The Effects of AI Assisted Programming in Software Engineering

An Observation of GitHub Copilot in the Industry

Master's thesis in Computer science and engineering

Johan Gottlander

Theodor Khademi

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

The Effects of AI Assisted Programming in Software Engineering

An Observation of GitHub Copilot in the Industry

Johan Gottlander
Theodor Khademi

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

The Effects of AI Assisted Programming in Software Engineering
An Observation of GitHub Copilot in the Industry
Johan Gottlander
Theodor Khademi

Supervisor: Robert Feldt, Department of Computer Science and Engineering
Examiner: Lucas Gren, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of a software engineer pair-programming with an AI.

This image was generated using Tome, a generative AI tool that can create images. The image was generated using the prompt "a developer writing code at a desk, with a robot sitting next to him giving instructions".

Typeset in LaTeX
Gothenburg, Sweden 2023

The Effects of AI Assisted Programming in Software Engineering
An Observation of GitHub Copilot in the Industry
Johan Gottlander
Theodor Khademi
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The recent emergence of artificial intelligence (AI) learning algorithms has brought generative AI (GAI) tools to the market. In software engineering, an example of such a tool is GitHub Copilot (Copilot), which can generate code suggestions in real-time and through natural language input.

In contrast to contemporary studies, this report attempts to fill a knowledge gap by employing a qualitative study, gaining insights into professional software engineers' opinions regarding GAI use in natural settings. While Copilot was the primary reference point, the study acknowledges the emergence of other GAI such as ChatGPT which also fit within the scope of the thesis.

The study was initially designed to let engineers use Copilot in their work for two weeks, followed by a semi-structured interview. However, hesitance from approached companies to use Copilot in their code due to legal and privacy concerns led to an alternative study design being used in tandem. Retaining the interview format and questions, participants were instead shown a demo showcasing Copilot's features. In total, 13 professionals participated in the study.

Through thematic analysis, findings revealed that utilizing Copilot can increase efficiency through auto-completion specifically. A lack of conversational capabilities and disruptive elements of Copilot lead to hindrances in development and code analysis. Furthermore, GAI tools allow engineers to focus on higher-level problems and offer inspiration, enhancing end-product creativity. Engineers also emphasized the retention of base knowledge to criticize GAI output. Finally, widespread GAI integration can lower the profession's entry barrier, and developer roles can shift to take advantage of the enhancements the tools provide. It is still evident that there are currently many concerns with the technology for trusted integration. Therefore, efforts should be made to address these issues, which in turn can make studies in natural settings more viable.

Keywords: generative AI, software engineering, field experiment, semi-structured interviews, problem-solving, programmer efficiency, AI's long-term effects

Acknowledgements

We would like to thank our supervisor Robert Feldt for the continuous help and support during this project, both regarding the content of the thesis and the structure of the report. We also want to thank our examiner Lucas Gren for reviewing the thesis and giving valuable feedback. We would also like to thank our opponents Marcus Axelsson and Daniel Karlkvist for giving us feedback on the report. Finally, we want to thank all interviewees who participated as this research could not have been done without your valuable reflections and perspectives.

Johan Gottlander, Gothenburg, June 2023

Theodor Khademi, Gothenburg, June 2023

Contents

List of Figures

List of Tables

1
Introduction

Artificial intelligence (AI) has had a profound impact on humanity for nearly a century [1] and due to recent advances in data collection and computer power [2], AI implementation and research have seen a rapid increase in different industries on a global scale including healthcare, human resources, finance, and many more [3]. While the main purpose of integrating AI use in these industries is to optimize the efficiency of more laborious, computationally intensive tasks such as face recognition and natural language processing [1], the emergence of AI in the world of creativity and arts in recent years has also been seen through the utilization of deep- and reinforcement learning in large neural networks [4]. In the music world, programs such as *Amper*, *AIVA*, and *FlowComposer* can generate completely new compositions of music and help composers in their work through incremental suggestions [5]. *Dall-E 2* is another example of an AI in the art world generating new unique art pieces, or expanding upon existing ones. According to the AI research laboratory *OpenAI* who created the tool, Dall-E 2 can "create realistic images and art from a description in natural language" [6].

The world of software development has also received a similar tool as FlowComposer and Dall-E 2 in the name of *GitHub Copilot*, hereinafter referred to as "Copilot". Copilot is an AI tool that the version-control hosting service GitHub announced in June 2021 as an extension for the integrated development environment (IDE) Visual Studio Code (VSCode) [7]. Copilot is powered by Codex, an AI model based on deep learning that has been trained on existing source code on GitHub to translate natural language to code [8]. Developed by OpenAI as well, Codex is a specialized model tailored to computer code generation and is a descendant of the model Dall-E 2 employs. Copilot makes suggestions on code snippets, entire functions, and test cases in real-time while the user is writing code [9] (see Figure 1.1). The user can also ask Copilot to produce code suggestions by writing a comment in natural language inside the codebase (see Figure 1.2).

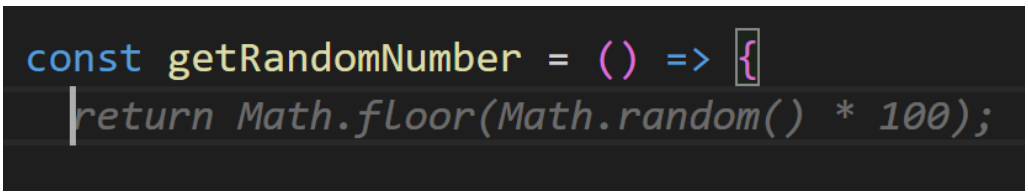

Figure 1.1: Example of the auto-completion feature in Copilot. The text in gray is what Copilot is suggesting.

```
// Function that returns a random number between 1 and 100
const getRandomNumber = () => {
```

```
// Function that returns a random number between 1 and 100
const getRandomNumber = () => {
  return Math.floor(Math.random() * 100) + 1;
```

Figure 1.2: Example of Copilot's commenting feature. The user writes a comment in natural language and presses enter to come to a new line. Copilot then gives suggestions row by row or entire functions all at once.

In the relatively short time it has been available to the public, research on Copilot has predominantly focused on analyzing quantitative data in empirical studies (see Section 2.5). The aim of these studies has been on analyzing the tool itself, for example with what efficiency and quality it helps solve tasks, how understandable the suggestions are to humans, and if the suggestions produce any bugs. In contrast to these studies, this thesis aims to look at Copilot from a different perspective, that being from the standpoint of the programmer itself. By observing the cognitive processes of software engineers using this tool during normal work, the goal is to analyze the effect Copilot has as a pair programmer on the profession itself. This includes investigating the advantages and hindrances of using the tool in their work; if the tool in any way can alter how a programmer thinks about solving tasks; and how it can affect the prospects of the software engineering (SE) profession.

1.1 Problem Statement

As mentioned earlier, contemporary research on Copilot has primarily focused on evaluating Copilot itself and analyzing quantitative data in empirical studies. For example, one study evaluates the accuracy and quality of Copilot's code suggestions [10]. Another study analyzes the security risks that the code suggestions present [11]. Finally, one study does analyze the effectiveness of having Copilot as a pair-programmer versus a human pair-programmer [12], but the focus is again on the effectiveness of the tool and not on the actual experience of using it or its implications on the profession as a whole.

The trending paradigm is that AI will become more and more integrated with human life as assistive tools in our work [3]. As seen in the aforementioned studies, various research has already been conducted to analyze Copilot as an assistive tool when writing code. However, one aspect that is not being addressed in this research is the cognitive effects that programmers experience while using it in their day-to-day work. The general problem with this lack of research and knowledge gap is that it could be hard to determine the potential long-term effects humans experience through the continuous use of AI tools such as Copilot, and the potential risks for the SE industry when integrating such tools. Could the use of AI limit creativity in a programmer's problem-solving process? Could programmers start to lack a sense of purpose as AI becomes more and more sophisticated at doing their tasks for them?

1.2 Purpose

This study aims to observe and explore professional software engineers' experience and thoughts regarding Copilot. However, while Copilot is the primary focus and starting point of discussion, it is natural that an extension of other tools will be discussed as the study acknowledges the recent emergence of generative AI (GAI) tools in the world. This will be done in two ways: (1) professionals will use the tool in their day-to-day work in a real-world setting and (2) professionals will be shown a demonstration of the tool in a contrived setting to familiarize themselves with its features. In both cases, interviews will be held in purpose to investigate the experience had by the professionals. In contrast to other contemporary studies on Copilot, this study aims to gather qualitative data with the programmer in focus and highlight potential opportunities, challenges, and risks of using the tool. This study intends to benefit both practitioners and other researchers. It can provide practitioners insight into how well the tool can affect their day-to-day work, rather than only in a controlled laboratory setting. Furthermore, it can clarify the opportunities and risks of using the tool and other AI tools for programming. For researchers, it can help construct a stronger knowledge base on the issue of GAI tools utilized in natural settings and bring light to potential issues and/or elements of strength the tool has for the industry, so that they can be researched further.

Three research questions (RQs) will be used in this study to mainly investigate Copilot's impact. As mentioned however, because of the recent developments of GAI, it is natural that the findings will reflect on other tools as well. This can especially be true for the third RQ which focuses on the broader/macro level. The first two focus on the individual/micro level.

RQ 1: *What features of the tool help programmers be more efficient in their work and conversely what features hinder them?* We want to investigate how much help the tool can offer programmers and where it potentially could be improved.

RQ 2: *To what extent can the tool affect programmers' problem-solving process?* We want to see if the tool in any way shapes a different cognitive process for solving the problems programmers face in their work compared to what they previously had.

RQ 3: *Can the potential possibilities and/or hurdles introduced by the tool change the prospects of software engineering as a profession?* Here we want to evaluate the results elicited from the previous research questions and investigate the tool's effect on the profession.

1.3 Limitations

The intention of this study is to investigate the effects of Copilot on professional software engineers. Students were also considered as candidates for participation but ultimately not chosen as the study's findings aim to answer questions more related to the effect on the industry.

The tool to be studied was narrowed down to Copilot, even though there exist other AI tools for programming assistance, for example ChatGPT [13]. This is due to the fact that while ChatGPT can offer code suggestions, it is not solely meant for programming, and is more of a general-purpose tool focused on natural language communication.

2
Background and Related Work

This chapter will provide context and background information for the thesis by exploring different aspects of AI technology. First, a brief overview of key concepts of AI will be presented. Next, the history of AI development, its ethical implications, and the recent rise of other GAI will be explored. Finally, other work related to this thesis will be presented to review relevant literature.

2.1 Key Concepts of AI

A brief overview of key concepts used in this thesis pertaining to AI technology will be explained to clarify their significance and give readers a more comprehensive knowledge surrounding the research topic.

Deep learning is a subset of machine learning and involves an artificial neural network that tries to simulate human intelligence by "learning" from large sets of data. Typically, these networks comprise of three or more layers. Due to its capacity for learning from significant amounts of input, it is especially beneficial for tasks like speech and image recognition [14]. In contrast to more traditional machine learning, deep learning uses unsupervised data to learn.

Reinforcement learning concerns a learning paradigm of AI where the goal is to optimize sequential decisions. Broadly, by applying a reinforcement learning algorithm, AI can learn similarly to how humans learn by continuously receiving rewards for finding the best strategy to navigate an environment, otherwise known as a policy [15].

Natural language processing (NLP) is another branch of machine learning. NLP studies how to use natural language to connect computers and people by understanding more complex aspects such as tone and context. NLP uses methods such as deep learning to give computers the ability to comprehend, decipher, and create human language. Sentiment analysis, chatbots, machine translation, and text summarizing are just a few of its many useful applications [16].

Large language models (LLM) are AI algorithms with large sets of parameters trained on huge datasets of text to generate text akin to humans, respond to prompts, and accomplish other language-related activities with precision [17]. An example of an LLM is GPT-3, having 175 billion parameters and is trained on 570 gigabytes of text [18].

Codex is an OpenAI-developed AI system and can produce code from inputs in natural language. It creates the code that best satisfies the provided requirements after using a deep learning model to comprehend the intended purpose of generating the code. Codex might accelerate the software development process considerably, which would boost developer productivity [8].

2.2 A Brief History of AI

AI as a concept was already being discussed in ancient Greek mythology where the poet Homer talks about the God Hephaestus, who forged automated assistants made of gold but appeared as young woman [19]. They were imbued with intelligence and strength by the gods, making this one of the first mentions of intelligence artificially made. Homer also wrote of mechanical assistants called “tripods”, waiting for the gods at their dinner tables [20]. While AI has been a topic of fiction since then, the first real “machine intelligence” seen was created by the English mathematician Alan Turing during the second world war [21]. Turing created a machine that could break the Enigma code used by the German Army. This led to Turing publishing an article named “Computing Machinery and Intelligence”, in which he described both how to create intelligent machines and most importantly, how to test them. This test, which Turing initially calls the “Imitation Game” but is now called the “Turing Test”, is meant to analyze a machine’s capacity to demonstrate intelligent behavior comparable to, or indistinguishable from, that of a person [22]. Essentially, the test consists of a human evaluator having two conversations, one with another human and one with a machine. The evaluator is not able to see who they are having the conversation with and it is limited to a text-only channel. The machine passes the test if the evaluator could not distinguish the conversations between the computer and the human [22], and is still used today to determine the intelligence of an artificial system [21].

There were steady improvements to AI technology after the publishing of Turing’s article. One example is in 1956 when Dartmouth College located in New Hampshire, USA, hosted a research project on AI [21]. This project had a high significance as participants included Nathaniel Rochester, who designed the IBM 701 which was the world’s first scientific computer, and Claude Shannon, who created information theory. The research on AI continued for almost two decades but saw some decline in funding in the mid-70s. It was not until the early 80s that funding started increasing again [21].

While AI continued to be researched and improved, the first major breakthrough in modern AI came in 2015. Google presented a program called *AlphaGo* which is a computer program that was able to beat the world champion in the board game Go [21]. Notably, Go is far more complicated than chess, where at the start of a game there are 361 potential moves compared to 20 in chess. It was long thought that computers would never be able to outperform humans in this game. Google, however, accomplished this by using deep learning for its AI. Since then, the majority of AI has been developed using this technique, forming the basis for algorithms pertaining to image and speech recognition, NLP, etc [21].

2.3 Generative AI and ChatGPT

In short, GAI refers to a machine learning technology that through human input can automate the output (generation) of a large quantity of new content/media that is seemingly plausible for a human to create themselves. This can be in many different forms as presented earlier in the introduction, such as audio, images, text, etc [23]. Due to the increases in data volumes for training the models and improvements in more advanced algorithms, computer processing power, and data storage [4], these GAIs have in recent years garnered popularity not only in the academic and research world but also on the commercial side due to their more sophisticated human-like behavior [24]. GAI tools are also offering public and more accessible interfaces, which in turn also leads to higher user bases being observed. One example is ChatGPT, reaching one million users within five days of its release in November 2022 [25].

ChatGPT was created by the same company (OpenAI) that powers the model for Copilot and is an AI chatbot focusing on providing human-like conversation to its users via text. It is capable of handling a large range of queries, including code generation [13] (see Figure 2.1). As the name suggests, the AI model is built using a natural language processing deep learning framework called Generative Pre-trained Transformer (GPT), which takes in a substantial amount of data in both supervised and unsupervised training to comprehend and output human-like conversation [26]. More specifically, ChatGPT uses a so-called GPT-3.5 model which was created by OpenAI to enhance task completion and the ability to follow instructions compared to other earlier models such as GPT-3, which powers the previously mentioned Dall-E 2 [13]. As of March 2023, OpenAI has also added a feature to premium users of ChatGPT to enable another model called GPT-4, which is said to further improve on GPT-3.5 by enhancing creativity, having a greater ability to handle longer text inputs, and even offering visual input [27].

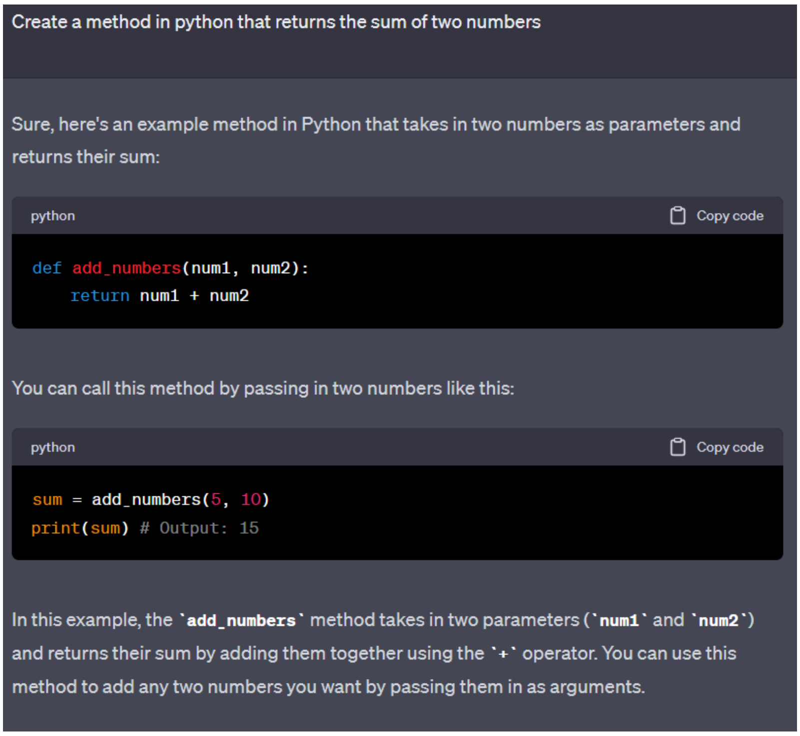

Figure 2.1: Simple example of code generation from ChatGPT.

While GAI tools have only existed for a relatively short amount of time, their impact can already be seen on many different levels of society, especially in education. Dwivedi et al. provide accounts from experts in different fields discussing the rise of ChatGPT and its implications [28]. The experts discuss the disruption ChatGPT has caused, due to its effectiveness in producing larger amounts of text with at least perceived high quality that can turn homework obsolete. While tools have been created to detect texts written by AI, the experts do not agree that a ban on these tools is useful, as students will live in tandem with these technologies in the future. They compare the introduction of ChatGPT with the introduction of the calculator. While controversial at first, it ultimately had to be used in educational settings as the technology was being used everywhere else.

2.4 The Ethical Intricacies of AI

AI can present a plethora of opportunities. Maedche et al. discuss the expectations of opportunities that AI-based digital assistants could bring, such as resources needed for more routine tasks can be significantly lowered, which in turn leads to more time for more demanding tasks [29]. One example that Maedche et al. present is IBM, which claims that AI-powered chatbots can reduce the cost of customer service by 30%. This can also be seen in the investigations that Kumar [3] leads in the human resources departments of different industries, where many were positive towards the introduction of AI assistants in their everyday work to eliminate more mundane tasks. Siau and Wang add to this sentiment by discussing other positive impacts of AI, such as economic growth, social development, human well-being, and safety improvement [30].

On the other hand, while AI continues its evolution to become more and more autonomous and integrated into human life, various complex ethical questions and issues are increasingly raised [31] [29]. Gratch and Fast identify that the degree to which software consumers can create unethical behavior in AI agents is a more challenging conundrum than the commonly researched issue of how the development of AI can introduce human biases [32]. As seen in the rise of GAI, consumers are given an increased amount of interactions with the agent itself, giving them the ability to customize, or "program" as Gratch and Fast call it, the AI's behavior, goals, and values directly. Gratch and Fast tested this claim through three different studies. The first was allowing a human to personalize their own AI agent to handle legal negotiations. Secondly, they let humans personalize AI assistants as tutors or coaches. Lastly, participants were placed in a managerial role and then asked to choose whether to conduct employee check-ins themselves or through an AI avatar. The results of these studies found respectively that humans were most likely to deceive through AI, received less blame for the unethical behavior that their AI caused, and that an AI avatar was a preferred choice due to humans anticipating less criticism towards them by the AI.

Flick and Worrall have also identified other ethical issues specifically within GAI, one being copyright infringement [33]. AI needs to be trained on existing material produced by humans and because of this, the output of the AI is legally ambiguous. It is also hard to determine who to blame for a potential infringement, i.e., the user of the AI tool or the creator of it. Another issue identified by Flick and Worall that is often brought up is the possibility of author/artist replacement. With an interface that is easier to use, people who are not in the artistic/creative field can produce work themselves without much hindrance, potentially rendering a subdivision of professional capability as seemingly superfluous.

2.4.1 Ethical Dilemmas of Copilot

Copilot also falls under the umbrella of GAI as it authors code with AI technology based on human input. Because of this, the tool has not been without issues being raised about it, both in ethical and legal realms. Caballar presents and discusses a class-action lawsuit that has been filed against GitHub, Microsoft (GitHub's parent company), as well as OpenAI [34]. According to the lawsuit, the code created by Copilot does not include any acknowledgment to the original author of the code, copyright notices, or a copy of the license, all of which are required by most open-source agreements. This lawsuit is unprecedented in that it is the first to challenge GAI and will set a benchmark for other jurisdictions on the subject. The defending entities of this lawsuit have moved to dismiss the plaintiffs' claims, stating that the complaints fail by "lack of injury", "lack of an otherwise viable claim", and a lack of providing real scenarios to which someone was personally harmed by the tool. As of writing this report, the claim has yet to be resolved by the court of California, as the hearing of the dismissal will take place in May 2023 [35].

While the lawsuit focuses on the licensing issues that Copilot has, there are other risks and ethical shortcomings that large language models, such as Codex, can face. Two examples of such risks are security and privacy. As these models are trained on massive amounts of code that are not manually curated, there is a chance that vulnerable and insecure code can be suggested by them, and Copilot is no exception. In a study by Pearce et al., it was observed from 1689 programs completed with the help of Copilot, that 40% of them had some vulnerability in their code introduced into them [11]. It should be noted that the code generated by Copilot in Pearce et al's. study cannot be directly reproducible, and alternative percentages could be derived in other studies regarding vulnerability. The study was also controlled in the sense that artificial scenarios were created to produce these programs, limiting the insights one can connect the study to more real-world coding scenarios. Pearce et al. do however conclude that developers should not blindly accept the suggestions that Copilot generates, and be aware that it can introduce vulnerabilities. This is something that GitHub also discusses in their Copilot FAQ [9], stating that: *"Public code may contain insecure coding patterns, bugs, or references to outdated APIs or idioms. When GitHub Copilot synthesizes code suggestions based on this data, it can also synthesize code that contains these undesirable patterns. ... Of course, you should always use GitHub Copilot together with good testing and code review practices and security tools, as well as your own judgment."*.

On the topic of privacy, since code can contain sensitive information like credentials, personal information, or even in-code discussions by developers, large language models could be trained on this data. There is therefore a chance that sensitive information can be extracted from the model [36]. When asked if Copilot can output personal data, GitHub responded with this in their FAQ: *"Because Codex, the model powering GitHub Copilot, was trained on publicly available code, its training set included public personal data that was included in that code. From our internal testing, we found it to be very rare that Copilot's suggestions included personal data verbatim from the training set. In some cases, the model will suggest what appears to be personal data – email addresses, phone numbers, etc. – but those suggestions are actually fictitious information synthesized from patterns in training data and therefore do not relate to any particular individual"*. GitHub also offers two settings for Copilot handling data both in terms of input and output. One setting allows a user of the tool to block Copilot from allowing suggestions matching public code, and another is to block GitHub from using one's code snippets for research purposes and product improvements [9].

2.4.2 Addressing the Issues

As ethics in AI is regarded to be in its inception stage, one crucial concern that is yet to be fully mapped out is how to address these issues being raised [30]. Danaher attempts to analyze and evaluate some of the concerns to form a framework for thinking about AI in a more harmonious, ecological way on an individual level [37]. Danaher presents a comprehensive review of several different issues raised by other authors where AI assistants potentially inflict harm to mankind. One such is the degeneration argument “If anything that forces us to use our own internal cognitive resources enhances our memory and understanding, then anything that takes away the need to exert those internal resources will reduce our memory and understanding”. Essentially, one can lose cognitive ability when using AI assistants to perform tasks over an extended period, as the brain is not stimulated enough. Danaher concludes that one should be aware of the risk that this could happen depending on the primary value and the risk of decoupling from the AI. If the primary value of the task is intrinsic to oneself and/or the risk of decoupling from the AI is high, one should try to use their own cognitive ability to perform the task. Another issue is autonomy. It is something that is cherished in modern society, and AI assistants have the potential to reduce it by both removing the channels between our choices and their results, as well as homogenizing the way we construct decisions. Danaher argues however that such forces that reduce autonomy already exist in other forms such as in one’s culture and in one’s general environment. The approach to not letting those factors manipulate one’s decision-making should be the same with AI.

Dhirani et al. identify an issue in that compared to creative technological developments, there has been less growth in the ethical arena regarding AI. In their article, they give an overview of the ethical standards for AI technology currently being developed by regulatory bodies [38]. The authors chose to focus on the European Union (EU), which has already established regulations regarding technology in the General Data Protection Regulation (GDPR). Looking at more recent history, the EU has in 2022 passed several different acts in an attempt to further form a legal framework around technology, those being the EU Artificial Intelligence Act, the EU Digital Services Act, the Digital Markets Act, and the EU Cyber Resilience Act. Focusing on the EU AI Act, its goal is to form a framework around AI to enhance trust and limit possible harm that the technology may create. According to the EU, it is the first law on AI by a major regulator anywhere [39]. While Dhirani et al. focus on the EU and its strides in ethical AI, they do also put the issue of interoperability between regulatory bodies to light. Large industries usually have multiple production regions/setups all over the world (eg. Europe, the United States, and China) and are subject to different jurisdictions, rules regarding data, and compliance. There is much work to do when it comes to regulating AI and as the acts mentioned above are very new, it is hard to measure their impact. Moreover, it is challenging to determine the threats of emerging technologies such as AI until they have been used over several years. Dhirani et al. also mentions that the AI Act should comply more with GDPR to create a more cohesive regulation. Despite these issues, the acts are a starting point and have the potential to set up a guideline for other regulatory bodies to follow [38].

While having regulations and guidelines on how to build ethical AI is important on the government level, it is also critical that the businesses in the AI industry take their own measures and governance practices to reduce the risks associated with AI [40]. Eitel-Porter provides a multi-step framework for businesses to ensure that the AI being developed is ethical and free of risk, not only during the model-building and development stages but also during the deployment and scaling of it. The framework is mainly based on the five common pillars of ethical AI: Fairness, Accountability, Transparency, Explainability, and Privacy [40] and includes for example creating an ethics board in the business, establishing metrics to ensure that AI principles are followed, etc. The framework would assist to safeguard businesses from danger and build customer trust in their services, which will help to boost usage even further, both in private and enterprise settings.

2.5 Related Work

A review of related studies will be conducted with the intention of offering a more thorough understanding of the topics of Copilot, using large-language models, and conducting experiments in natural settings in the field of SE. This will mainly be done by performing in-depth evaluations of past studies that are relevant, highlighting their contributions and limits to understand the present state of research within the topics. The review can also contextualize the research undertaken in this thesis and demonstrate the gap in the literature that this thesis attempts to address.

2.5.1 Empirical Investigations of Copilot

In an empirical, quantitative study on Copilot by Imai [12], the results indicate that using Copilot produces more lines of code but with a lower quality compared to pair programming with a human. 21 persons with at least some programming skills took part in the experiment where the number of lines of code added as well as deleted was recorded. When using Copilot compared to a real human as a pair programmer, more lines were deleted, hence worse quality. One explanation for this could be what Dakhel et al. discuss in their study [41], as they discovered that Copilot had trouble understanding details in the description of what to code. For example, telling it to return a list in an order that "...older people are at the front..." was much harder for it to understand than telling it to return the list in "...descending order...". They also noticed that Copilot could have difficulties understanding long descriptions of a problem to solve, and as a result, it could misunderstand an entire problem. However, it was also concluded that even though Copilot could not always satisfy the description completely, the developer could incorporate the code generated by Copilot with little to moderate change. Copilot's difficulties to understand longer descriptions were explained further by Chen et al., showcasing that Codex with 12 billion parameters performs exponentially worse as the chained building blocks in a prompt increase [42]. An example of a building block could be "convert the string to lower case", and a chain of building blocks could be the prompt "Convert the string to lower case, then remove all instances of the letter e from the string, and finally add the word apples after every word in the string". Prompts like these could be hard for Copilot to complete, but not so hard for a professional developer.

2.5.2 Programmers' Experience using LLM-powered Tools

There are a few studies that investigate Copilot and similar tools from a qualitative aspect, such as the usability of the tools and programmers' general experience. In a within-subjects user study [43], Vaithilingam et al. found that the majority of the participants preferred to use Copilot over VSCode's default code completion tool called *Intellisense* when solving programming tasks. The task completion time when using Copilot was not statistically significant, and the participants using Copilot failed three more tasks compared to the ones using Intellisense. However, the qualitative results of the study indicate that Copilot offers features that affect the programmer's experience to such an extent that 23 of the 24 participants stated that Copilot was more helpful than Intellisense. The study had a focus on how the user perceived Copilot and how the user interacted with its code suggestions, and one reason why participants favored Copilot was that it helped them get started with a solution, even if the suggestions were not always completely correct. On the other hand, some participants further explained that they found it hard to understand and correct code that was incorrectly generated, while some had an over-reliance and trusted the suggested code as it was. As a result, they came to the conclusion that there is room for improvements such as adding support that makes it easier for the programmer to understand and validate the generated code. It should be noted that only one of the 24 participants was a software engineer and the rest were either undergraduate, master's, or Ph.D. students but all except one had at least more than 2 years of programming experience.

Ross et al. have also conducted research on users' interaction with an LLM-powered developer tool which they developed and called the *Programmer's Assistant* [44]. The tool combines a code editor with a chat interface which is powered by the Codex model through its web API. Compared to Copilot, this assistant does not give real-time suggestions but only responds to the programmer's request, and the suggestions must be transferred from the chat to the editor via copy/paste. The study did not track metrics such as code quality or time to complete a task but focused on the participants' (mainly software engineers) attitudes toward the tool. The result was divided into three sections: (1) expectations and experience, (2) utility of conversational assistance, and (3) patterns of interaction and mental models. Regarding (1), the participant's overall experience was beyond their expectations. Similar to previous studies [43] the generated code was not always correct (correct 80.2% of the time), but most participants still found it very helpful as minor tweaks often solved issues. Regarding (2) they found a conversational assistant valuable, especially when solving smaller and simpler tasks. Since the participants could have a conversation with the assistant, they also found it helpful by asking it to explain and document the code. Regarding (3), two patterns were discovered where one was that participants asked the assistant to solve complete programming challenges, and the second was that participants broke down the challenge and asked for help from the assistant for each sub-challenge. Overall, the participants felt that their workflow was impacted for the better since the assistant could help with lower-level details, allowing the programmer to focus on development on a higher level and speeding up work.

2.5.3 Limits and Risk Factors Using Language Models

Codex is a descendant of the language model GPT-3 created by OpenAI [42]. GPT-3 has about 175 billion parameters, compared to the previously biggest language model (Microsoft's Turing NLG model) with 10 billion parameters. However, generalization comes with the cost of performance which is why GPT solves 0% of a set of problems compared to Codex, which has been specifically trained on code and can solve 28.8% of the same problems having around 12 billion parameters with just a single sample [42]. A more fine-tuned version of Codex named Codex-S can solve 77.5% of the problems with 100 samples.

In the same paper, Chen et al. identified several risk factors which help to better understand the hazards of using language models like Codex. One risk factor is "misalignment", which is described as even though Codex is capable of performing task X, it "chooses" not to, i.e., it is not incompetent of performing the task but does not do it due to various mistakes. This is concerning as it is a problem that could become worse when scaling up data, parameters, and training. Another risk discussed by Chen et al. is "over-reliance", meaning that Codex can suggest solutions that appear correct but do not perform the asked task, such as producing insecure code. This can be related to the risks identified by Challen et al., who discuss quality and safety issues using AI and divides them into short, medium, and long-term [45].

The research was done to identify issues using AI within medicine, but the issues could be applied to SE to some extent as well. The long-term risks are taken from the framework by Amodei et al., discussing AI's potential risks [46]. One of the long-term risks of using AI is the negative side effects and it is discussed how to prevent AI from disturbing the environment while pursuing its goals. This is relatable to "over-reliance", as Codex can suggest insecure code while trying to complete a task which is an example of disturbing the environment. One of the short-term risks discussed by Challen et al. is "distributional shift" which is explained as a mismatch between the environment the data has been trained in and the environment it is used in. This is relatable to "misalignment" since Codex suggests code that is as similar as possible to its training distribution.

2.5.4 Field Experiments in Software Engineering

Given that field experiments will be a part of data collection in this thesis, taking a look back and reflecting on other studies that have been done with a similar technique in the same field can be pertinent to improve the thesis study design and give the thesis a reference point. One such study was made by Grimstad and Jorgensen, who explored the effects of manipulating variables in a natural setting in which software companies estimated the effort (time) it would take to complete the same project [47]. What Grimstad and Jorgensen recognized through previous studies was that software developers often subconsciously were having their judgmental processes affected, both by relevant and irrelevant information, such as the client's expectations, variations in the wording of the requirements, and wishful thinking

[47]. However, these previous studies were in a controlled laboratory environment, which could lead to certain threats to their validity. For example, the high time pressure in a laboratory setting could affect the expended effort more than what is seen in a natural setting. Focusing all efforts on estimating could also lead to more causal variables [47]. Therefore, Grimstad and Jorgensen recognized a need to study the estimation of software projects in a field setting as well. Recognizing a lack of knowledge in a more natural setting was identified in the context of this thesis as well. As mentioned in Section 1, many of the studies on Copilot were done in a controlled fashion, giving a precedent to fill the gap in knowledge and employing a similar study design. Looking at the results, Grimstad and Jorgensen documented a decrease in effect sizes when comparing their earlier laboratory studies and the experiment they conducted, indicating that the irrelevant information taken into an estimation of a project is less likely to affect the judgment process [47]. One can not always rely on controlled laboratory experiments to generalize findings, as they might differ from what is happening in reality, hence the significance of conducting such a study as the one this thesis will present.

Another study using the field experiment design was done by Anda et al. [48]. They conduct "a longitudinal multiple-case study of variations and reproducibility in software development, from bidding to deployment, on the basis of the same requirement specification". This was done by giving four software companies the same project to complete, that being the manipulated variable, and observing the differences in reproducibility of different aspects such as code quality, schedule overrun, and actual lead time. While the article by Anda et al. offers little in terms of direct comparison to this thesis purpose, one can still find interesting details and general erudition that can be applied and taken into account in this thesis. When discussing external validity, the authors reflect upon how much they actually can generalize their findings, and realize that it is a difficult task. Counteracting effects such as the size of the companies could partly affect variability as well as the degree to which the company can be flexible. Therefore, the authors restrict their generalizations to smaller Norwegian companies developing smaller Java systems. Generalization restrictions should therefore be acknowledged for the findings in this thesis as well.

3 Method

This chapter presents the research designs and methodologies used to investigate the impact of integrating Copilot into software engineers' work. Firstly, a description of the recruitment process will be presented, within which an alternative study design had to be made to support an unforeseen hesitance to participate in the study's original design. Therefore, two different strategies were employed, named Study Alternative 1 (SA1) and Study Alternative 2 (SA2). These will be presented in detail in Sections 3.2.1 and 3.2.2 respectively. Furthermore, a description of the data collection and data analysis used to generate the findings for this thesis will be provided. This will include arguments to support the choices made for the different techniques.

3.1 Participants

The goal of recruiting participants was to get professional software engineers working in different fields to get as good of a spread as possible, allowing better generalization of the data gathered to the SE profession. The methods used to recruit participants varied, such as posting an advertisement on LinkedIn, but in the end all participants were recruited due to personal contacts.

Originally, the study only had one design in which the participant was supposed to use Copilot for two work weeks, followed up with an interview. However, very early in the recruitment process, it was recognized from the first six larger companies contacted that there was a hesitance to join the study due to the legal problems that Copilot has faced. There was also worry expressed about security and privacy elements that could be introduced when integrating Copilot into their code. There was however a large interest in the topic and people in the approached organizations were eager to discuss GAI in programming. Because of this, a new approach or alternative for the study was created to be able to remove any security, privacy, or legal concerns and recruit more participants to get a good enough sample size. After creating the new alternative and offering it in tandem with SA1 in the recruitment process, more participants were recruited for SA2.

In total, 13 participants were recruited, with three participating in SA1 and 10 participating in SA2. The participants were guaranteed anonymity and are therefore given a participant number rather than a name, and their demographic data can be seen in Table 3.1 below.

Participant	Main area of profession	Age	Study alternative
1	Web developer (consultant)	27	1
2	Fullstack developer	29	1
3	Frontend developer	24	1
4	Solution architect	37	2
5	Fullstack developer	27	2
6	Fullstack developer	30	2
7	Fullstack developer (consultant)	29	2
8	Fullstack developer (consultant)	26	2
9	Sofware developer	26	2
10	Backend developer	34	2
11	Backend developer	27	2
12	Fullstack developer	26	2
13	Fullstack developer (consultant)	28	2

Table 3.1: Demographic data of the participants.

3.2 Research Designs

This section will describe the different approaches and execution strategies used for SA1 and SA2. While they do differ in many aspects, both research approaches sought qualitative data and insights on how AI technologies such as Copilot may influence the SE profession.

3.2.1 SA1

There are several research strategies that can be used in SE. In a framework by Stol and Fitzgerald [49], eight different strategies are described that can be used depending on what level of generalizability or obtrusiveness one wants in the study. For SA1, the strategy of the study was designed to fit the field experiment model that Stol and Fitzgerald present. Field experiments, together with field studies are conducted in natural settings to capture realism gained at the price of low generalizability. The main difference between the two is that in field experiments, the researcher can manipulate properties or variables to observe an effect as the researcher is not manipulating anything in a field study. SA1 was conducted in a natural setting with manipulation of a variable to study its effect [49]. The setting was the participants' workplace and the variable was Copilot, allowing evaluation of how the tool affected them in their work.

For this option, participants were asked to set up Copilot on their own device and use it during their day-to-day work for two weeks, i.e., ten work days. An initial meeting with the participant was held to present the purpose of the study, give them instructions on enabling Copilot on their IDE, and inform them of the legal and ethical dilemmas that Copilot has faced. A consent form was also sent out to the participant for them to sign electronically. When this was signed and sent back, the two weeks were considered to have begun. During these two weeks, the participants were asked to answer a set of questions in a survey at the end of each day that asked about their experience and how helpful they found the tool. A semi-structured interview was held after two weeks of using Copilot and allowed for a deeper dialog and qualitative data to be gathered.

3.2.2 SA2

Getting people to participate in SA1 was harder than expected and the main explanation was that companies had to turn down the offer to participate due to legal and privacy concerns about using Copilot in their work. However, the most important goal of this study was to interview professional software engineers to get their perspectives on how Copilot and similar AI tools can affect the profession, and it was therefore not necessary for them to use Copilot in their day-to-day work to accomplish this. Therefore, SA2 was created and subsequently offered in the recruitment process as an alternative to participating in the study.

From the discourse in the recruitment process with larger organizations, it was noticed that a lot of people working in the industry have a large interest in tools like Copilot and ChatGPT. Most have used them in one way or another and were eager to share their opinions and experience about them, which coincides with the aim of the thesis. To retain the qualitative study design and gather data similar to what SA1 could collect, SA2 was designed to include the same interview format and questions. In contrast to SA1 where the participants would have guaranteed experience with using Copilot, this could not be assured for the participants in SA2, as this thesis does not limit participants to only ones with prior experience with the tool. To retain the same interview format and questions, a demonstration of the tool's capabilities was chosen as an alternative method to provide the participants with fundamental knowledge about Copilot and its features before conducting the interview. The demo was showcased on the interviewer's own systems to remove the potential legal, privacy, or security risks for the professionals. Before the demo and interview session, a consent form was sent out to the participant for them to sign electronically.

In reference to the framework provided by Stol and Fitzgerald [49], the design of SA2 does not align with any of the eight research strategies directly. It does however incorporate elements of different research strategies discussed by the authors, such as gathering data from experts discussing a given topic of interest (judgment study) and setting up an environment that would not exist without the study (experimental simulation). As the demo was shown in a contrived setting with professional software engineers (experts) discussing Copilot as the topic of interest, SA2 can be viewed as a separate, artifact-driven research strategy inspired by the aforementioned designs.

3.2.3 Creating the Demo

All demo sessions for SA2 aimed towards creating the same small-scale full-stack application using the so-called “FERN”-stack, a collection of four technologies that can be used to build a web application. The technologies are Firebase, Express, React, and Node, hence the name “FERN”. Firebase is a platform developed by Google and offers services such as a real-time database, authentication, storage, and hosting. Express is a web application framework used on top of Node and it helps build web API’s quickly and easily among other features. React is a JavaScript library used to build a web application’s client side, i.e., the user interface. Finally, Node is a runtime environment used to run JavaScript code and allows running web applications outside the client’s browser.

A separate, finished version of the demo application was created first with the help of Copilot. This was done to provide a finished product to show the participants before demoing the tool what the interviewers were going to work towards. It also provided a form of backup to the interviewers when implementing the demo if they got stuck. To uphold the “FERN”-stack, a Firebase/Firestore database was created that could be used both by the already finished version and any subsequent demo attempts. A simple Node server was then created for the backend part of the application and an Express server was instantiated within that. A React application was created for the client side (frontend), and the Firestore database was then configured in the backend application.

The idea of the application was to fetch a random quote from a dummyJSON API and fetch data (name and timestamp) from the database to display on the page (see Figure 3.1). Users were then supposed to insert that quote and their name into respective inputs and submit it. Their name, along with a timestamp, was then inserted into the database. The backend handled adding and fetching data, and API routes were created which the frontend could access.

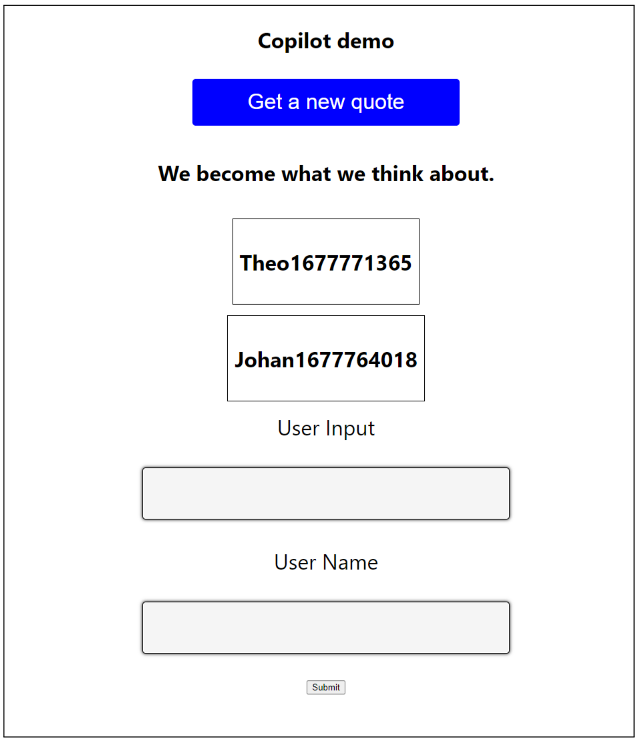

Figure 3.1: Original version of the demo.

Copilot was used in tandem when developing this original application to produce various code. For example, configuring Express on the backend side and helping with writing the routes for fetching and adding data to the database. On the frontend side, it helped structure HTML, styling, and testing. Using Copilot while implementing the original application gave an idea of where its features could be shown during the other demos.

For the sessions with participants, a separate project was created for the showcase which was uploaded to GitHub via the version control tool Git. To save time, certain parts of the demo were prepared before the sessions. On the backend side, the configuration of the database, the API route for fetching data, and all necessary imports of libraries and configurations were implemented. On the frontend, necessary imports were created as well, along with the request to the route in the backend to fetch data and an HTML skeleton.

3.2.4 Demo

Before every session, a new Git branch of the demo was created from the main branch version that only had the preparations. This removed the need for multiple projects and only required one. During the sessions, the participants were first shown the original, finished version of the application that the interviewers were going to work towards implementing. They were then explained the technologies used for the application. The participants were also told that while the interviewers were doing the coding, they were allowed to interrupt to ask questions or give suggestions of something they wanted to test Copilot on. The demo aimed at being a maximum of 30 minutes. After this time, the participants were quickly shown the code for the original version of the application to give them more examples of where Copilot could be used.

3.3 Data Collection

Hearkening back to the purpose of this study, the aim is ultimately to assess the impact of Copilot on software developers' work processes and their opinions on the future of their profession from a qualitative perspective. The data collection methods should consequently align with the goals of this thesis. Based on a taxonomy on the degree to which interaction with software engineers is necessary, Singer et al. present a comprehensive list of different data collection methods for different contexts [50]. According to the list, one can form an evaluation by collecting general information and opinions about a process, a product, and even personal knowledge through interviews and questionnaires.

Google Forms was used to distribute the surveys for SA1 that the participants filled out every workday. Questionnaires are mainly applicable when collecting large amounts of data as they are very fast to create, send out, and analyze [51]. They are also a good complement to use together with other methods when gathering data as it gives a broader knowledge basis [51]. Therefore, it was still suitable to use them for the goals of this thesis as they could support the subsequent interviews. The questions of the survey were tailored to address the research questions and can be seen in Appendix A.1, with a majority of them using the *semantic differential scale*. This technique allows the respondent to select a degree of opinion when answering a question using a bipolar adjective scale, usually between one and seven [51] (see Figure 3.2). Each end of the scale host antonym adjectives (e.g. fun - boring), and the respondent select the number on the scale they feel represents their answer. The technique also offers an advantage compared to other techniques such as the Likert scale in that it can be analyzed using qualitative methods. One can analyze ratings to identify patterns or themes, which in turn can provide a deeper understanding of how a concept or object is perceived and evaluated [51].On top of this, some open-ended questions were included as well to allow the participants to provide more detailed accounts and insights as well as other comments.

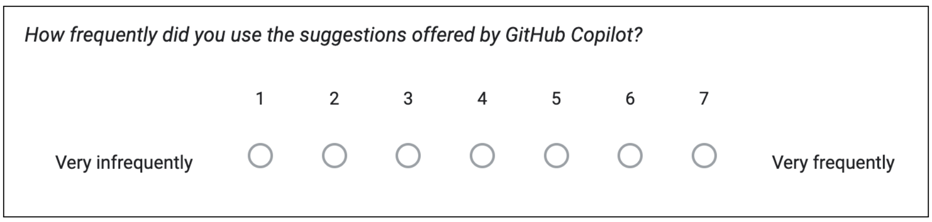

Figure 3.2: An example of a semantic differential scale question from the survey.

The interviews for both alternatives did not differ in structure or technique. Each interview followed a semi-structured format design and lasted around 40 minutes. A more focused set of questions was created for each individual RQ to ensure that the respondent discussed topics within the scope of the study, and the interview questions can be seen in Appendix A.2. However, because of the more free, conversation-like setting that semi-structured interviews provide [50], questions did not necessarily have to be discussed in the order written down in the script and there was an option to ask follow-up questions to get more data on interesting topics brought up by the interviewee. One difference between the interviews conducted with the participants doing SA1 was that the data from the surveys could be used to ask the participants more deeply why they answered the way that they did. The interviews were done either online using various digital meeting tools or on-site at the participant's workplace. Before starting the interview, the participant was first asked for consent so that the interview could be recorded for later transcription. The interviews were recorded with the standard recording app of an iPhone 13. After the interview, the recordings were uploaded to the Google Chrome extension *Transkriptor*, a transcription software that automatically writes out the transcription of the interview from the uploaded recording. The recordings together with the transcriptions were however cross-checked for every interview and polished to remove any errors created by the software.

3.4 Data Analysis

Because of the limited amount of sample size for SA1 specifically, it was deemed to be insufficient to use the data provided by the surveys for meaningful analysis and as something that could be used in the result section as significant data. Instead, the data from the surveys were used in the interviews with the participants doing SA1 as a form of diary for the interviewers to reference and ask the interviewee about data points that were deemed particularly noteworthy during the interviews.

An iterative, inductive approach was used to perform thematic analysis on the transcriptions to identify, analyze and report patterns, or themes, in the data [52]. The iterative process was done in accordance with the phases Braun and Clarke identify [53]:

1. Familiarising yourself with the data: simply reading and re-reading the data, making notes of ideas that spring to mind.
2. Generating initial codes: coding the entire dataset systematically and collating data that is relevant to each code. They define codes as labels that "identify a feature of the data (semantic content or latent) that appears interesting to the analyst".
3. Searching for themes: gathering codes (and related data) into candidate themes for further analysis.
4. Reviewing themes: checking whether the themes work with the data and creating a thematic map of the analysis.
5. Defining and naming themes: refining the themes and the overall narrative iteratively.

6. Producing the report: which will, in turn, require further reflection on the themes, the narrative and the examples used to illustrate themes.

The analysis was done with the help of the software *Atlas.ti*, allowing the user to create a database of documents to highlight and analyze their transcriptions. Firstly, familiarization with the data was made with each transcript being read through thoroughly. Quotes were then highlighted and marked with an initial code relevant to the study's purpose, for example "Efficiency" or "Hindrance". Candidate themes were then created, for example, "Features increasing efficiency", and served as a basis for a reviewing process. The reviewing process was done by producing a code distribution report, provided by Atlas.ti, on the entire project (see Figure 3.3).

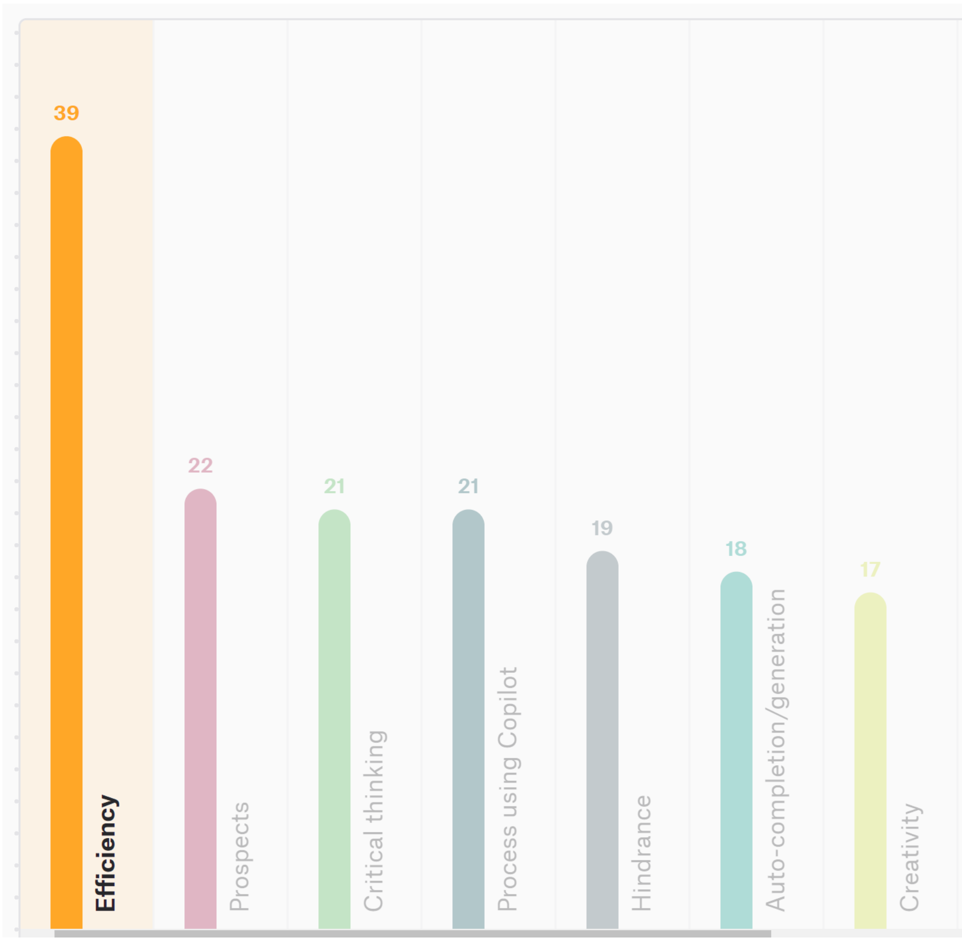

Figure 3.3: A section of the most used codes in the code distribution report provided by Atlas.ti.

The quotes could then more easily be mapped out using the candidate themes and were iteratively analyzed to see if any adjustments needed to be made by adding and removing codes from the quotes. Once new codes could not be formed, the code distribution on the quotes was once again analyzed to form overarching themes based on the candidate themes and the RQs. Some of the candidate themes were turned into categories of the more overarching themes, and a final report could then be produced.

4
Results

This chapter presents the results elicited from the thematic analysis where three themes were created. The themes aim to cover one RQ each and will be presented in their own section, while Tables 4.1, 4.2, and 4.3 below give an overview of the findings. Each theme also consists of multiple categories in which quotations from the interviewees could be divided. Do note that various quotes presented in these findings have been translated from Swedish to English. Furthermore, note that specific participant identifiers such as the participant number have been omitted when presenting the quotes in the results to further insure anonymity to those who have participated. Lastly, while 261 quotes could be given codes in the thematic analysis, only a subset of these will be showcased in the findings to capture the quotes that best summarized the overall sentiments of multiple participants' responses along with quotes that were deemed particularly compelling.

Table 4.1: Showing an overview of the theme "Direct effect on daily work" and its categories with example quotations.

Theme	Category	Examples of supporting quotations
Direct effect on daily work	Features increasing efficiency	"Yes, but this with auto-completion, it goes much faster than if you had written everything yourself. Then as a programmer it is important to understand whether it is right or wrong, what it autocompletes. But yes, it just goes much faster simply."
	Features being a hindrance	"Sometimes you've accidentally tabbed something you don't want, so you have to sit and delete it instead. A small little thing. But when you have used it more, I assume that you also get to know the way of working better. But for example at the beginning also like, you have to wait a little while for the Copilot to think, then you have to sit and (inaudible) for 2 seconds and then it will figure out what to do."
	New or improved features	"...a little console that you can ask questions or something like that. Maybe it would be really nice if Copilot could not only write the code but explain it as well. I know that it can generate the comments, then you're generating the code as well, but it would be really nice to have like a more in-depth explanation of what it's doing, like ChatGPT can actually do it at the moment. So that would be really useful."

Table 4.2: Showing an overview of the theme "AI in the problem-solving process" and its categories with example quotations.

Theme	**Category**	**Examples of supporting quotations**
AI in the problem-solving process	Positive effects	"I Google a lot. I kind of have the editor and Google, and so I just copy back and forth. So often a lot is already copied from something that someone else has done. The difference here was just that I didn't have to google. But it was just there [the suggestion] and then I changed in it."
	Negative effects	"But I also think that maybe it will be a bit that you become unsure of yourself, like you have a plan of where you are going with your code and then it gives a completely different suggestion, that it becomes a bit like an interruption that you start thinking but is this what I should do now or should I, like, what should I do?"
	Critical thinking	"Especially with security issues you really have to be aware that it is able to handle the security. Because if that goes wrong, it's less good than if it's just a regular bug. Then you have to know yourself what security routines are needed to secure, that is, to write this system."

Table 4.3: Showing an overview of the theme "Prospects using AI in SE" and its categories with example quotations.

Theme	Category	Examples of supporting quotations
Prospects using AI in SE	AI tools in education	"It's the same as this about it being so problematic to write articles and essays with ChatGPT. What I think is that in the future, everyone will write bulk works via that type of AI. So it consists of learning how to use the tools that you will use in professional life, for example in school. Same way as the calculator and all the other tools that exists."
	Worries and Questionings	"With Copilot, it really becomes more accessible than it is if you have to Google it too. So is very easy to just take a solution and be a bit lazy. All developers are lazy. We kind of like abstracting things, not having to do things twice. So I think wrong solutions or non-conventional things can stick."
	Career prospects as a software engineer	"Yes, I think it's the same there that you might focus on what you need to move forward. If there is a tool that solves more low-level things for you, then there is less reason to be really good at that. So perhaps for efficiency reasons, I absolutely believe that it can affect the level of competence."

4.1 Direct Effect on Daily Work

This theme was created to represent the findings regarding RQ1 and the three categories include different aspects of the participants' opinions on Copilot's features. RQ1 sought to find out which features might make a developer more efficient and/or hinder them in their daily work, but also what features could be improved or ideas for completely new features, hence the naming of the categories.

4.1.1 Features Increasing Efficiency

All participants emphasized in some way that Copilot could enhance their efficiency in their day-to-day work. When it comes to the specific features that Copilot offers, the efficiency was mostly attributed to the auto-completion of code. When writing a piece of code, Copilot could recognize what one was writing and suggest what would come next based on what had been written before. The user can press the tab key to accept the suggestion.

One participant of SA1 mentioned how Copilot was especially helpful when doing smaller parts of a method.

> *"What's been helpful is primarily when Copilot has determined smaller things for me, i.e. auto-complete. When it sees that I am attempting to write a state variable for example, or a branch in a switch-statement, a branch in an if-statement, those smaller pieces. Then it has been nice not to have to write it, I just tab."*

Five participants mentioned similar sentiments where auto-completion specifically could help one become more efficient, which was when writing more repetitive code. The participants mentioned for example boiler-plate code when building web components from scratch in React or initial configurations for databases. Essentially, writing code when completing tasks that were seen more as chores and did not contain much business logic was where Copilot was seen as the most effective with its auto-completion suggestions.

> *"I think what really can be helpful is features which help to write code, like repetitive code. You maybe don't write it very often, like: OK make an HTTP request, that you always need to Google. Even if you know that yeah, this problem would do the trick, you would still Google just to make sure it's the correct one. So probably for such cases and maybe not where business logic is involved. But, like, from web development side it could be: OK how to center this block? Every time you would Google it. If the Copilot can generate the code yeah that would be really really helpful."*

Eight participants identified new development as the main area of use when examining the specific domains of software development in which the tool could be most useful for efficiency. Participants saw that Copilot could be most effectively utilized when building a program from scratch and also where they could be most efficient with the tool. Out of the three participants in SA1, two said that the tool was most helpful in new development.

> *"It didn't feel like it could improve existing code that much, but it's more giving new suggestions as you are writing. So it's only creating boilerplate code right now anyway. It feels like ChatGPT is more like: how can I improve this function to be more efficient?"*

Conversely, four participants instead saw the benefits of Copilot when creating tests for the code more efficiently. The participants felt that testing was something mundane that could be automated to a certain degree. One participant specifically mentioned how the commenting feature of Copilot, where the tool gives a suggestion of a block of code based on an earlier comment by the user, could potentially make them more efficient in testing.

> *"Something I think is very boring is testing, unit testing, and what I have mainly thought about with these, specifically with these comments and being able to write a test through comments. I think that could really reduce the boring work maybe and receive a bit more help on the way even if [Copilot] doesn't solve all tests. ... It takes a while to set them up and being able to get some rows for free would be nice."*

4.1.2 Features Being a Hindrance

While participants found Copilot to be very effective in helping them complete tasks faster, some also found certain aspects of the tool as more of a hindrance to their day-to-day work. Five participants expressed that giving instructions to Copilot explicitly by commenting directly in code was an inefficient way to work. This was due to a couple of different reasons: (1) Copilot being slow to give suggestions or not giving suggestions at all after one had commented, and (2) the process of commenting felt unnecessary and required too much extra work.

> *"The bulk of the time when I wrote the comment and then tabbed down I got nothing, so I don't know if you have to wait or if [Copilot] was sort of thinking, but yeah. So it was very difficult to determine if I had done something wrong or if it was [Copilot] who had done something wrong, or just wait."*

> *"I think that writing these comments the way that you do, it's nice sometimes, but sometimes when you do it feels like it takes a longer time to come up with how to write them than to just continue writing."*

While the auto-completion suggestions were seen as making one more efficient most of the time, six participants also voiced concerns about how it could hinder them. It was expressed that having a pair programmer with oneself at all times during development can be an efficient way to move forward if one gets stuck or wants an extra opinion, but at times it can become a disturbance that disrupts one's train of thought when inputs come in at inopportune times. It was also mentioned that it could make one insecure about one's work. One participant talked about how Copilot made them insecure when the tool constantly gave suggestions as they already had a plan on what to do next. When the tool gave suggestions it disrupted them and made them think twice about what should be done, and ultimately it took longer time to finish the task. Another participant in SA2 who had used Copilot in their work before had to remove the tool because they felt it brought more hindrance than effectiveness in their work.

> *"Initially I thought this could be good, but then somewhere along the way it became a bit too overused and a bit annoying, so I chose to turn it off myself. It was in the way. … I knew the complete scope in my head and everything and I didn't have to auto-complete the parts that I sat with because all the times I tested [Copilot] I refactored myself anyway. So I could have written correctly from the beginning instead"*

4.1.3 New or Improved Features

While there were some perceived hindrances of the tool, many participants came up with some interesting suggestions on how the tool can be improved by either adding new features or improving existing ones to reduce these hindrances. While this study is not meant to be an analysis of how Copilot should be improved necessarily, it is interesting to see how AI tools in programming can be better adapted for a better developer experience, and how that would affect developers' work.

Eight participants spoke about how they would like Copilot to be an AI assistant that they could communicate with more directly. Many mentioned ChatGPT as an inspiration for how Copilot could be changed in that ChatGPT opened up for more discussion and analysis of existing code rather than just providing new development. According to the participants, this would also improve areas such as maintenance and bug-fixing.

> *"I was playing around with ChatGPT a bit and I really liked that you can really interact with it, like more. It would be nice to see if Copilot could do something like that. Like instead of only suggesting the code that you can write there would be, I don't know, a little console so that you can ask questions or something like that. Maybe, it would be really nice if Copilot could not only write the code but explain it as well. I know that it can generate the comments, when you're generating the code as well, but it would be really nice to have like more in-depth explanation of what it's doing like, ChatGPT can actually do it at the moment, so that would be really useful."*

The idea of a smaller interface outside of the code that not only provides new code suggestions but also discusses the code one has already written was shared by four other participants as well. One participant also predicted that this change could eliminate the hindrance of Copilot disturbing one's thought process in that they are given a choice to interact with the AI, rather than it providing suggestions at all times.

> *"I would like to have more of a discussion platform like ChatGPT and also analysis tools so you can ask it: Why do I get this error message when running? So it can sort of figure it out. Such things would have been really mind-blowing."*

4.2 AI in the Problem-Solving Process

RQ2 aims to answer how developers' problem-solving process could be affected by using AI tools in their work, and the participants reasoned about the topic in different ways. The majority believed that these tools would help them in their process by allowing them to be more creative and efficient, but many also pointed out ways that AI tools could affect their process for the worse. Another key aspect of this theme was the participants' emphasis on being critical when using code suggestions or other outputs from an AI.

4.2.1 Positive Effects

One aspect that participants pointed out was that AI tools can replace steps in their problem-solving process in which they search for information, such as how to write a piece of code. Six participants pointed out that Google is open next to their code editor a lot of the time but that with these AI tools, they could find solutions to tasks in a more efficient way.

> *"I Google a lot. I kind of have the editor and Google, and so I just copy back and forth. So often a lot is already copied from something that someone else has done. The difference here was just that I didn't have to google. But it was just there [the suggestion] and then I changed in it."*

Similar to the findings regarding increasing efficiency, participants described that AI tools could add value when they need to implement functionality that is already existing or solutions to lower-level problems that the AI can solve itself. This could let software engineers solve problems in a different way, focusing more on the end product and less on the implementation details.

> *'It's easy as a developer to want to just write something fancy, but the important thing is how you deliver*

> *value, and there I think that the less time I need to spend on the implementation details, the more room there is for me to be creative in the final product."*

Two participants even referred to programming as being boring, and that with AI tools they could focus more on solving problems at a higher level that were perceived to be more enjoyable.

> *"You don't have to do the boring bits, you get to be part of the fun and build up the system. And then when you actually know that: now I want a function like this, so give me a function like this.*

Furthermore, it was highlighted by five participants that AI tools could increase creativity. One participant for example made a comparison between AI tools and using programming libraries in the sense that it makes programming more abstract rather than less creative.

> *"I think that ChatGPT is rather a sounding board on like we have this problem what would you do, and then you get some ideas and then maybe you just: Ah, that's a way you can do it! Then you test it and you will think of other ideas and you spin on further. So I rather think that it kind of increases creativity because you maybe start from zero and have no idea what to do but you receive some ideas."*

Six participants also mentioned that it is a great starting point for solving problems as it is a source of inspiration, and even if all suggestions do not solve an issue fully they can be tweaked to your liking.

> *"Because in my particular example, I guess it would really enhance my creativity because I learn from seeing the example being done and since GitHub Copilot can actually implement something for you step-by-step or line-by-line, at least for me, that would be really beneficial because I would see a good example and maybe it could spark something in my, I don't know, subconscious and maybe I can remember and learn a few things that I maybe haven't seen before."*

4.2.2 Negative Effects

Although many participants put emphasis on the benefit of not having to use Google and forums such as Stack Overflow as much in their problem-solving process when using AI tools, one participant felt that not reading about and comparing different techniques and solutions online could be a disadvantage. Two other participants reasoned in a similar way and saw a risk of getting lazy in the problem-solving process, i.e. trusting blindly on the AIs' solutions and not considering alternatives.

> *"... when you start relying on ChatGPT's capabilities and what it can do or Copilot and what it can do, I think it's easy to ignore looking for other alternative solutions to any suggestions. And then whether that limits creativity, I don't know, but I think it's generally tricky where you start to trust something far too much that it becomes easy to have a bit of tunnel vision and you might not even think about what you're doing, then hope that it sort of works out".*

4.2.3 Critical Thinking

Even though AI tools can help software engineers in their work it became clear that critical thinking and base knowledge are required when using these tools, both to know how to ask the correct questions to get what you are looking for and to validate the outputs. This was agreed among all participants. Two participants also mentioned that you should be extra aware since Copilot in this case is trained on public repositories. They reasoned that just because Copilot suggests a solution that is most common among all data it is trained on, it is not necessarily the best or correct way to solve the problem

> *"You can always jump into this GitHub Copilot and start making the application without any knowledge. Like you're just writing comments, GitHub is generating code, and then you pray that everything runs fine the first time. But that's not really effective. I guess, yeah, in order to use it effectively, you will still have to understand what it is outputting."*

Another aspect two participants felt needed to be taken into consideration when using AI in the problem-solving process is to be aware of the security risks the outputs can entail. One needs to have a fundamental knowledge of how secure code is written to prevent the blind acceptance of insecure suggestions.

> *"Especially with security issues you really have to be aware that it is able to handle the security. Because if that goes wrong, it's less good than if it's just a regular bug. Then you have to know yourself what security routines are needed to secure, that is, to write this system."*

4.3 Prospects using AI in SE

The final theme includes the participants' thoughts and opinions on the prospects of using AI in SE. This is in connection with RQ3 and will be answered by eliciting results from the two previous themes, but also from the interview questions about how AI tools could change how programming is taught, how the prospects of a software engineer could change in terms of a shift in the role as a developer, and the effect on expertise and newly graduated developers. Finally, worries and questionings raised by the participants regarding the integration of AI will be presented.

4.3.1 AI in Education

While this study does not have students as its subjects, if or how the teaching of programming in schools will be handled differently with the introduction of GAI is relevant for the future of the industry. 12 participants had a thought process that was positive about the introduction of GAI into SE education.

> *"I don't think ban. I absolutely don't think so. You will always have [AI-tools] in your working life later. It is more about learning how to work with them"*

One participant interestingly also brought up the example of the introduction of the calculator and used it to justify that AI will impact education in a similar fashion.

> *"Yes, it certainly will. I think it will have a big impact. I think it will be a bit like the calculator, you won't have to be able to calculate everything yourself and you just can figure out, like overall how it should work and then you let calculators do the heavy lifting."*

The effectiveness one can increase when using these tools could play a role in school settings as well. Five participants foresaw scenarios where teachers could move on more quickly from the more technical concepts such as syntax and computer architecture, as that will be provided by the AI. Teachers can instead focus on discussing higher levels of abstraction such as the infrastructures of building programs and best practices in SE.

> *"... where I am from, we have lots of people now changing their careers from a totally different area to IT and I've noticed that in those courses they are taught like, okay, how to develop in C# and JavaScript, etc, but maybe lack, like, architecture best practices or like what's best practices of software development in general? What are restful APIs? Something like that. So yeah, using this Copilot or other tools would give more time for other topics, so then it would be really really*

> *nice. So I think yeah definitely there could be a bit of change in the academic setting, how we teach those things"*

In general, it was hard for participants to predict what would have to happen in regard to testing and assessment if AI assistants are incorporated into a school setting. The only real conclusions that participants could think of were: (1) having tests where these tools were not allowed, or (2) having assignments and tests where it is known that these tools can be used, and formulating the questions around that.

Outside of more formal education, four participants saw the potential of GAI tools in enhancing one's personal learning. Emphasis was put on the value of practical experience for obtaining competency in all aspects of SE, such as lower-level programming and software architecture. It was thereby, according to the participants, important to view the implementation of theoretical concepts, the experimentation of new languages and frameworks, as well the handling of bugs and errors as individual learning opportunities. These opportunities could then be learned even faster when integrating GAI tools into one's workflow.

> *"You don't always have Google giving all of the best examples or answers, and I think that would be, I don't know, quite a nice tool to learn some of this stuff. Especially if you're, I don't know, learning a new programming language or a new concept. As I've said, again, it would be a nice example like teaching the concrete, I don't know, piece of codes, so I guess that would be useful."*

4.3.2 Career Prospects as a Software Engineer

Inquiries were also made about how these AI tools such as Copilot and ChatGPT can potentially alter the careers of software engineers. Specifically, if it will become easier or harder to enter the profession, if the skill requirements of a software engineer will change, and if there would be any effect on the level of expertise within the industry.

There were some mixed opinions by the participants regarding entering the profession. Nine participants felt that it would become easier as people without any prior knowledge would have more tools at their disposal to potentially learn programming from. In general, participants felt that it has become easier and easier over the years to become a programmer as so much has become abstracted and there exists so many free learning tools online. The introduction of AI tools was considered the next level of abstraction that could lower the entry barrier into the programming world even further.

> *"Yes, it can probably lower the requirements [to enter the profession] in the future because it will be easier to train people and it is going to be easier for people to learn things because you receive help at all times with the more basic things."*

One thing that a participant mentioned that was deemed important to highlight was that one should be transparent with one's use of AI to companies recruiting you.

> *"I think that might be easier, but I think you would have to be transparent about that. For example, when you're learning new stuff there's like a few ways to do that. You can, I don't know, read books, take a look at YouTube, and I believe Copilot is, would be one of the learning paths. For example, if you're applying to a company that requires you to do a homework task. For example, make a small application, I guess you would have to be transparent about it that you have used Copilot if you use it. Because I guess it really shows if you have knowledge about your code or if you have actually, I don't know, written everything with Copilot prompts and then you couldn't explain what's in your program."*

Four participants said that it would be more difficult to enter the profession. While the individuals who felt this way mainly agreed that learning programming would be simpler, they followed their thought process in a different direction and considered the repercussions of programming being an easier discipline. One school of thinking is that you will not need as many programmers since fewer individuals with all of these new AI tools can do the same amount of work as previously in larger teams, reducing the number of jobs needed and making it more difficult to find work.

> *"Like from a learning perspective like if a new person is learning something and gets stuck, yeah, maybe he or she can ask Copilot to create things. So from a learning perspective, yeah you might learn things even faster, understand them faster. But if like there's a lot of such developers who learn it this way, so there might like be more tricky to get into the market"*

Eight participants felt that there would be a shift in the requirements in the future and that the title "Software Engineer" would change meaning in terms of what is needed to perform the profession. This sentiment mostly centered around how AI tools can perform more low-level tasks at a more efficient level, which in turn can shift the focus to solving problems on a higher and more abstract level. It was also mentioned that knowing how to control your AI tool and effectively writing prompts will become a required skill in the future. Two participants even imagined there could be a specific title for a profession as an “AI prompt engineer”.

> *“I really think this is just the next step in programming. Since I started, I have always raised the level of abstraction, that a line I write today, how many computer instructions isn’t that? I have no idea. So, and there are certainly people who say that you don’t even know what your code does, because you don’t understand which ones and zeroes this really becomes, this is the same thing. I’m super abstract now. The language solves this for me, the details. And somehow, that’s how I see my career developing. That you become more and more abstract where this is just a giant shortcut to it, that we can all become more tech leads, than that you have 50% of the workforce who are junior people who sit on Stackoverflow and try to google implementation details in the language they working with. Maybe this tool can make us realize, the same way we’ve realized with our programming languages, it’s okay to abstract and we can all level up and have greater impact that way”*

> *“Then, I can probably imagine it in the future. Certainly not now, but in the future someone will surely write: you must know... you must be good at asking an AI. Now I don’t know, there’s probably nothing there right now, but there will probably be courses and workshops on how to ask the question to AI, and that will be what determines how well an AI can help you in the end.”*

When looking at the level of expertise in the industry being affected, there was a clearer hesitance in predicting how or where the AI tools could affect it from the participants, at least compared to the answers about the knowledge requirements and entering the profession. Many commented on how the question was complex and had a hard time formulating a concrete answer. One participant commented that they could see themselves answering that question in 10 years when these tools have had a longer effect on the industry. More concretely, five participants believed that, for the time being, the level of expertise would not be affected. However, newer developers would learn at a faster rate. In other words, the knowledge floor would be increased, but not the ceiling.

> *“More people could, yeah, actually get up to a certain level faster. And the people that already know stuff, it wouldn’t really have that big of an effect. Like it still would be like a nice thing to have, but it’s not going to be groundbreaking.”*

> *"Difficult question, I don't think the skill level is that great, to begin with. I don't know, those who are good are good and those who are less good can get better from this. Those who are very good, don't need these kinds of tools. You are pulling people up to a fairly standard level or a basic standard level anyway."*

4.3.3 Worries and Questionings

Even though GAI in SE has become a recent topic of discussion it was evident from some responses that it remains a type of technology that raises a lot of questions as it is still in the early stages of widespread utilization and development. One concern that two participants raised was the ramifications of how data is collected for training the AI model. More specifically, the risk of company secrets being used as training data, potentially leaking sensitive information to other users of the AI outside of their organization.

> *"Definitely, something that companies may not take into account today, but the data that you send into Copilot and in GPT is used to train the AI. If you don't think about it, your using the companies code. Then Github might do it anyway because they have access to everything. But they just use your code and see your code when you type keys and everything and that's actually very dangerous. So I know that some who have used ChatGPT have to strip all indications of what this feature is about, who it belongs to and things like that. Because I don't know if there's any policy in I think most companies about this. But there can be consequences if you leak code."*

Four participants raised questions on a similar topic of licensing, i.e. what could happen if you use code without the correct license. There were also worries about who actually owns the code generated by an AI and whose fault it is if there is a breach of information or plagiarism in the code.

> *"It's really interesting who owns something that an AI has generated. If you have an idea on paper, like you can give it: I would like that from this data we can predict earthquakes. Whatever it might be and feed it into this and then it finds the golden sort of algorithm for this. Whose idea or who has actually written this actually? So there are definitely such difficulties. But yes, it's probably such litigation, I don't have a great grasp*

> *of it, but that can definitely become a problem. ... and whose fault is it also like, it's also legal. But if we say that Copilot solves a whole API to fetch, use data and stuff and there's a breach there, whose fault is it?"*

Two participants mentioned the risk of an AI being biased, that is the risk of the AI giving discriminatory outputs.

> *"So, I've heard how AI is developing and that they, they take after humanity and humanity is not ethically correct all the time. So, but I'm... I don't know... I'm too poorly informed about AI in general I think to discuss it too much. It's because you've heard that there are AI bots that sort of become racist etc.. So it's a risk"*

Two participants emphasized the necessity of clearer regulation in how to develop AI in a safe way.

> *"Yes, regulation is probably needed in that case externally because otherwise the companies will probably continue to develop them because there is money in it. So it is probably necessary globally to take care of those issues in order for it to happen safely."*

In contrast to the previous responses, one participant did not express worry and stated that the issues of GAI will be resolved in the near future.

> *"... what rights will I have? Who only gives it prompts and then it outputs it. Am I the one that owns the output? Those types of things are probably an issue. But we'll get a standard there. After all, this is not something that our children will think about, but it's just the new world."*

In some interviews, the topic of how much AI can replace human work was discussed. The findings from previous parts of the result indicate that most participants believe that there will be a shift in work practices as a result of using AI tools. It can take care of problem-solving on a lower level and allow software engineers to be more efficient and solve problems on a higher level. However, three participants highlighted the importance of human interaction and that this is something that can not be replaced by AI. One outlined a situation in which a single coder would not be able to solve all the integration between multiple teams collaborating on one product or service only using AI tools.

> *"I think within development it's still, the infrastruc-*

ture and similar aspects are so complicated that it still requires people who make decisions."

"But an event-based system that requires setup of infrastructure in two different places, [GAI] does not solve this whole part. Especially if we are talking in a large organization where you have 40 teams, each team sort of has its own like ownership of their part that they code with. How can one as a single coder be able to solve that problem only with ChatGPT that will integrate with three other systems in that organization? It is absolutely impossible because you have to talk for it to work, you have to ask people if it's going to work. So what I think regardless of how good GPT becomes, you can only take it so far. Because in the end, human interaction is still required."

5 Discussion

In this chapter, analysis and evaluation of the results will be made to attempt to answer the three RQs. While the study primarily investigated Copilot as a tool to answer these, a more generalized discussion on GAI tools in programming will also be made due to the similarities and interesting discussions developed during the interviews. Firstly, a discussion on participation hesitance for SA1 will be made. The three RQs will then be discussed and answered based on the findings and relevant literature, followed by a section discussing whether the responses from the interviews differed between participants in SA1 and SA2. Furthermore, a deeper dive into the significance for researchers and implications for practitioners will be explored. Finally, a discussion surrounding the study's validity threats and where research can be taken further will be made.

5.1 Hesitance of Participation

As discussed in Section 3.2.2, another alternative for participation in the study was created to get enough interviews to collect qualitative data. It was noticed early on in the recruitment process that companies were hesitant to allow their employees to take part in installing Copilot on their work devices and use the tool while doing programming tasks. The main reason for this hesitance was privacy, something that has been a problem for Copilot as has been presented in Section 2.4.1. Many did not like the idea of GitHub, and by extension Microsoft, having access to their code and it being able to be used as training data. Even though a setting could be turned off so GitHub would not use code snippets for product improvement, it can still be unclear exactly what data would be used and for what purpose.

While the problems of Copilot were known to the authors before undergoing the recruitment process, the degree to which the industry hesitated to use the tool in a professional setting was unexpected. Focusing on the engineers using GAI and how it would affect them and their outlook on their industry is, to the best of the authors of this thesis knowledge, a novel study. This made it hard to predict how the industry would react to it. Larger companies mentioned privacy as the major reason for their hesitance but from the background research, it can be argued that regulation is the larger issue. The commercialization of GAI is such a new phenomenon in the world that there has not been enough time for regulatory bodies to form laws and guidelines on how to be transparent in developing GAI and making sure the privacy of its users is upheld. How their models handle data, what data is being trained on, and so forth has to be more clear for the companies to trust in these technologies, at least in a real work environment.

For this study, an alternative solution to fulfill the purpose and answer the RQs could still be made. Luckily, the main data-gathering elements did not have to be altered much which meant that the study could go on with the same timeline as planned. However, due to this change, most participants did not actually use Copilot in their work and were only shown a demo, which does affect their experience with the tool. It also affects how one can generalize the results, which will be discussed in further detail in Section 5.8.

5.2 Programming with Copilot

RQ1: *What features of the tool help programmers be more efficient in their work and conversely what features hinder them?*

Compared to previous studies [12] [41] [42] [43] which focused more on the quality of the code suggested by Copilot, this discussion will focus more on the participants' experience when using the tool. In addition to discussing Copilot's features' effect on efficiency and how they could be a hindrance for programmers, the other major topic of discussion in the interviews was how Copilot could be improved, with adding conversational AI being the largest feature request.

5.2.1 Effect on Programmer's Efficiency

It was found that Copilot can increase a developer's efficiency, and this is mainly motivated by the auto-completion feature of Copilot which predicts the next line of code or complete functions, letting the user accept suggestions with a press of a button instead of having to type the whole code snippet themselves. It was described as significantly useful when writing repetitive code which occurs in many places but differs little in functionality, or when the code did not contain a lot of business logic. This makes the feature most valuable for new development since this is a part of development when code is written from scratch, as compared to maintenance or bug fixing where work is done on already existing code. Reflecting on the discussions had with the participants who had used this feature, one conclusion is that the tool can produce code faster but often with worse quality than a human programmer. As a result, if a developer can use Copilot and alter the suggestions to increase the quality to what is expected faster compared to writing the same quality code without the tool, it certainly makes developers more efficient. Conversely, if suggestions are accepted without further tweaking, it is likely that bugs occur, tests fail, and the overall code review will take more time. This also aligns with the results of previous studies on the topic presented in Section 2.5, as Imai found that code produced by Copilot had worse quality compared to a human pair-programmer [12], and Vaithilingam et al. found that participants using Copilot failed to complete three more tasks compared to participants using Intellisense [43]. In conclusion, it is still the developers' coding knowledge combined with utilizing the tool that will affect how much it impacts efficiency.

The other main feature of Copilot which gives suggestions after the user asks for something in natural language in a comment had divided opinions among the participants. While Chen et al. [42] found that Copilot's ability to solve a chain of prompts was worse compared to a human programmer, the findings in this study were more related to the developer's experience. The overall feeling was that the feature requires enhancement to make the integration into the developer's workflow more natural because as of now, it cannot really separate between what is a comment for documentation and what is a comment asking Copilot for suggestions. Furthermore, when a suggestion has been accepted by the developer, the comment is not removed. This could eventually clutter the file with prompts that the developer later has to remove one by one. Another hindrance was regarding the auto-completion as participants pointed out that most of the time it increased their efficiency, but on some occasions, it could be a disturbance when writing code. When Copilot makes suggestions all the time it can remove focus from the developer's own thought process. When the developer already knows what to write or has an idea of how to move forward, suggestions that pop up on the screen can disrupt and bring insecurities, potentially decreasing efficiency. Compared to the commenting feature where the developer actively asks for suggestions, the auto-completion is always turned on. To disable it, one must turn off the Copilot plugin completely which could lead to disruptions in the workflow.

5.2.2 Drawing Inspiration from Conversational AI

When asking the participants how Copilot could be improved, either by enhancing existing features or suggesting new ones, there was a strong pattern that many appreciated the dialog you can have with ChatGPT and therefore would like to see something similar in Copilot. The value this would bring is that developers can ask questions to better understand the code as the AI can give explanations in terms of documentation and examples. This would make developers more aware of why a certain suggestion was given by the tool, and the ability to give follow-up prompts such as questions or instructions to add/remove functionality of the code suggestion. This was also highlighted in the result of the study by Ross et al. [44], who developed a chat assistant powered by the Codex model. They also found that the tool's suggestions were not always correct, but that the participants appreciated the conversational approach since the tool could explain and document code via the chat interface. Participants in the study conducted by Vaithilingam et al. [43] also emphasized the need for improving Copilot to make it easier to validate and understand the generated code.

Combining this study's findings with results from previous studies, it can be stated with confidence that bringing a conversational feature into Copilot could resolve many of the participant's requests. Examples of such requests were being able to highlight code and asking Copilot to write tests. Additionally, it would allow Copilot to explain a block of code or error messages, which code be especially useful for programmers learning a new language or working with legacy code. Furthermore, as one of Copilot's hindrances was that it could remove focus from the programmer and potentially make them more insecure due to the suggestions from the auto-completion, that feature could be turned off and the chat dialog could be accessed when the programmer is in need of assistance. In addition, asking for functionality in natural language using a chat dialog as compared to writing comments directly in the code file would remove any clutter.

During the time this study was carried out, GitHub have taken note of this requested feature. GitHub introduced an enhanced version of their tool called "GitHub Copilot X" with the X representing different aspects of the service, such as Copilot *chat*, *pull requests*, *docs*, and *CLI*. As it is still in its testing phase and yet to be released to the public, GitHub aims to bring many of the requested features by the participants in this study. The chat feature will be implemented as a part of VSCode, allowing developers to interact with it more seamlessly as a part of their workflow compared to having ChatGPT open in the browser. Features such as highlighting a piece of code and asking Copilot to explain it, fixing a bug, or generating unit tests are some examples of the tool's capabilities.

5.3 An Optimized Problem-Solving Process

RQ2: *To what extent can the tool affect programmers' problem-solving process?*

Analysis of the results indicates that software engineers can utilize AI tools in their work to solve problems in a different way compared to not using these kinds of tools. Discussions in the interviews mainly revolved around a shift in how problems could be solved using GAI, and that a fundamental understanding of programming is required to best utilize these tools.

5.3.1 Higher-Level Problem Solving and the Effect on Creativity

Copilot and similar GAI tools such as ChatGPT can be used as sources of inspiration when programming and a shortcut for whole implementations for lower-level problems. This replaces steps that software engineers otherwise would have to take, such as using Google or forums like Stack Overflow to seek information on how to implement certain functionality. A significant advantage of using AI tools is that software engineers can work more effectively, allowing them to spend more time solving problems on a higher level rather than having to deal with simple and repetitive programming, which some of the participants referred to as tedious tasks. As an example, this could lead to software teams delivering products of higher quality because there will be more time for the team members to focus on the overall system architecture, rather than implementing basic functionality, writing tests, or fixing bugs. This was also seen by Ross et al. [44], where the study's participants felt that the tool could assist in solving lower-level tasks which in turn would let them focus more on higher-level development.

Another topic discussed in the interviews was if the participants thought that using Copilot or other AI tools would increase or limit their creativity in their problem-solving process. The general impression was that these tools could help programmers be more creative as multiple participants emphasized that creativity lies in the end product, and not in the code itself. Copilot's features can spark ideas by suggesting code one might not have thought of previously, and ChatGPT can take this further by effectively generating multiple different suggestions based on the programmer's needs. It can be especially useful when a programmer has an idea of how he/she wants to implement for example a user interface in their mind, but having difficulties turning their vision into code. Rather than having to implement every individual component from scratch, GAI can assist in the process by generating code for components that can be tweaked and put together to fit the programmer's vision. As GAI tools become more advanced through further development in the following years, a possibility is that they will be able to generate more complex solutions such as generating complete modules based on software requirement specification documents.

On the other hand, it was also seen that continuous over-reliance on GAI when solving problems could potentially limit the knowledge software engineers gain as a result of relying too much on these tools, rather than reading documentation and comparing different solutions themselves. This sentiment overlaps with the discussion Danaher [37] had towards balancing the utilization of AI while continuing the development of one's own cognitive abilities. Developers and people with other roles in the industry come across different obstacles every day at work and some participants believed it is of great intrinsic value to try solving problems on their own before using the first suggestion given by a tool. This would allow them to better understand the code which would also make them better at their profession, knowing what actions to take in similar situations in the future.

5.3.2 Retain Base Knowledge and Critical Thinking

The participants in this study did however not feel that using GAI in their work would limit their problem-solving capabilities in general if they are used in the correct way, and could rather increase their creativity and assist in solving problems faster by being a source of inspiration. Using the tools in the "correct way" implies that you are critical of the suggested solutions and that you have the base knowledge to understand what the code is doing so it can be tweaked to suit the specific problem you are dealing with. One concern when lacking base knowledge is that security vulnerabilities can be introduced in the program which could bring serious harm if ignored or not taken care of due to incompetence. Chen et al. also recognized this risk in their study [42] and referred to it as over-reliance, i.e., that Codex can suggest solutions that appear correct but do not perform the asked task. Another consequence, if software engineers are lacking base knowledge, is that it can make them more ineffective in their problem-solving process because they are unsure how to inquire correct questions or prompts to the AI to get the outputs they are seeking. On a similar note, they must be able to put the given outputs into the correct context and alter suggestions since they often do not fit the context fully. An additional issue that might arise as these tools become more sophisticated is that software engineers become "lazy" in the sense that base knowledge could erode over time. When these tasks could much more effectively be solved by AI, it leaves them no choice but to accept the output without much thought. Danaher discusses something similar to the degeneration argument, where memory and understanding can be reduced by using AI over a longer period [37]. A counter to this is simply to not use the AI tool when learning new concepts to stimulate one's cognitive ability and learning. This holds especially true for students, engineers on a more junior level, and in general when learning new concepts within SE. If software engineers are aware of these elements and use Copilot and similar services as a tool rather than trusting suggestions blindly, along with continuing to stimulate their cognitive ability by applying concepts practically without the over-reliance on AI in the context of learning, they have the power to make themselves more efficient in their problem-solving process and the potential to reshape what the process looks like in the future.

5.4 Prospects of SE

RQ3: *Can the potential possibilities and/or hurdles introduced by the tool change the prospects of software engineering as a profession?*

While RQ1 and RQ2 focus more on the individual level, RQ3 focuses more on the macro level, exploring the prospects of the profession and thereby the industry with the introduction of not only Copilot specifically, but other GAI tools for programming as well. This will be evaluated based on the answers to the preceding RQs, answers from the interviews focusing on prospects, as well as relevant background research made for this thesis.

5.4.1 A Saturated Industry

Based on the results, a clear majority of participants believed that software engineering would have a lower barrier of entry to a junior level. More simply put, **it will be easier to become a software engineer**. While this was mostly attributed to the further abstraction of programming GAI would introduce, one could argue based on the findings that the increased effectiveness and opportunity for education could affect this as well. GAI can help more people learn the basics of different languages and frameworks and apply that knowledge more efficiently.

With a lower barrier of entry, a logical thought process would be that the industry could experience a saturation of workers. As we live in a digitalized age, having more people able to enter the field can have several advantages for the industry. While the findings did not produce any clear advantages for more people in the market, some disadvantages could be identified. As these tools make lower-level tasks easier and quicker to complete, as well as optimizing workflows, there would be less need for larger teams working on projects. In other words, the perceived value of a software engineer, as we see it today, could be decreased. This can have several consequences, especially with a saturated workforce in that salaries might decrease and there will be more competition in the job market. One can also connect disadvantages to the findings regarding the expertise level in the industry. With proportionally less amount of expert engineers, there is a possibility that software systems will have a lower quality in the future.

5.4.2 Metamorphosis of the Software Engineer

The introduction of GAI has the potential to significantly impact the profession of software engineering and the industry as a whole. What becomes important will then be how to handle the effects of that introduction, i.e., how a software engineer can maximize the advantages and mitigate/minimize the disadvantages. Based on the findings, it is clear that **the role of a software engineer would need to undergo a form of metamorphosis and be redefined in a way**. What is needed of a software engineer today will not be the same as in 10 years for example. This is not something that is new within the profession. A programmer 60 years ago, or even 10 years ago, did not do the same things a programmer does today due to the abstractions of technologies, such as hosting software, cloud engineering, as well as new more declarative languages and frameworks. GAI would simply be the next level of abstraction.

The findings indicate that software engineers in the developer role would very likely have to shift focus from implementing basic functionality to being more involved in the implementation of higher-level tasks such as system architecture. This coincides with the answers elicited for RQ2, where many mentioned how Copilot can reduce the effort needed for lower-level tasks that are more repetitive and mundane. Because of this shift, greater emphasis on other, less technical, or softer skills could be focused on more.

Technical skills can not be fully discarded, however. The importance of critical thinking, which was elicited from RQ2, is something to keep in mind here. GAI tools are just that, tools, and are still reliant on human developers to make all the final decisions. GitHub highlights this argument as well in their FAQ (see Section 2.4.1). As touched on in Section 5.3, participants stressed that base knowledge of software development is still needed to be able to think critically and ask the most appropriate questions to the AI for their context. While retaining base knowledge, the focus on technical skills could then be put on developing skills within for example software architecture and infrastructure instead of implementation details.

One interesting development that the results produced was a discussion of a new form of SE in which one could integrate GAI tools into the profession's job requirements. It was found through the findings from RQ1 that many engineers preferred a more conversational interface, which for example ChatGPT provides. When discussing the prospects of the requirements needed for a software engineer, one mentioned how there will be workshops or courses on how to interact with an AI to more effectively produce higher-quality solutions. If used correctly, people could train on how to construct the best questions or instructions for the AI so that it can deliver the best suggestions or analysis for the given context. This could give rise to a new role altogether in the industry. An *"AI prompt engineer"* of sorts, where the primary requirement of the job would be to prompt and interact with the AI. As discussed in Section 5.3, the majority of participants felt that these tools could enhance creativity and give inspiration for new ideas, so the focus for this new role would be on the creative aspects as well as taking advantage of increased efficiency.

5.4.3 Integration of Generative AI

Concerning the discussions surrounding the worries and questionings around Copilot and GAI, the findings suggest that **there still exist many different concerns over its use today from professional software engineers**. Most of these concerns were about legal and ethical issues. While the level of perceived severity varied from participant to participant, the overarching theme here is that there still is some hesitance to integrate the tools into professional work. In the same vein as the discussion had around research with GAI (see Section 5.6), there is a sense of "wait and see" here on how regulations and ethical guidelines will be developed over time before one can trust the use of these tools in production.

It will also be interesting to see the development of how schools handle the possible integration of GAI tools in education. It can be concluded from both the findings and background research that schools should integrate the tools. An interesting caveat here however is the expressed importance of having base knowledge to be able to think critically about the output of GAI tools in programming. In conclusion, students studying SE can not blindly take the answers provided by GAI without understanding what is being outputted. They still need to be tested in a way that requires knowledge beyond what GAI can provide. How this integration can be done in education is however beyond the scope of this thesis.

5.5 Differences in Responses of SA1 and SA2

In both the results and the discussion sections of this thesis, the answers to SA1 and SA2 have not been separated. While this was an initial plan for the presentation of the findings and a potential point of exploration, the findings from the interviews made it apparent that the way participants responded did not in a significant way proportionally differ in sentiment based on which alternative they participated in.

The process of recruiting participants did not take the experience of using Copilot and similar GAI tools into account. However, given the recent emergence of GAI in programming, every participant possessed at least basic awareness of the topic. Thus, participants were capable of forming opinions and perspectives about how both Copilot and other GAI tools could affect their work, along with their implications on the profession. It is also possible that the demo used in SA2 was able to sufficiently show the capabilities of Copilot for participants to form these opinions, even if they had limited experience with the tool themselves.

One area where the answers could differ in nuance and depth was on the interview questions associated with RQ1. Participants in SA1 could always give more concrete examples of specific instances in code where Copilot helped them be more efficient or hindered them in their work. The surveys could also be used here to ask follow-up questions to gain further insight into the examples that they provided in the interviews.

In contrast, RQ2 and RQ3 did not have the same differences in nuance and responses were similar in depth for participants in SA1 and SA2. The questions pertaining to these RQs in the interviews focused on broader aspects that did not require direct experience with the tool. There was less of a requirement for more concrete code examples, and the questions were designed to elicit broader views on processes and effects outside of actual programming tasks, allowing the responses to be more easily aligned in sentiment.

5.6 Implications for Research with GAI

The identified issues of GAI in programming, that being privacy, security, legal, and ethical, present a high risk for SE companies if they were to integrate these technologies, thereby limiting the possibility to conduct studies on these technologies in natural settings at the time of writing this thesis. Due to the security and privacy concerns brought forth in Section 2.4.1 by Pearce et al. [11], as well as the statements by GitHub itself [9], it is evident that the code that is generated cannot be fully trusted for production purposes. Furthermore, given the current absence of clear legal regulations or ethical guidelines to be agreed upon for GAI, additional uncertainty from potential participants can be seen. The issues of copyright infringement and licensing, as well as the questions surrounding how one should interact with and develop AI in an ethical way, discussed in Section 2.4, further exemplify where the legality and AI usage currently lies in a gray zone.

For researchers, the purpose of this study was to contribute to the knowledge base of the issue of GAI utilized in natural settings. While this could not be achieved to its full potential in this study itself, the hope is that the insights learned from this thesis can provide researchers with a valuable starting point on how to, or how not to, approach related research endeavors while navigating the aforementioned challenges. One imperative to recommend based on SA1 is to uphold active communication and transparency regarding these issues during the study to engender trust with the participants. The initial presentations done in SA1 with participants where these issues were highlighted provided them with an initial understanding of the impact and limitations of the code generated by Copilot.

Another recommendation could be made surrounding collaborative efforts with the industry, something that was considered one of the bigger challenges of conducting the original study design for this thesis. Not having designated industry partners made it difficult to know how to align the theoretical research goals to practical applications, especially considering the fact that this type of study had not been conducted before. Having an industry partner could have alleviated this by more clearly defining the objectives of the study collaboratively so that they can be applied in a natural setting while mitigating the concerns of GAI. One example of how this could be done if the study were to have an industry partner would be to have software engineers work on a new smaller-scale project to be solely used within the organization itself while doing the SA1 design. This way, experiences and thoughts can still be elicited, and all the code that is being generated can more easily be controlled by review processes such as code reviews, ensuring that the issues are mitigated.

Finally, while the design of SA2 was not an established one, combining elements of a judgment study and experimental simulation, along with an artifact-driven interview and demo sessions did not compromise the qualitative aspects of the study. This method could still effectively gather thoughts and experiences around the artifact, that being Copilot in this context. Therefore, the study design can serve as precedence for future research for these GAI tools in SE, provided that the need for data gathered in natural settings is minimal or unnecessary.

It is important to acknowledge however that these recommendations for research with GAI in SE are based on implications that can be classified as current or accidental. These implications were not something that was expected or specifically desired when first undertaking the study, but are a factor of the current status quo of research endeavors involving GAI programming tools utilized in natural settings. As GAI is a quickly developing area of technology, many implications mentioned here may not be applicable in the future. As the issues of GAI continue to be discussed and expectantly reach resolutions, it could become increasingly viable to conduct natural setting studies with GAI as well. As presented in Section 2.4.2, the EU is working on solutions for AI regulation, and lawsuits such as the one mentioned in Section 2.4.1 can provide judicial significance for future disputes regarding copyright for AI-generated code. Furthermore, companies developing GAI systems such as OpenAI should follow frameworks to uphold the standard of ethical AI development. Lastly, the meteoric rise of ChatGPT and other GAI tools may exacerbate the need for heightened awareness of the legal and ethical ramifications of such technologies even further as they become more commercialized.

5.7 Implications for Practitioners

Practitioners can be divided into three groups: AI developers, SE organizations, and individual software engineers. The implications will thereby be looked at from these three perspectives. For organizations developing AI technology, the recruitment process made it apparent that larger SE organizations are aware of the issues that these tools possess. Since protecting their code from third-party sources, upholding licensing standards, and not introducing vulnerabilities in their code are paramount for these organizations, they can not openly use or trust these tools. Therefore, AI developers must address these issues. One solution that could be developed specifically for companies is to create a designated version of the AI that comes in the form of a centralized instance hosted by the company's own servers. This isolates the data within the company itself, thereby removing privacy and security concerns. On top of this, frameworks for developing ethical AI and collaborative efforts with regulatory bodies to clearly map out the legality of these technologies should be made. Furthermore, a current/accidental finding from the interviews that can be applied to AI developers, specifically OpenAI and GitHub, was the request to develop a more conversational interface for Copilot. However, as mentioned previously, GitHub is already planning on fixing these issues with Copilot X.

For other SE organizations, the results suggest that while these GAI tools could potentially be integrated for its engineers to use when developing software, they can not completely replace the roles of software engineers. The potential vulnerability the AI could introduce is an example of a scenario where a human reviewer needs to be present to check that the code is secure and fits the context of what is being developed. Human interaction in larger projects is also still needed as these technologies are not sophisticated enough to connect different components. This was apparent from both the development of the demo used for SA2 and the responses. Taking the development of the demo as a more concrete example, while the AI tool could give general ideas of how to integrate the database, connecting the business logic of precisely what and how the data was to be fetched from and added to the database could not be accomplished by only using Copilot, and adjustments had to be made from the initial suggestions provided by it. This further implies that these technologies are merely tools for enhancing human work and that workers cannot be completely replaced. The results also imply that GAI tools could conversely enhance creativity, innovation, and collaboration within an organization by providing points of inspiration and optimized workflows. The recommendation for organizations would then be to maximize the utilization of GAI tools in tandem with their developers by actively educating how to use these tools in a safe and correct way through workshops or providing access to courses surrounding these topics.

One can summarize the discussion surrounding the RQs to form implications on how Copilot, and by extension similar GAI tools in programming, can quickly impact a software engineer's work on an individual level. Firstly, the compelling aspects of increased efficiency in new development, enhanced creativity, and inspiration that come with the use of GAI imply that engineers may be expected to develop minimal viable products at a much faster rate. In turn, engineers must maneuver the pressure of having to push digital products to market considerably more quickly. This environment adds more stress and strain on the profession. Secondly, the findings suggest a more fundamental implication tied with the introduction of more abstract technology, which can allow software engineers to adjust their focus to solely solving tasks on a higher level. This is something seen over the course of time in the SE domain and has led to improvement in aspects such as learning curves and optimized development processes. This coincides with the findings and can thereby hold true with the introduction of GAI in SE. An aspect to consider here however is the ease of integrating such technologies to be compatible with the current software engineer while also navigating the GAI systems limitations. As discussed in Section 5.4.2, there will have to be an adjustment period for software engineers and potential changes in roles, for example the AI prompt engineering role. This can cause initial disruptions in the workflow through for example the hindrances experienced by the participants and thereby delays in development.

5.8 Threats to Validity

Even if anonymity was guaranteed to the extent it can be, there is always a potential risk that participants altered their answers to try to meet the expectations of the study. As an example, they might have had a more positive attitude toward the use of Copilot compared to if they had used the tool outside of the study, and as a result highlighting the positive experiences. To mitigate this risk, emphasis was put on having a natural experience with the tool for SA1 without any requirements on how the participant should use Copilot in their work. That is, trying it out but using it in their workflow as they normally would, and they could turn it off if they felt that Copilot was disturbing. Furthermore, the questions in the interview were created to discuss both positive and negative experiences of the tool, rather than only asking about their overall experience.

Since participants in SA2 were shown a demo of Copilot's features and did not use the tool themselves, although some participants had used it on their own outside of the study, they might have lacked sufficient experience and understanding of the tool to give elaborate answers regarding RQ1. RQ1 was more focused on Copilot compared to RQ2 and RQ3, which could be answered by discussing similar GAI tools such as ChatGPT, which all participants had experience with. However, combining the knowledge gained from reviewing previous studies investigating the efficiency of Copilot's features with the results of this study, an established conclusion could be made to answer RQ1.

Regarding the sample size with 13 participants being part of the study, it was a slightly smaller size than originally desired which aimed to include 15 participants. The relatively small sample size combined with not using an established sampling method could affect the generalizability of the findings. However, the depth of responses gathered by both spending more time on a lower number of participants and employing the semi-structured interview technique allowed the conversations to be diverse and have insightful responses. Additionally, the effort of achieving a good spread in fields when recruiting software engineers was accomplished to a sufficient degree, capturing responses from the perspective of many different industries. Although responses were sufficient to answer the RQs, one validity threat identified is that the participants were not randomly recruited and as a few of them have had some prior contact with the interviewers, there is a possibility that answers could be biased and that the generalizability is limited.

The age of the participants might also be a factor for generalization, as learned from Anda et al. [48], limits to the findings generalization had to be made. Similar considerations of limiting generalization could be made here as well, as the average age of the sample size was 28 years old, with the youngest being 24 and the oldest being 37. Not only do software engineers of older age have more experience within the field, but they might also have different experiences using GAI tools compared to the younger generation and opinions on how they should be integrated.

5.9 Future Research

From the discussion in this chapter, it is apparent that future research should be done to strengthen the knowledge base on the new topic of GAI in programming from the perspective of the software engineer. An interesting development will be if studies on GAI in more natural settings can become viable. Similar research endeavors to SA1 could then be made to gather more robust insights on the experience of using GAI, utilizing both quantitative and qualitative research methods. Similar research endeavors could also be made on students in the SE field, given that schools provide clearer guidelines on how GAI tools can be used.

The importance of having enough base knowledge to be able to think critically about the outputs of the tools was a common theme at both the individual and industry levels. How one can accelerate knowledge gain to advance one's competencies with the help of GAI tools without risking the potential degeneration of one's cognitive ability can therefore be an interesting area to research. This can also be applicable to both students as well as professionals.

Another interesting area to research is how one can learn to use these tools to prompt the most effective responses. As identified in Section 5.4.3, the role of a software engineer will have to change if these GAI tools can be integrated into their work. Conducting studies on how to be more proficient with these tools, as well as how the tools can provide a better developer experience can then be of great importance for practitioners.

6 Conclusion

As GAI tools have gained prominence in the SE industry recently, this thesis aimed to fill a knowledge gap by focusing on the long-term effects Copilot can have on professional engineers using it, as well as the broader impact GAI can have on the industry. In contrast to previous studies focusing more on the quantitative aspects of the tool itself, this research had a qualitative approach to investigate professional software engineers' perceptions, experiences, and reflections in relation to the use of Copilot and similar GAI tools. Through interviews and inductive thematic analysis of the recurring themes and patterns recognized in the data, the three RQs could be answered:

- What features of the tool help programmers be more efficient in their work and conversely what features hinder them?
- To what extent can the tool affect programmers' problem-solving process?
- Can the potential possibilities and/or hurdles introduced by the tool change the prospects of software engineering as a profession?

Firstly, it was found that the auto-completion feature of Copilot can positively affect a programmer's efficiency in new development, especially when working on more repetitive coding tasks. Conversely, the other feature that Copilot provides, that being the generation of code based on comments in natural language, was seen as a hindrance in their work as it was considered cumbersome to clutter the codebase with unnecessary comments. To combat this hindrance, a feature akin to ChatGPT where the programmer can have a dialog with the AI was the most significant request to make the tool more interactive, enhancing its ability to resemble an actual pair programmer. This way, the tool can be utilized not only in new development but in other areas such as maintenance, testing, and bug-fixing as it would allow the programmer to ask questions to better understand the suggested code or code already written.

Secondly, integrating the tool into software engineers' workflow can optimize their problem-solving process by redirecting focus towards problems at a higher level, while GAI tools can assist in solving lower-level problems, such as implementing code or writing tests. Not only can the use of these tools to some extent eliminate the steps of extensive online searches for solutions, but they also have the potential to increase a programmer's creativity as they can be used as sources of inspiration, suggesting solutions one might not have thought of previously. However, base knowledge and critical thinking are required to ensure the quality of the suggested code, as blindly accepting solutions can harm the codebase. Furthermore, excessively depending on generated solutions could potentially have adverse consequences in the sense of reducing engineers' problem-solving skills and knowledge base.

Finally, regarding the prospects of the SE industry, it is important to note that the use of GAI tools is still in its early stages, and future research should be done in the coming years to analyze the impact. Nevertheless, interesting discussions arose from the emergence of GAI technology in SE. One prospect found was that it will be easier to enter the SE profession by integrating these tools. While this will have advantages and disadvantages, the important thing for future software engineers will be to adapt to these changes to benefit from the advantages. Consequently, it can reshape the role of a software engineer. Furthermore, there still exists some hesitance and concerns surrounding the integration of GAI tools in the industry and as a result, regulations and guidelines on how to use and develop GAI tools will have to be clearly put in place before they can be integrated and trusted fully. This holds for education within the SE field as well, especially considering the marked importance of having a base knowledge of SE topics when using these tools.

Similar to the use of GAI tools in the SE industry, research is also in its early stages. As observed from the process of conducting this thesis, having software companies integrate these tools into their work is not trusted to uphold their privacy, and the legal consequences of using the tools are still in a gray area. Therefore, it can be concluded that studies on GAI in natural settings are not yet fully viable, and the hope is that trust can be established through the implementation of regulations and guidelines, making it possible to analyze these tools in natural settings in a more profound way.

Bibliography

[1] C. H. Hoffmann, "Is AI intelligent? An assessment of artificial intelligence, 70 years after Turing," *Technology in Society*, vol. 68, p. 101893, 2022.

[2] A. van Wynsberghe, "Sustainable AI: AI for sustainability and the sustainability of AI," *AI and Ethics*, vol. 1, no. 3, pp. 213–218, 2021.

[3] S. A. Kumar, "THE RISE OF AI," *Human Capital*, vol. 2, no. 8, pp. 14–28, 2019.

[4] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, *Deep learning techniques for music generation.* Springer, 2020, vol. 1.

[5] J.-P. Briot, "From artificial neural networks to deep learning for music generation: history, concepts and trends," *Neural Computing and Applications*, vol. 33, no. 1, pp. 39–65, 2021.

[6] "Dall-E 2," https://openai.com/dall-e-2/, acessed: 26 nov 2022.

[7] D. Sobania, M. Briesch, and F. Rothlauf, "Choose your programming copilot: a comparison of the program synthesis performance of github copilot and genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1019–1027.

[8] "OpenAI Codex," https://openai.com/blog/openai-codex/, acessed: 26 nov 2022.

[9] "Your AI pair programmer," https://github.com/features/copilot, acessed: 26 nov 2022.

[10] N. Nguyen and S. Nadi, "An empirical evaluation of github copilot's code suggestions," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 1–5.

[11] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," in *2022 IEEE Symposium on Security and Privacy (SP).* IEEE, 2022, pp. 754–768.

[12] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 319–321.

[13] "ChatGPT: Optimizing Language Models for Dialogue," https://openai.com/blog/chatgpt/, acessed: 13 dec 2022.

[14] IBM, "What is deep learning?" accessed: 27 may 2023.

[15] ——, "What is reinforcement learning? Learn about Automated AI For Decision-Making, a process in machine learning that identifies data points, events, and observations that deviate from a data set's normal behavior," accessed: 27 may 2023.

[16] University of York, "The role of natural language processing in AI," accessed: 27 may 2023.

[17] K. Enkelejda et al, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023.

[18] A. Tamkin, M. Brundage, J. Clark, and D. Ganguli, "Understanding the capabilities, limitations, and societal impact of large language models," *arXiv preprint arXiv:2102.02503*, 2021.

[19] P. McCorduck, M. Minsky, O. G. Selfridge, and H. A. Simon, "History of artificial intelligence." in *IJCAI*, 1977, pp. 951–954.

[20] B. G. Buchanan, "A (very) brief history of artificial intelligence," *Ai Magazine*, vol. 26, no. 4, pp. 53–53, 2005.

[21] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *California management review*, vol. 61, no. 4, pp. 5–14, 2019.

[22] A. M. Turing, *Computing machinery and intelligence.* Springer, 2009.

[23] M. Muller, L. B. Chilton, A. Kantosalo, C. P. Martin, and G. Walsh, "Genaichi: Generative ai and hci," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.

[24] Y. Cao et al, "A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt," *arXiv preprint arXiv:2303.04226*, 2023.

[25] J. Rudolph, S. Tan, and S. Tan, "Chatgpt: Bullshit spewer or the end of traditional assessments in higher education?" *Journal of Applied Learning and Teaching*, vol. 6, no. 1, 2023.

[26] B. D. Lund and T. Wang, "Chatting about chatgpt: how may ai and gpt impact academia and libraries?" *Library Hi Tech News*, 2023.

[27] "GPT-4 is OpenAI's most advanced system, producing safer and more useful responses," https://openai.com/product/gpt-4, acessed: 23 mar 2022.

[28] Y. K. Dwivedi et al, ""So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy," *International Journal of Information Management*, vol. 71, p. 102642, 2023.

[29] A. Maedche et al, "Ai-based digital assistants: Opportunities, threats, and research perspectives," *Business & Information Systems Engineering*, vol. 61, pp. 535–544, 2019.

[30] K. Siau and W. Wang, "Artificial intelligence (ai) ethics: ethics of ai and ethical ai," *Journal of Database Management (JDM)*, vol. 31, no. 2, pp. 74–87, 2020.

[31] H. Yu, Z. Shen, C. Miao, C. Leung, V. R. Lesser, and Q. Yang, "Building ethics into artificial intelligence," *arXiv preprint arXiv:1812.02953*, 2018.

[32] J. Gratch and N. J. Fast, "The power to harm: Ai assistants pave the way to unethical behavior," *Current Opinion in Psychology*, p. 101382, 2022.

[33] C. Flick and K. Worrall, "The ethics of creative ai," in *The Language of Creative AI: Practices, Aesthetics and Structures.* Springer, 2022, pp. 73–91.

[34] Caballar, Rina Diane, "Ownership of AI-Generated Code Hotly Disputed A copyright storm may be brewing for GitHub Copilot," 2022.

[35] M. A. Jacobs, J. C. Gratz, and T. Cheung, "OpenAI Motion to Dismiss," 2023, https://s3.documentcloud.org/documents/23589439/openai-motion-to-dismiss.pdf.

[36] A. Al-Kaswan and M. Izadi, "The (ab) use of open source code to train large language models," *arXiv preprint arXiv:2302.13681*, 2023.

[37] J. Danaher, "Toward an ethics of ai assistants: An initial framework," *Philosophy & Technology*, vol. 31, no. 4, pp. 629–653, 2018.

[38] L. L. Dhirani, N. Mukhtiar, B. S. Chowdhry, and T. Newe, "Ethical dilemmas and privacy issues in emerging technologies: A review," *Sensors*, vol. 23, no. 3, p. 1151, 2023.

[39] European Commission, "The Artificial Intelligence Act," 2021, https://artificialintelligenceact.eu/.

[40] R. Eitel-Porter, "Beyond the promise: implementing ethical ai," *AI and Ethics*, vol. 1, pp. 73–80, 2021.

[41] A. M. Dakhel et al, "Github copilot ai pair programmer: Asset or liability?" 2022. [Online]. Available: https://arxiv.org/abs/2206.15331

[42] M. Chen et al, "Evaluating large language models trained on code," 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[43] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491101.3519665

[44] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, "The programmer's assistant: Conversational interaction with a large language model for software development," in *Proceedings of the 28th International Conference on Intelligent User Interfaces*, ser. IUI '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 491–514. [Online]. Available: https://doi.org/10.1145/3581641.3584037

[45] R. Challen, J. Denny, M. Pitt, L. Gompels, T. Edwards, and K. Tsaneva-Atanasova, "Artificial intelligence, bias and clinical safety," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, p. 14, 2018.

[46] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," 2016. [Online]. Available: https://arxiv.org/abs/1606.06565

[47] M. Jorgensen and S. Grimstad, "The impact of irrelevant and misleading information on software development effort estimates: A randomized controlled field experiment," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 695–707, 2010.

[48] B. C. Anda, D. I. Sjøberg, and A. Mockus, "Variability and reproducibility in software engineering: A study of four companies that developed the same system," *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 407–429, 2008.

[49] K.-J. Stol and B. Fitzgerald, “The ABC of Software Engineering Research,” *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, pp. 1–51, 2018.

[50] J. Singer, S. E. Sim, and T. C. Lethbridge, “Software engineering data collection for field studies,” *Guide to advanced empirical software engineering*, pp. 9–34, 2008.

[51] J. Preece, H. Sharp, and Y. Rogers, *Interaction design: beyond human-computer interaction*. John Wiley & Sons, 2019.

[52] D. S. Cruzes and T. Dyba, “Recommended steps for thematic synthesis in software engineering,” in *2011 international symposium on empirical software engineering and measurement*. IEEE, 2011, pp. 275–284.

[53] V. Braun and V. Clarke, “Using thematic analysis in psychology,” *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

A

Appendix

A.1 Survey questions

Semantic differential scale 1-7

- How helpful was GitHub Copilot in increasing your work efficiency?
  - Very unhelpful - Very helpful
- Was Copilot effective in finding solutions to problems?
  - Very ineffective - Very effective
- How significant was Copilot in influencing your approach to solving problems in software engineering?
  - Very insignificant - Very significant
- How frequently did you use the suggestions offered by GitHub Copilot?
  - Very infrequently - Very frequently
- Were you satisfied with the features offered by GitHub Copilot?
  - Very dissatisfied - Very satisfied
- Was it easy for you to use Copilot?
  - Very hard - Very hard

Open-ended questions

- Have you encountered any obstacles in using GitHub Copilot for problem-solving? If so, what/why?
- Can you describe a specific instance where GitHub Copilot helped you complete a task more efficiently?
- Can you describe a time when GitHub Copilot assisted you in finding a solution to a problem?

Other comments

- If you have any other comments or viewpoints to bring up (eg. "I didn't use Copilot today)

A.2 Interview questions

Demographic data

- Whats your main area of the profession?
- What is your age?

RQ1

- Which features do you think would be most helpful for programmers in their work?
- Can you provide an example of a task that you think Github Copilot could make easier for programmers?
- In what ways do you think Github Copilot could help programmers be more efficient in their work?
- What areas of software development do you think Copilot affects the most? (Evolution/Bug-fix/Testing/Maintenance/New Development)
- Are there any features you think would hinder a programmer in their work?
- Can you think of any ways that the tool could be improved?

RQ2

- Do you think the tool could change the way programmers approach problem-solving?
- Can you give an example of how you think Github Copilot might influence a programmer's problem-solving process?
  - If they are struggling to give an example, provide an example of our own
  - Let's say you are given the task to program the fibonacci sequence but you get stuck and don't know how to solve it off the top of your head. What would the steps be in your process to solve the task today without this tool? How would you solve it if you had the tool available?
- Do you think that Github Copilot could potentially limit a programmer's ability to come up with original solutions to problems, i.e limit their creativity?
  - If they are struggling to find answers, give them this definition to base their answer off of
  - "Problem-solving also has to do with utilizing creativity and logical thought processes to identify problems and resolve them with software."

RQ3

- How do you think Github Copilot could change the skill requirements for software engineers?
- Do you think Github Copilot could make it easier or harder for new programmers to enter the profession?
- How might GitHub Copilot impact the way we approach and teach programming and software development in educational setting dis?
- Can a widespread adaptation of AI-tools like Copilot have an effect on the level of expertise within the profession?
  - For example, people's attention spans and memories have worsened due to use of mobile phones that help us with various tasks as well as entertainment

  you imagine any potential negative consequences that could arise from widespread adoption of Github Copilot?
  - Ethical issues

- Legal issues
- Problem-solving process