deskilling problem

Managers and developers express strong concerns that GenAI creates a dependency that risks ruining basic professional coding skills. The fear is described not only as a concern about GenAI making developers redundant or losing their jobs, but also

as a risk that GenAI usage makes developers lazier, dependent on GenAI, and less competent at programming. Developers warn of vibe coding and "semi-knowledge", where code is implemented without full understanding. Developers appreciate that GenAI gives them answers quickly, but emphasize that they do not learn from the answers or care enough to do so, thus quickly becoming dependent on the GenAI's responses.

A macroeconomic concern from a management perspective is also raised about how organizations in the future will be able to recruit senior developers, especially if today's juniors are trained as "AI programmers" or if the junior developer role is diminishing. Managers also describe seniors having a more difficult time keeping up with GenAI development and that their syntax or specific language knowledge is in some way losing meaning and becoming obsolete. This dynamic therefore is believed to be a potential threat to the depth of knowledge of the profession in the future.

Challenge 2: Accountability and technical debt

When asked about who has the responsibility for the AI-generated code, all developer units (ABCD) agree. The developer using GenAI has to take the accountability for the output it generates. The increased volume of AI-generated code creates an enhanced demand for verification. Several participants raise discussions about code ownership as GenAI is neither deterministic nor does it have an understanding of the organization's specific system architecture. Unit D (QA/Testers) highlights another problem with AI-generated code and compares it to sourcing code from Stack Overflow. Unlike Stack Overflow, GenAI creates novel and unverified code. Developers are now expected to comprehend this implementation from scratch, which leads to more review work. Uncertainties about code quality and potential security vulnerabilities are also reported. Another developer describes that GenAI can get around 90% right, but the last few percent can create a lot of problems, so it is not enough. It is, therefore, argued that GenAI cannot currently be trusted with more autonomous responsibility, as unexpected code can quickly increase technical debt.

Challenge 3: Limitations in context windows

GenAI tools are reported to have difficulty handling large existing systems, especially for Unit C (Embedded). When context windows run out, the tools tend to hallucinate and lose reliability. Embedded systems have closed source code, and it is explained that the GenAI tools' training data is likely lacking, causing these problems. Units A (Full-stack) and B (Backend) are also highlighting this problem.

Challenge 4: The knowledge gap

A noticeable gap is emerging between development teams and the management side. As management representatives realize the potential of GenAI, they often get higher expectations for faster and cheaper delivery without a deeper understanding of the underlying technical risks. Managers are engaging in more technical discussions without having the broader knowledge that developers have. This transition phase is also characterized by an uneven rollout between different teams and departments, in

addition to a lack of common guidelines or a clear framework on how to use GenAI. Units A (Full-stack) and B (Backend) are missing a "Shared Manifesto".

Challenge 5: Data integrity, costs, and standardization

Data integrity and the fear of source code leaks are pervasive concerns across units. There are also concerns raised by managers due to the high cost of licenses and tokens for GenAI usage. Overuse of GenAI is reported by Units A (Full-stack) and B (Backend) to risk standardizing code or UI to the point that all software design becomes generic and creative freedom is stifled.

Bottleneck 1: Requirements Engineering, *from how to what*

As GenAI streamlines the actual coding, the surprising finding of this study is that the main bottleneck of the system is described as shifting towards requirements engineering and business analysis. The biggest obstacle, confirmed by both managers and developers, is no longer building the solution, but understanding the customer need and bridging the gap between product owners and development teams. The architecture and planning phase is therefore reported to be the most important step in the development process.

Bottleneck 2: Legacy-friction

The speed of GenAI adoption is slowed by existing technical debt and legacy systems. Developers report constantly having to divide time and attention between maintaining old code, inadequate documentation and a fear that new migrations will break existing dependencies. Many integration barriers are also rooted in administration rather than the technology itself. Even if security restrictions are set, there are still licensing issues creating a bottleneck for the developers. Simultaneously, the pressure to deliver means that developers are paradoxically not given enough time to learn how to optimize their usage of GenAI tools. Scaling up faster is therefore explained to be hindered by the company's legacy structure.

Bottleneck 3: Cognitive overload

Some participants described the human factor as also being a limitation in the overall throughput of the system. Humans lack the mental bandwidth to continuously consume and review the massive amount of output that GenAI tools generate. Developers describe that there is an inability to constantly be in a productive state of flow, which sets a natural limit to the speed that GenAI code can be verified and implemented.

Summary of Organizational Challenges and Bottlenecks

Organizational challenges:

1. Knowledge Loss and The Deskilling Problem
2. Accountability and Technical Debt
3. Limitations in Context Windows
4. The Knowledge Gap
5. Data Integrity, Costs, and Standardization

Organizational bottlenecks:

1. Requirements Engineering, *from How to What*
2. Legacy-friction
3. Cognitive Overload

5.3.3 Future Opportunities

This study has also collected a number of significant opportunities associated with GenAI. These can be divided into two broad categories: organizational gains that act as a catalyst for macro change and managerial strategies to realize these gains.

Org. opportunity 1: Cognitive offloading and the new digital colleague

GenAI is increasingly moving from being seen as a passive software tool to an active colleague or collaborator. Developers report that GenAI can reduce their cognitive load by taking over simpler and repetitive tasks. This allows them to focus more on complex and creative problems, as it frees up valuable time. One participant explains that this puts the human at the center instead of the machine, as human judgment becomes the primary driving force.

Org. opportunity 2: Cross-functional communication and innovation

GenAI is described as lowering the threshold for experimentation and creating mockups. Developers report that colleagues without a traditional coding background can perform more technical tasks and, for example, create prototypes which reduces innovation cycles. GenAI can therefore also act as a bridge that facilitates cross-functional communication. It is becoming easier to try out new platforms and technologies with less time invested. This is also emphasized by managers, who note that the result is a more agile organization that is less dependent on isolated technical silos.

Org. opportunity 3: Realizing the low-code promise

There is a strong belief among developers that GenAI will finally fulfill the long-held industry vision of *low-code development*. Developers are likely to be able to build larger, more advanced solutions at a lower cost when streamlining code creation. Managers predict that the ability to do more in the same amount of time will lead to a higher ROI and a general productivity boost of the entire development cycle.

Managerial opportunity 1: Model selection

A potential strategy identified by some of the developers is the importance of using the right model for the right thing. Managers and more experienced developers using GenAI are moving away from a monolithic view of the tool. Instead, they propose using more advanced and expensive models (e.g., Opus 4.6) for complex architecture and planning, along with cheaper and faster models for repetitive code generation and unit testing. They expect that by using this strategy, they will be able to optimize both cost and performance of the tools.

Managerial opportunity 2: Proactive investments in learning

A common stance across all units is that successful GenAI implementation does not happen automatically. Managers acknowledge that dedicated time and conscious investments are required to evaluate the tools and drive skill development. To realize the full potential of GenAI, developers are expecting managers to act proactively and create space for learning, experimentation, and even failure.

Managerial opportunity 3: Internal knowledge infrastructure (RAG and context repositories)

Without the right context, GenAI tools will remain insufficient. An important request highlighted by the developers is to build internal context repositories and vector databases (RAG) that contain the organization's domain knowledge, documentation and history of decisions. Changes in infrastructure are also seen as especially valuable within Unit C (Embedded) for managing and modernizing the resource-intensive legacy systems. Aside from providing a better understanding of older systems by connecting GenAI to the right internal context, further opportunities were also suggested by developers about automating the generation of technical documentation and reports.

Summary of Future Opportunities

Organizational opportunities:

1. Cognitive Offloading and the New Digital Colleague
2. Cross-functional Communication and Innovation
3. Realizing the Low-code Promise

Managerial opportunities:

1. Model Selection
2. Proactive Investments in Learning
3. Internal Knowledge Infrastructure (RAG and Context Repositories)

6

Discussion

This chapter discusses the findings of the study in relation to the research questions and previous related work. The chapter begins by describing how GenAI is utilized within development workflows and its resulting consequences. It then examines the transformation of the developer role in combination with shifting competencies. Following this, the discussion addresses the growing concern of GenAI overuse and the diminishing role of junior developers. The chapter also analyzes the disparities in GenAI adoption across the different units. Based on these insights, a set of recommendations is presented, rooted in fostering an agile and knowledge-sharing culture. Finally, the chapter concludes by addressing threats to the study's validity.

6.1 The Expanding Developer Role

The integration of GenAI into existing software development workflows not only involves a change in competencies around AI literacy and usage, but also drives a fundamental transformation of the entire developer role. The results from this study not only confirm and build upon the usage patterns previously identified in related works, but also show a shift in where value is created when using GenAI in the development process.

6.1.1 Acceleration and Exploration Mode

Similar to previous research on AI-assisted programming (see Section 3.1), the study's results regarding developers' use cases of GenAI (see Section 5.2.2) can also be divided into acceleration and exploration modes. Acceleration mode is the most prominent across all units and is applied to speed up tasks such as writing boilerplate code, generating unit tests, autocompleting code, and performing basic debugging. This is in line with findings from Pereira et al.'s [31] case study of a Brazilian company described in Section 3.5, where repetitive tasks were the primary use case for GenAI.

Exploration mode, on the other hand, is used when the solution is not clear or given in advance, and developers report brainstorming ideas and evaluating design trade-offs as they would with a senior colleague. This study has identified some frequent GenAI use cases that go beyond direct code generation, but they can still be categorized as a consequence of this so-called exploration mode. With the main consequence being

the continuous learning enabled by GenAI, developers are experiencing continuous skill development integrated into their daily workflow. All participants use GenAI as a search engine and are actively engaging in deeper technical discussions thereafter. This means almost instant access to even more information about a topic via a single prompt. Whether this information is actually retained, however, is another question.

6.1.2 Cognitive Offloading Affects Productivity

The frequent use of GenAI in acceleration mode is proven to lead to significant perceived time savings at the individual level for developers. They become more productive in the sense that simpler tasks are completed faster. Tracing the time savings upstream raises the question of why the entire development process does not also accelerate proportionally. The saved time appears to be absorbed somewhere else.

The results indicate that this time gain rarely translates into a proportional increase in the number of tasks or features being delivered. Instead, indications show that the baseline for what is considered good-enough quality and acceptable test coverage has been substantially raised, along with overall expectations. A significant effect of the time saved is instead found on the psychological level through cognitive offloading. By delegating syntax and repetitive tasks to GenAI, the mental friction during task switching is reduced. Even when GenAI cannot solve the entire problem itself, it can help facilitate planning, which leads to reduced stress and therefore higher job satisfaction. This confirms the relevance of the SPACE framework (see Section 3.4), where the developer experience is closely linked to overall perceived productivity. The developers feel that having GenAI as an assistant provides them with some form of mental relief.

The net effect on overall cognitive load, however, remains ambiguous among the participants. GenAI can help initiate tasks, overcome blockers, and ease context switching, but is also described as introducing new cognitive responsibilities such as keeping track of many ongoing tasks at once. These opposing effects do not necessarily exclude or overlap with one another, leading to even more divided opinions on whether the mental load has decreased or shifted.

At the same time, this newfound efficiency carries the risk of overproduction. Where resource constraints have historically demanded strict prioritization, for example, choosing one out of ten projects, GenAI now enables more projects to be developed. However, there is a consequence of maximizing customer deliveries with new features or products. The software can become unnecessarily complex and unmanageable over time, building up technical debt more quickly. An unbalanced expansion can risk feature bloat, where the core value of a service or system becomes diluted by all the excessive additions. Organizations may also face strategic challenges from the customer side, needing to have the discipline to say no, even when the short-term technical cost of saying yes to new features has dropped drastically because of GenAI.

This discrepancy also highlights a connection to how performance is measured.

Despite the nature of work changing, participants state that KPIs remain the same. Opinions differ on whether this is a problem, but both sides agree that the real goal is to create customer value. If GenAI accelerates technical delivery, it becomes increasingly paradoxical to measure developer performance in code deliveries instead of the actual value created. This points towards an idea of measuring performance based on outcome and value created, instead of production-based output. Future KPIs will likely also need to prioritize long-term value creation. However, evaluating new workflows or achievements retrospectively can be difficult.

6.1.3 The Developer Role Redefined

The main consequence for the developer role based on the automation of the coding phase is how it expands both upstream and downstream. The responses describe a blurring of the traditional professional roles between developers, product owners (POs), and project managers (PMs). When GenAI assists the implementation phase (the *how*), the developer's primary challenge shifts to requirements (the *what*). This manifests in the emerging AI collaboration method called *spec-driven development*. The middleman disappears and the developers become responsible for breaking down requirements into unambiguous and detailed instructions to the GenAI tools. This may also be connected to the assumption that GenAI tools need very specific instructions to behave as intended, supported by the findings of another study in Section 3.3. This working methodology places new demands on the developer's metacognitive ability. To effectively control the GenAI tools, developers now have to reflect on their own problem-solving processes.

The blurring of boundaries does not necessarily imply that the PO or PM roles will be entirely absorbed by developers. Rather, it presents a need to also re-evaluate adjacent roles that are likely to be affected by the integration of GenAI. The traditional PO, PM, and managerial roles may evolve to address the new organizational needs or even give rise to new roles. Consequently, it becomes important for organizations to redefine role boundaries and establish who holds accountability for specific tasks during the shift to a more AI-integrated workflow.

This finding represents an important contribution to existing research. While Pereira et al.'s [31] case study of a Brazilian company (see Section 3.5) found that the use of GenAI in non-coding tasks, such as requirements and architecture, was severely limited, this study shows that development has progressed rapidly. The real bottlenecks right now are no longer just the technical limitations, but the human ability to comprehend the system as a whole, specify intent and review results. AI transformation is moving extremely fast. Especially regarding technical limitations, the identified key points worth focusing on are becoming more connected to humans. Humans are, therefore, paradoxically, one of the main factors that will affect new AI collaboration.

6.1.4 The Shift in Competence

As the professional role broadens, there is also a shift in which competencies are valued the most. Previous studies (see Section 3.2) highlight general abilities such as AI literacy, critical thinking, problem-solving and creativity. This case study's results confirm and add nuance to this by showing how these skills manifest in practice. Critical thinking skills are explicitly mentioned by all developer units as essential for identifying logical errors and hallucinations from the GenAI output. In addition, it appears that the new competency requirements operate on several interconnected levels. On the individual level, there appears to be a high degree of personal responsibility, driven by curiosity, continuous learning, and the ability to break down and plan one's own workflow. On the structural level, the focus shifts from isolated technical skills to systems understanding. Developers must now increasingly understand the underlying need, purpose, and intent even outside their own directly assigned tasks. The evaluation of skills required for their role is therefore less about isolated technical ability and more about how well the individual can interact with the system, colleagues and GenAI. Developers who already possess strong soft skills, such as adaptability, effective communication and systems thinking, will therefore navigate this transition more easily. Interestingly, "creativity" was not mentioned by the participants as an increasingly important characteristic, despite participants describing concerns that GenAI tools risk making code and design more homogeneous.

This shift towards softer skills and a more holistic understanding adds a new dimension to the inherent conflict in how skills development is managed. If deep intuition, communication skills and systems understanding are becoming essential skills for the redefined developer role, they cannot be learned in isolation in front of a screen. This type of contextual experience and tacit knowledge is usually built through peer interactions such as mentoring, code reviews, and engagement in complex learning environments. If the learning responsibility is completely shifted to the individual, organizations risk overlooking the fact that AI experience will require structured time and shared forums. These are necessary not only for discussing technical aspects of GenAI, but also for encouraging collaborative practices and a broader curiosity for learning. This also highlights a vulnerability that lays the foundation for one of the biggest risks of GenAI integration: the difficulty of cultivating future deep skills.

6.2 The Deskilling Problem

Knowledge loss is a growing problem in the software development field. As GenAI advances and assumes responsibility for more technical tasks, software engineers risk becoming completely dependent on these tools and losing skills that were previously essential to their roles. While there is a clear consensus on the importance of understanding implementations, developers may easily become complacent, neglecting to analyze GenAI output provided that the software appears functional. When encountering a bug, they might rely on iterative prompting (especially since developers struggle to debug GenAI code, see Section 3.3) to reach an apparent solution without

fully comprehending the underlying issue or the mechanics of the model's generated fix. This approach to programming, commonly known as vibe coding, can result in degraded code quality, inefficient implementations and security vulnerabilities. Furthermore, it risks creating future complications that are increasingly difficult and costly to resolve.

Prior to the widespread adoption of GenAI, developers who encountered a problem were required to understand and resolve it themselves. Although they could seek assistance from forums such as Stack Overflow or consult colleagues, this process still demanded significantly more cognitive engagement; the issue had to be explicitly formulated and discussed, and the proposed solution had to be carefully reviewed. This collaborative effort also led to broader dissemination of knowledge in the field of software engineering, since platforms like Stack Overflow are utilized by a global community of diverse developers. Today, in contrast, an AI agent can often resolve these issues autonomously. While interacting with these models presents a valuable learning opportunity, such as prompting the GenAI for explanations and engaging in insightful conversations, many developers disregard this due to time constraints, financial cost (as the better models do not allow free unlimited usage), or plain laziness. Some developers may not even feel a strong incentive to deepen their programming expertise, as they anticipate that GenAI will fully automate these tasks in the future anyway. The responsibility for learning is often placed entirely on the individual, typically requiring unpaid time despite creating organizational value. When organizations fail to integrate upskilling into a structured internal approach, it further disincentivizes developers from educating themselves and exacerbates the issue.

6.2.1 The Future of Junior Developers

While these challenges affect developers of all levels, they are particularly significant for juniors as advanced skills in software engineering are heavily contingent upon experience. If junior developers rely on GenAI to perform most of the basic tasks, they may fail to acquire the necessary expertise required for senior roles. Consequently, organizations risk a loss of deep domain knowledge. This issue is exacerbated by the trend of junior positions being the first to be eliminated (as GenAI can often perform their tasks) when, as anticipated, teams become smaller (see Section 5.1.1). This reduces the pipeline of developers advancing to senior roles and risks resulting in a significant shortage of developers possessing deep technical expertise in the future. If no measures are implemented, organizations may be forced to contend with a generation of mostly "AI programmers" who are unable to resolve complex problems or maintain legacy systems that exceed the capabilities of GenAI. The problem of potential workforce polarization has been observed in a 2025 study by Dijana Bakajac [29] (see Section 3.2).

6.3 Domain Disparities

The level of expertise regarding GenAI varies noticeably across different organizational units and within the units themselves. Several individuals have developed their own AI skills and more mature prompting strategies (see Section 5.2.3) to utilize GenAI effectively. However, these strategies are not consolidated in a shared repository. The absence of a shared manifesto and organization-wide knowledge-sharing practices for strategies further exacerbates the technical obstacles. This also creates disparities in how well the tools are adopted across the organization, with Unit C (Embedded) demonstrating lower levels of integration.

Beyond individual expertise, there are other differences in the specific use cases of GenAI between the domains. The results reveal a clear theme: developers in Unit C (Embedded) do not utilize GenAI to the same extent, nor do they view it as reliable and helpful, compared to developers in other domains. Interestingly, despite the lack of trust and lower usage overall, they still report that GenAI saves them time. This can be explained by the type of tasks they delegate to GenAI. While they do not rely on it for complex logic where high reliability is necessary, they still benefit from automating some simpler tasks. When they do use GenAI tools, Unit C (Embedded) primarily uses them for code generation (see Section 5.2.2) and much less for architectural planning, unlike Unit A (Full-stack) or B (Backend). This suggests that GenAI is currently less effective for evaluating design trade-offs or planning in embedded environments.

This discrepancy appears to stem from the additional constraints inherent to embedded systems. The first constraint is performance. While all systems should perform well, embedded systems require strict optimization, as the processors in these systems are typically less powerful than those found in servers or desktop computers. Although GenAI has been improving in this regard, the generated implementations are often suboptimal and unnecessarily complex, which is not acceptable for these environments.

The second possible constraint is context dependency. Hardware requirements often require highly specific implementations, unlike other systems that frequently utilize common frameworks and rely on open-source libraries. As GenAI is trained on vast amounts of open-source data, it has naturally become proficient at generating code for these common frameworks. Conversely, embedded systems are highly distinct from product to product and are often closed-source. Consequently, GenAI models must retain large portions of massive code repositories and other system-specific information within their context windows to accurately map the system and efficiently generate new code. This poses a challenge, as GenAI models have limited context window capacities. Once this limit is reached, the GenAI output becomes more generic, which is problematic for context-dependent systems, rendering the context window limit a severe bottleneck. While this will likely become less of a problem in the future as newer models can handle larger contexts, embedded system development will likely continue to lag behind other domains due to its inherent context dependency.

Although embedded development is the most severely affected by context limitations, it remains an issue in other domains when dealing with large systems and complex tasks, such as generating entire features. To mitigate hallucinations, it is crucial that the context is contained and that large features are broken down into smaller components so the assigned task remains small.

6.4 Recommendations

Establishing best practices for a domain that evolves daily presents a unique challenge. A fixed set of rules or static guidelines is almost guaranteed to become rapidly obsolete. Therefore, the following recommendations will be rooted in and focus on further fostering an agile and knowledge-sharing culture and workflows.

Recommendation 1: Taking organizational responsibility for GenAI upskilling

Organizations can no longer rely on developers driven by individual curiosity to learn GenAI tools in their spare time. As discussed earlier, core competencies are shifting toward soft skills like metacognition and systems understanding. Learning these skills and acquiring tacit knowledge requires social interaction, code reviews and peer discussions. Therefore, management must proactively create collaborative forums and opportunities for developers to explore these tools and learn from each other.

Recommendation 2: Investing in talent through junior apprenticeship

As developers and organizations increasingly rely on GenAI to manage entry-level tasks, the incentive for organizations to hire junior developers is decreasing. Furthermore, those who are hired often utilize GenAI extensively, which risks making them over-reliant on these tools and hinders the accumulation of the technical expertise and system knowledge necessary to advance to senior roles. These factors risk leading to a loss of deep domain knowledge within the organization (see Section 6.2.1).

To mitigate this, it is important that organizations do not cease hiring junior developers. Juniors cannot be expected to compete with GenAI in delivering immediate value, as GenAI will, in most cases, be more time- and cost-efficient at completing basic tasks. If producing short-term value remains the highest priority, it will be difficult for organizations to justify hiring entry-level talent. The view of junior roles must therefore evolve to accommodate this new reality and shift from being immediate value creators to being long-term investments in knowledge retention and future productivity. Even if GenAI can automate many of their tasks, junior developers will, over time, learn the internal systems, cultivate their technical expertise, and gain tacit knowledge.

Furthermore, organizations must take a more active role in professional development and place a greater emphasis on apprenticeships rather than value creation, as junior developers cannot be expected to independently acquire the necessary skills and knowledge given the reliance on GenAI.

This paradigm shift introduces new challenges that organizations must address. These include determining how to maximize employee retention, identifying which tasks to automate, and managing the short-term costs of hiring and training, as junior developers are unlikely to produce much value at the beginning of their employment.

Even with this revised strategy, GenAI will undoubtedly continue to assume some entry-level tasks, which will likely result in fewer junior developers being employed overall.

Recommendation 3: Unifying GenAI strategies and disseminating knowledge

As there are clear disparities in AI expertise across the organization, both within and between technical domains (units), there must be a strong focus on knowledge dissemination. Knowledge regarding GenAI usage should be shared throughout the organization in a structured manner and not left to the individual developers themselves.

Furthermore, unified GenAI strategies should be implemented. These strategies should include a shared framework for the effective and efficient use of GenAI, detailing prompting strategies, task-specific workflows, what to automate, and other relevant standards. A standardized suite of tools (e.g., Copilot or Claude) should be made available to developers, accompanied by clear guidelines on their application. These guidelines should also specify when to use which model to ensure cost-efficiency, as some are unnecessarily expensive for simple tasks.

A context database containing architectural patterns, AI skills, and other resources, similar to the one described in Section 5.2.5, along with other integrations for AI agents, should be considered. However, these advanced integrations should only be implemented after the other measures.

Summary of Recommendations

1. Taking organizational responsibility for AI upskilling
2. Investing in talent through junior apprenticeship
3. Unifying AI strategies and disseminating knowledge

6.5 Threats to Validity

This section addresses mitigation strategies to account for threats to validity, minimize risks, and increase the robustness of the results.

6.5.1 Threats to External Validity

A potential threat to the external validity of the study was the limited sample size, as the study was conducted within a single organization in combination with its particular field. The data collected might therefore not be fully representative of

the software population. However, the goal of the study was transferability, rather than statistical generalizability. Therefore, a large sample size was not necessary, since the focus was to gather conceptual insights that allowed for transferability to similar contexts. To strengthen the breadth of the study, the organization was divided into multiple data units to enable data to be collected for different types of roles. Through this approach, the study aimed to cover larger parts of the software development process and reduce the risk of skewed results if all participants were working towards the same product or deliverable. The unit and participant selection (as opposed to volunteer sampling) further mitigated the risk of selection bias where only AI enthusiasts participated, ensuring a more realistic representation of both frequent users and skeptics.

It is also important to note that the study primarily captured the earlier and middle phases of development. Later phases, specifically deployment and monitoring, were not featured in the data collection or analysis. Consequently, the findings regarding GenAI usage may not be transferable to DevOps engineers or operational roles.

The organization's internal policies surrounding the use of GenAI tools might also have affected how participants used GenAI and their responses. To reduce this impact, participant anonymity was ensured and clearly communicated to participants throughout the study. The organization may also have had preferred tools with their own limitations, indirectly affecting user options.

The rapid technological development in the GenAI field also posed a threat, as both tools and user patterns changed regularly. This was addressed by focusing on behaviors, workflow processes and collaboration patterns, rather than the performance of specific tools and their productivity measurements, which were more time-sensitive.

6.5.2 Threats to Internal Validity

Internal validity concerns the ability to draw causal conclusions and ensure that observed factors are not the result of unknown influences. Studies conducted in realistic settings, like this study, often face challenges with confounding variables that cannot be strictly controlled, such as the participants' programming experience or task complexity. There was a risk that observed changes in workflow or productivity were influenced by factors other than GenAI adoption, such as organizational restructuring or project-specific reasons [42]. This was mainly mitigated by data source triangulation. Beyond triangulation, the data analysis also included negative case analysis. By systematically searching for evidence that contradicted the emerging themes, the study ensured that conclusions were robust and not the result of confirmation bias [40].

6.5.3 Threats to Construct Validity

Since this study relied on interviews, there is a risk that the ambiguity of terms and subjective interpretation of words differed between participants and the researchers. This could make it difficult to confirm if the study actually measured what it was

supposed to be measuring. To mitigate this and ensure that the measures accurately reflected the study's purpose, a systematic mapping between the interview questions and the research questions was conducted (see Figure 4.1). This mapping ensured that all core components of the research questions were comprehensively covered. Another way to mitigate this threat was also through data source triangulation. If the same conclusion could be drawn from several sources in different roles, the conclusion was stronger and less likely to be biased by the interpretation of a single source [40]. This threat was also mitigated by using observer triangulation. By having both researchers actively participate in the interviews and review the analysis, the bias from the researchers' side was mitigated [40].

There was still a reliance on self-reported data. Without direct observation, the study captured the participants' perception of their workflow rather than the objective reality. This created a risk of subjective bias that had to be mitigated. The SPACE framework was used to address the perception-reality gap [36]. The first phase of the study also involved identifying additional resources to refine the interview guide, further ensuring that the interview questions elicited concrete, useful responses.

7

Conclusion

This case study provides a holistic view of how GenAI is reshaping developer roles, collaborative workflows, and organizational structures within a large manufacturing company.

The findings reveal that the integration of GenAI fundamentally redefines the developers' identities, shifting their primary roles from manual "coders" to "code reviewers" and "software architects". Because GenAI automates much of the repetitive coding, developers are moving "upstream" in the SDLC, taking on responsibilities traditionally held by product owners and project managers. Consequently, the skills most valued in the industry are shifting; syntax memorization is becoming less important, while systems thinking, the ability to define unambiguous requirements, and critical review skills are becoming paramount. Furthermore, as software engineering becomes less isolated, communication skills are becoming increasingly necessary.

In terms of collaboration, distinct disparities exist. While full-stack and backend developers are exploring advanced, spec-driven development with AI agents, embedded systems development lags behind due to hardware constraints and strict context window limitations. Across all units, developers remain reluctant to trust autonomous AI agents without strict human oversight.

At the organizational level, GenAI integration is a complex transformation rather than a simple technical upgrade. While GenAI provides significant perceived time savings and some cognitive offloading, this efficiency rarely translates into higher delivery rates. Instead, the saved time is reinvested into raising the baseline for quality. Furthermore, as the speed of code generation increases, the primary bottleneck in software engineering has shifted from how to build a solution to what to build. GenAI also poses a severe "deskilling" threat. As efficiency gains push organizations toward smaller development teams, particular concerns are raised regarding the diminished role and future pipeline of junior developers.

To realize the true potential of GenAI, organizations must move beyond treating it as a passive tool and integrate it as a collaborative partner. This requires moving away from an individual-driven learning culture and taking proactive organizational responsibility for upskilling, unifying AI strategies, and protecting the apprenticeship of junior developers to ensure the long-term retention of deep domain knowledge.

As the software development landscape continues to evolve, future research should focus on long-term studies tracking the actual ROI of GenAI in real-world contexts and on exploring pedagogical frameworks for training the next generation of software engineers in this new environment.

Bibliography

[1] M. Chui *et al.*, "The economic potential of generative AI: The next productivity frontier," McKinsey & Company, New York, NY, USA, Tech. Rep., Jun. 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier

[2] Stack Overflow, "2025 Developer Survey," Stack Overflow, New York, NY, USA, Jul. 2025. [Online]. Available: https://survey.stackoverflow.co/2025

[3] K.-J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. Art. no. 11, pp. 1–51, Sep. 2018.

[4] DORA and Google Cloud, "State of AI-assisted software development," Google Cloud, Mountain View, CA, USA, Tech. Rep., 2025. [Online]. Available: https://services.google.com/fh/files/misc/2025_state_of_ai_assisted_software_development.pdf

[5] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto, "Beyond code generation: An observational study of ChatGPT usage in software engineering practice," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. Art. no. 16, pp. 1819–1840, Jul. 2024.

[6] K. J. Klein and S. W. J. Kozlowski, Eds., *Multilevel Theory, Research, and Methods in Organizations: Foundations, Extensions, and New Directions.* San Francisco, CA, USA: Jossey-Bass, 2000.

[7] L. Banh and G. Strobel, "Generative artificial intelligence," *Electronic Markets*, vol. 33, no. 1, p. 63, 2023.

[8] OpenAI. (2025, Dec.) Introducing GPT-5.2. Accessed: Feb. 5, 2026. [Online]. Available: https://openai.com/index/introducing-gpt-5-2/

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Long Beach, CA, USA: Curran Associates, Inc., 2017, pp. 5998–6008.

[10] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, San Francisco, CA, USA, Tech. Rep., 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[11] N. Friedman. (2021, Jun.) Introducing GitHub Copilot: your AI pair programmer. GitHub. Accessed: Feb. 5, 2026. [Online]. Available: https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[13] OpenAI. (2022, Nov.) ChatGPT: Optimizing language models for dialogue. Accessed: Feb. 5, 2026. [Online]. Available: https://openai.com/index/chatgpt/

[14] Anysphere. (2023) Cursor. Accessed: Feb. 5, 2026. [Online]. Available: https://www.cursor.com

[15] Capgemini Research Institute, "Harnessing the value of generative ai: 2nd edition," Jul. 2024, accessed: Feb. 2026. [Online]. Available: https://www.capgemini.com/insights/research-library/generative-ai-in-organizations-2024

[16] J. C. R. Licklider, "Man-computer symbiosis," *IRE Transactions on Human Factors in Electronics*, vol. HFE-1, no. 1, pp. 4–11, 1960.

[17] M. A. Hearst, J. F. Allen, E. Horvitz, and C. I. Guinn, "Mixed-initiative interaction," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 5, pp. 14–23, 1999.

[18] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen *et al.*, "Guidelines for human-ai interaction," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[19] J. White *et al.*, "A prompt pattern catalog to enhance prompt engineering with ChatGPT," arXiv preprint arXiv:2302.11382, Feb. 2023. [Online]. Available: https://arxiv.org/abs/2302.11382

[20] L. Tankelevitch, V. Kewenig, A. Simkute, A. E. Scott, A. Sarkar, A. Sellen, and S. Rintel, "The metacognitive demands and opportunities of generative ai," in *Proc. of the 2024 CHI Conference on Human Factors in Computing Systems*, Honolulu, HI, USA, May 2024, pp. 1–24.

[21] N. Hinniger-Foray, G. Mognier, and A. Charikova. (2025, Oct.) Methodology: How we discovered over 2k high-impact vulnerabilities in apps built with vibe coding platforms. Escape. Accessed: Feb. 6, 2026. [Online]. Avail-

able: https://escape.tech/blog/methodology-how-we-discovered-vulnerabilities-apps-built-with-vibe-coding/

[22] J. Butler *et al.*, "Microsoft new future of work report 2023," Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-2023-34, 2023. [Online]. Available: https://aka.ms/nfw2023

[23] C. Bird, D. Ford, T. Zimmermann, and N. Forsgren, "Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools," *ACM Queue*, vol. 20, no. 6, pp. 35–57, Dec. 2022.

[24] S. Barke, M. B. James, and N. Polikarpova, "Grounded Copilot: How programmers interact with code-generating models," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. Art. no. 85, pp. 1–27, Apr. 2023.

[25] D. Long and B. Magerko, "What is AI literacy? competencies and design considerations," in *Proc. CHI Conf. Hum. Factors Comput. Syst. (CHI)*, Honolulu, HI, USA, Apr. 2020, pp. Art. no. 599, pp. 1–16.

[26] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Proc. CHI Conf. Hum. Factors Comput. Syst. (CHI)*, New Orleans, LA, USA, Apr. 2022, pp. Art. no. 332, pp. 1–22.

[27] H. Lee, Y. Ding, S. Octora, and A. Isomura, "The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers," in *Proc. CHI Conf. Hum. Factors Comput. Syst. (CHI)*, Yokohama, Japan, Apr. 2025, p. Art. no. 1121.

[28] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, "Who answers it better? An in-depth analysis of ChatGPT and Stack Overflow answers to software engineering questions," *Empir. Softw. Eng.*, vol. 29, no. 2, p. Art. no. 32, Mar. 2024.

[29] D. Bakajac, "The impact of ai on skill development and career progression in software engineering," Master's Thesis, TU Wien, 2025. [Online]. Available: https://repositum.tuwien.at/bitstream/20.500.12708/224146/1/Bakajac%20Dijana%20-%202025%20-%20The%20Impact%20of%20AI%20on%20Skill%20Development%20and%20Career...pdf

[30] S. Overflow, "2024 developer survey," 2024, accessed: 2026-02-07. [Online]. Available: https://survey.stackoverflow.co/2024/ai

[31] G. V. Pereira, V. Jackson, R. Prikladnicki, A. van der Hoek, L. Fortes, C. Araújo, A. Coelho, L. Chelli, and D. Ramos, "Exploring GenAI in software development: Insights from a case study in a large Brazilian company," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025, pp. 330–341.

[32] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*. Abingdon, U.K.: Routledge, 2016.

[33] W. Harding and M. Kloster, "Coding on Copilot: 2023 data shows downward pressure on code quality," GitClear, Seattle, WA, USA, White Paper, Jan. 2024. [Online]. Available: https://gitclear.com/research/coding-on-copilot-2024-developer-research

[34] A. Ndiaye, X. Taraj, B. Robeznik, G. Herzog, C. Gerber, and H. Purwins, "Generative AI in software engineering: Transforming the software development process," Accenture and DFKI, White Paper, Oct. 2025, accessed: Feb. 7, 2026. [Online]. Available: https://www.dfki.de/fileadmin/user_upload/DFKI/Medien/News/2025/Wissenschaftliche_Exzellenz/Generative_AI_in_Software_Engineering_Transforming_the_Software_Development_Process_2025.pdf

[35] J. Becker, N. Rush, E. Barnes, and D. Rein, "Measuring the impact of early-2025 AI on experienced open-source developer productivity," arXiv preprint arXiv:2507.09089, Jul. 2025. [Online]. Available: https://arxiv.org/abs/2507.09089

[36] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The SPACE of developer productivity," *ACM Queue*, vol. 19, no. 1, pp. 20–48, Feb. 2021.

[37] S. Rico and L.-M. Öberg, "Challenges and opportunities for generative ai in software engineering: A managerial view," in *Proc. ACM. Conf. Softw. Eng.*, Jul. 2025, pp. 1338–1344.

[38] M. Coutinho, L. Marques, A. Santos, M. Dahia, C. França, and R. de Souza Santos, "The role of generative ai in software development productivity: A pilot case study," in *Proceedings of the 1st ACM International Conference on AI-Powered Software (AIware '24)*. New York, NY, USA: ACM, Jul. 2024.

[39] T. Niemeijer, "Integrating generative AI tools into software development," Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2025. [Online]. Available: http://hdl.handle.net/20.500.12380/310917

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

[41] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," *Empir. Softw. Eng.*, vol. 27, no. 4, p. Art. no. 94, Apr. 2022.

[42] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in *Proc. Int. Conf. Softw. Eng. (ICSE)*,

Florence, Italy, May 2015, pp. 9–19.

A

Appendix - Interview Questions

1. Can you tell us a bit about your current role at Husqvarna?

- What are your main responsibilities?
- (Manager): Do you work with programming yourself?

2. What do you think about the use of AI in software development?

3. What does your use of AI currently look like?

- What types of tasks do you use AI for? (Give examples)
- How often do you use it?
- When did you (and your department) start using AI in your work?
- Do you use AI agents? (For example, Claude Code)

4. Have you developed any specific strategies for how you write your prompts or interact with AI?

- (For example, pasting file structures, setting roles, asking for step-by-step plans...)

5. In which phases of development do you use AI?

- (The idea/design phase, requirements, planning, coding, refactoring, documentation, testing or debugging)
- In which phases do you use it the most?

6. Do you perceive an increase in your productivity when working with an AI tool? What does AI help you save the most time on?

- (Manager): Do you feel that AI tools save time for your team, or does the time reappear later (for example in review or testing)?

- What would you say are the biggest bottlenecks or blockers in the development process today?
- What specific tasks or processes do you spend the most time on?

7. How much trust do you generally feel regarding the code the AI generates?

- How much of your time goes into verifying AI code or output compared to code you write yourself?
- Do you experience that there is a certain level of complexity before the tools start to hallucinate or make mistakes?

A1. Do you use AI to translate designs or mockups into functional code?

- *If so:* How?
- *If yes:* Do you feel it gives you more or less creative freedom?

B1. Do you let AI handle infrastructure, databases, and configuration?

B2. What are your ideas on delegating backend changes to an AI agent?

- *If skeptical:* What kind of proof or reliability would you need to see before making that shift?

C1. Do you feel that the AI models understand the context of the target hardware, or do you mostly get generic code that needs to be rewritten?

D1. Do you use AI to generate tests or for any other test-related tasks?

E1. Has the way you measure your performance (e.g., KPIs, story points...) changed since you started working with AI?

- *If not:* Do you think it should change?

E2. Integrating new tools requires significant time and resources for training both the organization and the individual teams. What potential challenges or bottlenecks do you see with this process today?

- How does the actual training happen in practice? For instance, do you arrange formal workshops?
- Do you have any initiatives or forums for peer-to-peer learning where colleagues can share knowledge with each other?

8. How has AI influenced how you view your own role?

- Do you see yourself more as an architect and reviewer than a coder when using

these tools?

- Has access to AI caused you to take on tasks that previously lay outside your role? (For example, are role boundaries blurring?)
- Do you experience increased pressure to deliver more or faster now that the tools exist?
- Do you think AI helps with dealing with complexity (both technical and organizational)?
- Does relying on AI help reduce your overall mental/cognitive load, or does it sometimes increase it (for example by forcing you to review AI-generated spaghetti code)?

9. Which skills do you perceive have become more or less important with the rise of AI?

10. What do you see as the biggest challenges with using AI in your field going forward?

11. What do you see as the biggest opportunities with using AI in your field going forward?

12. What do you think your role will look like in 2-3 years?

13. Is there anything else that you think is important to mention regarding your experience with AI?