

**Master's Programme in Software and Service Engineering**

# AI tools' effects on working methods, productivity and project metrics

A software engineering case study

---

**Tuomas Willberg**

Copyright ©2025 Tuomas Willberg

---

<b>Author</b>	Tuomas Johan Willberg		
<b>Title of thesis</b>	AI tools' effects on working methods, productivity and project metrics		
<b>Programme</b>	Software and Service Engineering		
<b>Major</b>	Computer, Communication and Information Science		
<b>Thesis supervisor</b>	D. Sc. (Tech.) Jari Vanhanen		
<b>Thesis advisor(s)</b>	M. Sc. Lauri Pitkänen		
<b>Collaborative partner</b>	Digia		
<b>Date</b>	<b>Number of pages</b>	<b>Language</b>	
21.11.2025	64 + 7	English	

---

### Abstract

AI powered tools for software development are rapidly becoming a common factor in developers' daily workflows. These tools have significant potential to reshape the best practices in software development, yet the responsibility for realizing this potential ultimately depends on their user. While much research has been done on AI and its capabilities and performance, AI tool use cases in software development and their effects on productivity have received less attention.

This master's thesis studied AI tool usage and their effects on productivity through a literature review as well as a case study. The literature review examined the current state of the subject through scientific literature, whereas the case study aimed at understanding practical application of AI tools in a real-world software development context. The case study also analysed numerical project metrics from time before and after the adoption of AI tools.

The research findings showed that AI tools have the potential to increase the productivity of software developers. The most common factors behind the increase in productivity with AI tools were automation of simple and monotonous tasks, helping with learning new technologies and topics and improvement of work comfort. The biggest benefits were recorded when AI tools were used with caution and predefined use cases.

---

**Keywords** AI, Software development, Copilot, Productivity

---

---

**Tekijä** Tuomas Johan Willberg

---

**Työn nimi** Tekoälytyökalujen vaikutukset työskentelytapoihin, tuottavuuteen ja projektien avainlukuihin

---

**Koulutusohjelma** Software and Service Engineering

---

**Pääaine** Computer, Communication and Information Science

---

**Vastuupettaja/valvoja** TkT Jari Vanhanen

---

**Työn ohjaaja(t)** TkM Lauri Pitkänen

---

**Yhteistyötaho** Digia

---

**Päivämäärä** 21.11.2025    **Sivumäärä** 64 + 7    **Kieli** Englanti

---

### **Tiivistelmä**

Ohjelmistokehityksen tukena käytettävät tekoälytyökalut ovat kovaa vauhtia yleistymässä osana kehittäjien arkea. Kyseisillä työkaluilla on valtava potentiaali muokata ohjelmistokehityksen parhaita käytäntöjä, mutta suuri vastuu potentiaalın realisoitumisesta on kuitenkin työkalujen käyttäjän harteilla. Vaikka tekoälyä ja sen kyvykkyyttä on tutkittu paljon, ovat tekoälytyökalujen käyttö ja niiden vaikutukset kehittäjän tuottavuuteen jääneet vähemmälle huomiolle.

Tässä diplomityössä tekoälytyökalujen käyttöä ja niiden käytön vaikutuksia tutkittiin kirjallisuustutkimuksen, sekä tapaustutkimuksen avulla. Kirjallisuustutkimus kartoitti tilannetta tieteellisestä kirjallisuudesta, kun taas tapaustutkimuksella pyrittiin ymmärtämään tekoälytyökalujen käyttökohteita todellisessa ohjelmistokehityksen arkipäiväisessä kontekstissa. Tapaustutkimuksessa analysoitiin myös projektien numeerista dataa ennen ja jälkeen työkalujen käyttöönoton.

Tutkimus osoitti, että tekoälytyökalut omaavat potentiaalia kehittäjien tuottavuuden parantamiseksi. Yleisimmät tekijät tuottavuuden kasvussa olivat yksinkertaisten ja monotonisten työtehtävien automatisointi, uusien teknologioiden opiskelun helpottaminen ja työ mukavuuden parantaminen. Suurin hyöty tekoälytyökaluista oli, kun niitä käytettiin maltillisesti ennalta testattujen käyttötapojen mukaisesti.

---

**Avainsanat** Tekoäly, Ohjelmistokehitys, Copilot, Tuottavuus

---

## Table of contents

Preface and acknowledgements .....	7
Abbreviations .....	8
1 Introduction .....	9
2 Literature review .....	12
2.1 Theoretical background .....	12
2.1.1 Software development.....	12
2.1.2 The productivity of a software developer .....	13
2.1.3 Effort estimation.....	15
2.1.4 AI and its applications .....	16
2.2 AI tools in software development .....	17
2.2.1 Use cases .....	18
2.2.2 Challenges and limitations .....	23
2.2.3 Impacts in productivity .....	28
3 Research methods .....	32
3.1 The case .....	32
3.2 Case study scope .....	33
3.3 Research methodology.....	34
3.4 Data collection .....	35
3.4.1 Questionnaire .....	35
3.4.2 Observations .....	36
3.4.3 Archival analysis .....	36
3.5 Data analysis methods .....	37
3.5.1 Questionnaire .....	38
3.5.2 Observations .....	38
3.5.3 Archival analysis .....	38
4 Results .....	40
4.1 Questionnaire.....	40
4.1.1 Quantitative answers.....	40
4.1.2 Thematic analysis .....	42

4.2	Observations .....	46
4.3	Archival Jira analysis .....	48
5	Discussion.....	51
5.1	AI tools' impact on developers' work.....	51
5.2	AI tools' effects on developer productivity .....	52
5.3	Best practices for AI tool usage.....	53
5.4	Future work.....	55
6	Validity.....	56
6.1	Construct validity .....	56
6.2	Internal validity.....	56
6.3	External validity .....	57
6.4	Reliability .....	58
7	Conclusion .....	59
	References.....	60
	A. Questionnaire .....	65

## **Preface and acknowledgements**

I want to thank Professor Jari Vanhanen and my thesis advisor Lauri Pitkänen for their good advice and guidance. I am also deeply grateful to my family and friends for standing by me throughout my studies.

Otaniemi, 21 11 2025  
Tuomas Johan Willberg

## Abbreviations

AI	artificial intelligence
LLM	large language model
NLP	natural language processing
GPT	generative pre-training transformer
RPA	robotic process automation
API	application programming interface
IDE	integrated development environment
MCP	model context protocol

# 1 Introduction

In the field of software development, AI-powered tools are rapidly becoming more prevalent. In their empirical study about the technical capabilities and developers' perceptions of AI tools, Kuhail et al. (2024) describe that tools like ChatGPT and Claude are large language models (LLMs) that work in automating monotonous tasks and otherwise improving developers' productivity and efficiency. These tools act as conversational agents that utilize natural language processing (NLP) techniques to understand human language as well as programming languages (Kuhail et al., 2024; Zhou et al., 2025). The significance of these AI tools is continuously growing, and they are becoming an integral part of the everyday work of a software developer. The capabilities of these AI tools are also growing rapidly in terms of problem-solving as well as understanding new and less common technologies (Kuhail et al., 2024).

The usage of AI powered tools is increasing in many fields as well as among students. Despite that, measuring of the perceived effects in productivity is difficult and has not yet been widely researched (Ziegler et al., 2024). As Ziegler et al., (2024) describe, it is tricky to measure the productivity of software developers in general.

As these new AI tools are continuously growing in usage, it becomes increasingly more interesting to understand how they should optimally be used and what are the best practices with them. As Kuhail et al. (2024) describe, the impact of these AI tools does not only rely on their ability to solve complex coding problems but also on developers' ability to utilize them. The study by Ziegler et al. (2024) highlights that modern AI tools have reached such a level of performance that their effectiveness largely depends on their user. As these tools work with conversational prompting the possibilities for their usage are endless. With various identified use cases such as code explaining, providing suggestions for new code and planning structure and required technologies (Kuhail et al., 2024; Zhou et al., 2025; Zhang et al., 2023), the interest towards understanding the best practices with AI tools grows in importance.

This master's thesis examines the influence of AI on the daily work of software developers. Its primary objective is to link the applications and use cases of AI tools to changes or impacts in productivity, thereby contributing to an understanding of best practices for their utilization.

This thesis can be divided into two main sections. The first one being a literature review exploring the current landscape of software development and the utilization of AI tools while the second part presents a descriptive embedded single case study in a unit of a Finnish IT company. The literature review examines the current state of AI tools and their effects on the daily routines and productivity of software developers. The literature review also offers significant background information on topics that lay the theoretical background for my research. The case study, on the other hand, is done to provide empirical insights into how AI tools are integrated into real-world software development practices and what are their effects on workflow efficiency and real-life project metrics.

The case study aims at shedding light on the actual impacts that the AI tools have had in the productivity and daily work of software developers as well as understanding where the field is headed and what are the best practices with these tools. The case study is carried out in a unit of a company that develops robotic process automation (RPA) solutions for customers to automate monotonous business tasks. The unit has been working on this field since 2018, and the AI tools were introduced as part of work in the beginning of 2024. Since then, developers have been able utilize these tools how they like. However, as the capabilities of these tools and assistants are growing rapidly, it has become more and more interesting to understand their impacts and how to make the most out of them. This study offers some concrete best practices in using AI tools in software development.

My research questions for this master's thesis are as follows:

**RQ 1.** How have AI tools affected software developers' daily work?

**RQ 2.** How have AI tools affected software developers' productivity?

Both the literature review and case study aim to answer both research questions. The case study seeks answers to these questions through a questionnaire on developers' practices with AI tools, as well as an archival analysis of historical project data from Jira, a kanban board containing data on all past and current projects. Developers' practices with AI tools are also studied through observations of daily work. The data collection methods employed in the case study approach the research questions from different viewpoints. The questionnaire and observations offer insight into practical AI tool usage and its effects whereas the archival analysis takes a more numeric approach

to the matter by analysing the change in project metrics, and thus productivity, after the introduction of Copilot.

The contents of this thesis consist of the following chapters. Chapter 2 provides a literature review on the topic first explaining related background information and then diving into the topic of AI tool usage in software development. Chapter 3 explains and motivates the research methods employed in this study and presents the case study scope. Chapter 4 presents the results and findings of the case study. Chapter 5 is discussion where the results of the case study are triangulated with findings from the literature review. Chapter 6 expresses the threats to the validity of this study as well as the measures taken to mitigate them. And finally in chapter 7 a conclusion is drawn. Last pages are dedicated to references and appendixes.

## **2 Literature review**

This chapter is a literature review exploring previous research work on AI tools usage in software development. The reference material for this chapter was chosen from scientific databases Scopus, Google Scholar and Web of Science based on factors such as release context and the number of times the article has been referenced. The search for scientific literature on the topic was conducted using keywords and phrases such as “AI tools”, “AI tool usage”, “software development”, and “productivity”.

The sub-chapter 2.1 presents the theoretical background related to software development projects as well as AI and its applications in software development tools. This information is crucial in understanding the current situation in AI tool usage in software development presented in sub-chapter 2.2. Furthermore, sub-chapter 2.2 analyses the benefits and limitations of using AI tools in software development linking them to concrete use cases from scientific literature.

### **2.1 Theoretical background**

This sub-chapter discusses the theoretical background for analysing AI tools usage in software development. It provides an overview of software development in general and the nature of the work. Then it explains some concepts that are important to understand the full implications of AI tools usage. These concepts include the productivity of a software developer, effort estimation and the working principles of these AI tools.

#### **2.1.1 Software development**

Software development is known to be a demanding and complex task. As Brooks (1987) defines, the essence of software is “a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions” making software abstract yet highly precise and detailed. Brooks (1987) follows by stating that this makes building software so hard. Brooks (1987) also emphasizes the difficulty of representing abstract software in a geometric form, noting that, unlike landscapes that can be illustrated with maps or silicon chips with diagrams, software lacks a comparable geometrical visual representation.

Faraj and Sproull (2000) describe software development as “knowledge work” where the most important resource is expertise. This highlights how

communication is an integral part of software development. Maurer and Hellmann (2013) explain that software development projects consist of teams with experts in various fields where business representatives, for example, understand the problem to be solved but do not have the necessary programming skills to create the solution for it. With this Maurer and Hellmann (2013) imply that communication between all stakeholders is mandatory for the project to succeed.

In addition to adhering to schedule and budget constraints, a critical factor in assessing the success of a software project is the quality of the software produced. Sommerville (2016, Chapter 24.1) explains that the software industry has established a standard definition of quality based on conformance to product specifications. Yet the quality of a software cannot only be measured through the created functionality but through its non-functional characteristics (Sommerville, 2016, Chapter 24.1). Sommerville (2016, Chapter 24.1) provides an example of non-functional characteristics in figure 1.

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

**Figure 1. Software quality attributes by Sommerville (2016, Chapter 24.1)**

All in all, as Brooks (1987) states, building software will always be hard and there is “no silver bullet”. The definition for project success is not as straight forward as one might imagine and at the centre of the process is communication and shared understanding of the process (Faraj and Sproull, 2000).

### **2.1.2 The productivity of a software developer**

The productivity of software developers is a hugely important factor for a software project to be successful, yet there is no easy solution to objectively measure this productivity (Oliveira et al., 2020). As stated in the systematic literature review by Chapetta and Travassos (2020), it is important to understand developers’ productivity as the profit of software companies directly depends on it. In their survey study, Meyer et al. (2017) explain that research on factors affecting developer productivity has been ongoing since the 1970s, characterizing it as a multifaceted phenomenon shaped by technical elements, including technologies employed, project scale, and quality, alongside social aspects such as team dynamics, professional experience, and

workplace conditions. Graziotin et al. (2013) note that despite well-established research, little is still known about the productivity of an individual developer. Oliveira et al. (2020) complement this by stating that it is much easier to measure productivity of a team or organization than an individual.

Chapetta and Travassos (2020) define the term productivity as “the relationship between the outcome and the resources used to produce it”. With this definition it would be very intuitive to measure the relationship between the output of developers work, code, and time used to produce it. In their study Oliveira et al. (2020) present two different approaches to measuring developer productivity - produced code per time and produced commits per time. The offered approaches for measuring productivity in produced code over a period of time either simply analysing the rows of code produced, number of files produced, or the amount of produced code as well as its complexity. In the commit-based approach, on the other hand, Oliveira et al. (2020) measured the number of commits, amount of committed code rows and amount of committed code characters. Oliveira et al. (2020) found in their study that metrics based on the amount and complexity of code produced by developers exhibited the strongest correlation with lead developers' perceptions of individual developer productivity. In the end, Oliveira et al. (2020) state that the observed productivity is always a subjective matter as developers themselves value high quality code more, whereas project managers tend to value the number of produced features higher.

Yilmaz et al. (2016) approach developer productivity more from a social activity standpoint, viewing software development as a collaborative process influenced by factors such as team dynamics, workplace environment, and social interactions, rather than a standalone metric suitable for objective measurement. They argue that productivity is shaped by a complex interplay of technical and social elements, with social capital playing a pivotal role in enhancing team performance. Complementing this, Graziotin et al. (2013) present a study where professional and student software developers were observed in their natural work environments on real software projects, with their behaviour monitored and self-assessments of state of mind and productivity collected every 10 minutes using a questionnaire. Their findings suggest that a developer's positive disposition is closely linked to higher productivity, reinforcing the notion that emotional well-being in the workplace is critical for higher performance. Together, these studies shift the focus from purely technical metrics of productivity to a more holistic understanding that incorporates social and emotional dimensions. For instance, Yilmaz et al. (2016) identify motivation and team alignment as key influencers, while

Graziotin et al. (2013) provide evidence that developers' subjective experiences directly impact their output.

As stated, the concept of software developers' productivity is complex and hard to grasp. Graziotin et al. (2013) state that a common practice in psychology is to let participants self-evaluate their own productivity, as self-evaluations often produce consistent objective measures. Their findings substantiate a robust correlation between developers' self-evaluated productivity and their perceived productivity within the study's context, underscoring the reliability of introspective measures in capturing productivity dynamics. Similarly, Meyer et al. (2017) offer a complementary perspective, implying that individual personality traits significantly influence perceptions of productivity-related factors. Their work suggests that developers exhibit diverse preferences shaped by their personalities. For instance, some thrive in solitary environments, valuing uninterrupted focus, while others derive motivation and efficiency from social environments. To summarize, productivity lacks a singular definition, but various approaches can be employed to understand individual and group productivities as well as variables influencing them.

### **2.1.3 Effort estimation**

Estimating a software project's required effort is an important part of software development and often necessary for a project to start (Pressman, 2005, p. 642). McConnell (1996) explains that only after this estimation can the needed number of people for the project be decided. McConnell (1996) also describes that some project attributes affecting the effort estimation are the size of the project, wanted features, needed quality and the trade-offs between features, quality and time. The more is known about these attributes, the more accurate the estimation. Due to the nature of software development, important information often emerges during the project and thus the effort estimation changes and optimally becomes more accurate during the project (McConnell, 1996).

Pressman (2005, pp. 659-661) explains that a common way for generating effort estimations is to use data in empirically derived formulas or algorithms. In practice this means using the data such as object points, a measure calculated from the number of screens, reports and components, with an algorithmic approach, the most notable example being COCOMO from the classic book "Software Engineering Economics" by Boehm (1984). However, Pressman (2005, p. 659) notes that a vulnerability of this approach is that

“most estimation models are derived from a limited sample of projects”, which limits the appropriateness of this approach in some situations.

Another common way to perform effort estimation is expert judgment. The study by Jørgensen (2004) shows that at least in the late 90s and early 2000s expert judgment was the most employed effort estimation strategy in software development. According to Jørgensen (2004), effort estimation qualifies as expert judgment when “the estimation work is conducted by a person recognized as an expert on the task” and the evaluation relies primarily on “a nonexplicit and non-recoverable reasoning process,” such as intuition. Jørgensen (2004) suggests that possible reasons for the high usage rate of expert judgment could include organizations being uncomfortable using models or algorithms that they don't fully understand or organizations not perceiving benefits from using them.

Due to the complex and hard to grasp nature of software development, effort estimation is often difficult. McConnell (1996) notes that forming an effort estimation often includes doing trade-offs between time and quality as well as accuracy of estimation and control over project. As previously noted, effort estimations frequently evolve throughout the course of a project. Pressman (2005) emphasizes that estimation should be regarded as just one part of the project's iterative nature. What makes effort estimation even harder is to find an estimate that pleases all different parties involved such as developers, project managers and customers (McConnell, 1996). Thus, it can be challenging to find a way to align the ambitious views from customers and the pragmatic thinking of developers.

#### **2.1.4 AI and its applications**

In recent years, the term AI has become very popular and often so much so that the meaning for the term has become somewhat unclear. Broadly speaking, the term AI serves as an umbrella concept encompassing a range of technologies and computational approaches. Abbass (2019), approaches the definition of AI to be about autonomy or “automation of automation” rather than pure automation. Abbass (2019) sees AI as a social and cognitive phenomena that involves socially interacting machines, performing complex cognitive tasks, and communicating efficiently within a society.

Defining AI is a demanding task, but in terms of AI, the scope of this thesis is large language model (LLM) based solutions, trained on large amounts of data, that can be leveraged in software development. A core part of these AI

tools is natural language processing (NLP) which is a combination of linguistics and computer science to convert human language to suitable inputs for the LLM and vice versa (Mohammad et al., 2023). According to the book “Exploring the Power of ChatGPT” by Sarrion (2023, Chapter 1.2) NLP is based on tokenizing user input, meaning that the input is split into smaller parts so that the LLM can understand the relationships between the words of the input. After this the LLM leverages multidimensional vector embedding to present the user input numerically to help the LLM in processing it (Sarrion, 2023, Chapter 1.2).

Following NLP tasks, the LLM leverages its core neural network to process input and craft responses, a process refined during customization where the network is tuned with training data to enhance its ability to handle specific tasks such as answering questions or generating domain-specific text (Sarrion, 2023, Chapter 1.2). Sarrion (2023, Chapter 1.2) explains that “neural networks have revolutionized machine learning and data processing” by enabling machines to learn from a dataset and perform tasks based on what they have learned.

All the features briefly described above form the actual large language model. As the study by Jiang et al. (2022) shows, LLM based solutions are typically “used” or instructed through prompts in natural language. As the study by Jiang et al. (2022) points out, quality and provided context of the prompt play a significant role in quality of the LLMs output. A particularly interesting AI tool for this thesis is GitHub’s Copilot. Chen et al. (2021) explain that Copilot is an AI tool built on top of OpenAI’s coding specific language model Codex, which was at the time of writing their article trained on 50 million GitHub projects equalling 159 gigabytes of data. At the core of GitHub’s Copilot is its capability to offer code suggestions in real time, provide documentation for code as well as providing information or code through prompts (GitHub, 2025).

## **2.2 AI tools in software development**

In recent times, the widespread introduction of AI tools and programming assistants like ChatGPT and GitHub Copilot has transformed the traditional approach to software development, significantly impacting the software engineering landscape as explained by the findings of the extensive survey about AI tool usage and usability in software development by Liang et al. (2024). There have been numerous studies measuring AI tools’ effects on programming productivity in recent years. The consensus seems to be that

the increase in productivity enabled by AI tools is somewhere between 30 and 80 per cent (Cui et al., 2024; Kuhail et al., 2024; Solohubov et al., 2023). As identified by Moroz et al. (2022) in their study analysing GitHub issues about problems users encountered when using GitHub Copilot, AI tools have the capabilities to provide assistance in each step of the software life cycle such as requirements engineering, architectural planning and development. As Kuhail et al. (2024) state, the perceived benefits of these AI tools are often highly dependable on their user's attitude towards them. To delve further into this topic, this chapter provides more insight into how AI tools are used in sub-chapter 2.2.1. Then possible limitations and challenges regarding AI tool usage are discussed in 2.2.2. And finally sub-chapter 2.2.3 analyses AI tools' effects on productivity from multiple viewpoints.

### **2.2.1 Use cases**

This chapter covers the use cases for AI tools from scientific literature articles. These use cases are findings from studies ranging from controlled environments to real world studies. These findings include technical programming related use cases like automated code creation and boilerplate code production (Pangavhane et al., 2024; Prather et al., 2023) and more non-technical use cases to optimize workflows by using AI tools as conversational agents (Kuhail et al., 2024)

By far the most common use case for AI tools in software development, mentioned in scientific literature, is automated code generation. As Solohubov et al. (2023) explain in their article, AI tools like ChatGPT and Copilot enhance software development by leveraging advanced language understanding and code generation to provide auto-completion, hints, and rapid implementation, significantly accelerating the programming process. These autocomplete suggestions and code generation capabilities help in creating "boilerplate code" and eliminating the struggle with an "empty canvas" by helping developers to quickly produce a solid starting point for their code (Liang et al., 2024; Vaithilingam et al., 2022). Ziegler et al. (2024) note that automated code generation not only provides a starting point but also enables developers to bypass writing simple, repetitive code that is simple enough for AI tools to generate effortlessly. In their study, Solohubov et al. (2023) reported that the usage of GitHub Copilot "allowed for approximately a 30% increase in writing the logic of the project". In contrast to analysing spent time, Liang et al. (2024) found out that developers using GitHub Copilot report a median of 30.5% of their code being generated with the tool's assistance. Developers cite the primary reasons for using AI programming assistants as their ability

to minimize keystrokes, expedite task completion, and recall correct syntax (Liang et al., 2024; Mendes et al., 2024). In the qualitative study mapping out developers perceptions of AI in software development with semi-structured interview, Stavridis and Drugge (2023) report that the biggest benefits from AI tools are accomplished when working with technologies and syntaxes completely new for the user.

In addition to code generation, AI tools also provide help in quality assurance (Mendes et al., 2024; Zhang et al., 2023). In the study by Zhang et al. (2023), it is noted that oftentimes the code generated solely by AI tools might be as good as one produced by humans, however the code from AI is often very well documented. This means that the code from AI contains comments and doc strings that help in understanding the code. In addition to comments, code produced by AI is often written in an easy to read and debug manner (Zhang et al., 2023). The study by Pudari and Ernst (2023) found out that AI produced code was always correct syntax wise. They also state that when provided with suitable scope and context, the AI tools are able to provide bug free code. In their study assessing GitHub Copilot's code output quality with different prompts in comparison to human created code, Moradi et al. (2023) add to the topic of importance of prompt quality by explaining that the quality of the AI generated code often heavily depends on the quality of the prompt provided by the user. The study by Mendes et al. (2024), analysing developers' experiences with AI tools through semi-structured interviews, also supports AI tool usage in quality assurance by stating that AI tools are often used to rewrite and document already created code to improve its quality through better readability and easier to understand design.

In their multi-case case study researching LLM adoption in industrial settings, Kemell et al. (2024) found that in terms of quality assurance, AI tools are also helpful in creating tests for code. While referring to software development, Pangavhane et al. (2024) state that businesses are able to speed up their releases with AI tools and AI driven testing leading to approximately 30 % decrease in testing times. They explain that this is achieved by utilizing AI's capabilities in creating high quality tests based on project and code requirements. In their study, Kemell et al. (2024) found out that one respondent had begun to enjoy test driven development more with these AI capabilities helping in test creation early in development. Pangavhane et al. (2024) add that the test creation capabilities are a result of AI's proficiency with a large number of programming languages, libraries and frameworks.

Not necessarily a use case, but rather a reason to use AI tools in software development is the reported user satisfaction (Kuhail et al., 2024; Prather et al., 2023). In their article focusing on GitHub Copilot's effects on developer productivity through a multi-team case study Smit et al. (2024) highlight developer satisfaction's importance in productivity and in their study, they found out that 80% of respondents reported an increase in job satisfaction after integrating GitHub Copilot as part of their routine work. Prather et al. (2023) state that the satisfaction gained through the usage of AI tools helps developers improve and learn new things faster. Furthermore, they highlight the importance of a positive mindset in learning. Kuhail et al. (2024) explain that developers are more motivated towards their work in general when AI tools speed up repetitive, monotonous tasks allowing them to spend more time on more complex issues. The study by Prather et al. (2024) showed that developers are interested in using AI tools in coding even though it sometimes might lead to long debugging sessions. This shows that sometimes developers might prefer the more fun way of doing things even if it might risk spending more time doing it.

In discussing common uses of AI tools in software development, information retrieval is another key focus in recent scientific literature (Kemell et al., 2024; Pudari and Ernst, 2023; Vaithilingam et al., 2022). Stavridis and Drugge (2023) expect that in the future the traditional workflow of searching answers from Google and Stack Overflow in particular shifts to mostly asking questions from AI tools as the primary option. They explain that with AI tools' vast amounts of training data, including scraped internet data from the previously mentioned sources, the AI tools can provide developers with faster and more accurate answers. Pudari and Ernst (2023) explain that especially when the first AI tools were introduced, information retrieval was a common use case for them. Kemell et al. (2024) underscore that with GitHub Copilot, the user does not need to even leave their IDE to search for information, as Copilot provides an integrated conversation window within IDEs. Moroz et al. (2022) note that AI tools' information retrieval capabilities also extend beyond programming topics, proving highly valuable for gathering background information needed to develop field-specific solutions.

In their case study researching AI tools integration to developers' daily workflows through observations and a questionnaire Coutinho et al. (2024) note that, beyond implementing functionality, AI tools are also capable of providing suggestions in problem solving as well as turning requirements into code level solutions. In their 2023 study, Stavridis and Drugge characterize AI tools as "collaborative entities," a term that underscores their role as

augmentations of users' capabilities rather than independent solution providers. In addition to code level problem solving, Mbizo et al. (2024) explain that AI tools are also helpful in designing system architecture and file structure especially in cases where data from previous similar projects is available. AI tools help in problem solving can also be approached from another angle. Pangavhane et al. (2024) state that with the help of AI tools in automating and speeding up other activities, users are left with more time for problem solving facilitating a room for more research as well as trial and error.

Another major use case for AI tools is code explanation and suggestions for technologies and frameworks (Kemell et al., 2024; Kuhail et al., 2024). In their study, Kuhail et al. (2024) list six of the most prominent AI tool use cases and these include code explanation, best practice suggestions and tool recommendations. Regarding tools and best practices, they explain that with AI tools' large knowledge base on several technologies, it can suggest best practices and tools for various scenarios. Kuhail et al. (2024) provide a detailed explanation indicating that this refers to AI-generated guidance on programming best practices, such as the use of design patterns and efficient implementation strategies that align with industry standards. At the tool level, this translates to recommendations for specific libraries or frameworks that are contextually appropriate for the task at hand. Moroz et al. (2022) add that these suggestions are made possible by modern AI tools' extensive knowledge in various technologies and frameworks.

The AI tools' capabilities also allow for fast code explanation for existing code. Kemell et al. (2024) explain that in this fashion AI tools can be extremely helpful in situations where one must understand or edit code produced by someone else. Furthermore, current AI tools have the ability to explain and summarize entire code bases or repositories. Kuhail et al. (2024) complement this by highlighting the usefulness of AI tools explanations for singular algorithms or code snippets. This means explaining how the code works and how it is used in the project. In addition to interpreting existing code written by others, AI tools can also assess the user's own code and provide explanations for potential modifications or improvements (Stavridis and Drugge, 2023).

The final major use case for AI tools in software development researched in the scientific literature is studying and learning new topics (Finnie-Ainsley et al., 2022; Moroz et al., 2022). This is an interesting theme as the software development industry is known for rapid changes and new emerging technologies requiring professionals to always keep their knowledge up to date

(Smit et al., 2024). In the study by Liang et al. (2024) many respondents reported being interested in GitHub Copilot to learn new technologies, mindsets and approaches in addition to being interested in automating code production to some extent. The article by Finnie-Ainsley et al. (2022) goes as far as stating that AI tool technology “should be of interest to all computing educators”. They explain that AI tools could be utilized in learning by example or in explaining and recognizing qualities that improve code in a defined context. On a concrete level this learning could mean studying new coding languages while utilizing AI tools to help with syntax or reviewing and analysing different code snippets to recognize what makes some snippets better than others. Finnie-Ainsley et al. (2022) further describes that in the future, coding education could include more code review and evaluation to avoid pitfalls with automatic code generation used in coding exercises.

**Table 1. Summary of identified AI tool use cases**

<b>Use Case</b>	<b>References</b>
Automated code generation	Liang et al., 2024; Mendes et al., 2024; Stavridis and Drugge, 2023; Solohubov et al., 2023; Vaithilingam et al., 2022; Ziegler et al., 2024;
Improving code quality and creating documentation	Mendes et al., 2024; Moradi et al., 2023; Pudari and Ernst, 2023; Zhang et al., 2023
Creating tests	Kemell et al., 2024; Pangavhane et al., 2024; Weber et al., 2024
Improving working mindset	Kuhail et al., 2024; Prather et al., 2023; Smit et al., 2024
Information retrieval	Kemell et al., 2024; Moroz et al., 2022; Pudari and Ernst, 2023; Stavridis and Drugge, 2023; Vaithilingam et al., 2022;
Problem solving and solution design	Coutinho et al., 2024; Kemell et al., 2024; Mbizo et al., 2024; Pangavhane et al., 2024; Stavridis and Drugge, 2023
Tool and workflow suggestions	Kemell et al., 2024; Kuhail et al., 2024; Moroz et al., 2022; Stavridis and Drugge, 2023
Code explanation and refactoring	Kemell et al., 2024; Kuhail et al., 2024; Stavridis and Drugge, 2023
Learning	Finnie-Ainsley et al., 2022; Liang et al., 2024; Moroz et al., 2022; Smit et al., 2024

## 2.2.2 Challenges and limitations

This chapter contains a detailed analysis of the limitations and challenges associated with AI tool usage in software development. These limitations and challenges are ones that are often caused by either the way these AI tools are used or by difficulties in integrating them as part of routine workflow. It is notable that most of the caveats in using AI tools can be avoided with sufficient behaviour and standards with AI tools (Finnie-Ainsley et al., 2022). Nevertheless, some of the most common problems are associated with the quality of the content produced by AI.

The most predictable challenge with AI tools is the quality of what they produce (Liang et al., 2024; Mendes et al., 2024; Moradi et al., 2023). The problems with AI tools' output usually requires users to go out of their way and change their workflows to adjust to mistakes made by AI (Liang et al., 2024; Moradi et al., 2023). The study by Liang et al. (2024) mapped out some of the most common problems with AI outputs. This included code that does not meet functional or non-functional requirements and code that is not fully understood by the developer. This in turn has two consequences, either the developer declines the AI code all together or then risks spending more time debugging it (Finnie-Ainsley et al., 2022; Liang et al., 2024; Moradi et al., 2023). To ensure high quality code and sufficient development, Kuhail et al. (2024) point out that human understanding of the code is essential. Moradi et al. (2023) articulate that problems with AI tools' outputs can often be avoided with sufficient prompting tactics. In practice this means giving enough context to AI to produce the code as well as not asking the AI to produce too large chunks of code at a time. Moradi et al. (2023) explain that AI tools work best when building something piece by piece and that they tend to neglect important details when asked to complete unnecessarily big pieces of code at a time. They illustrate this by noting that GitHub Copilot comprehends the bubble sort algorithm and can implement it effectively yet struggles to apply it when the contextual scope becomes overly extensive, even if bubble sort is the optimal choice for the scenario.

Another challenge with AI tools is the quality of their training data (Prather et al., 2023; Wong et al., 2023). As Fjeld et al. (2024) put it, "AI systems are as good as the data they are trained on". This is also at least a partial cause for AI's poor outputs. The two primary concerns with the quality of the training data are security (Coutinho et al., 2024; Mendes et al., 2024; Wong et al., 2023) and technical debt (Mbizo et al., 2024; Moroz et al., 2022). In terms of security, AI tools spark debate on whether they can deliver code that has no

vulnerabilities and handles data in a secure manner (Wong et al., 2023). Another security challenge is to understand what kind of information is safe to share with AI and is the information compromised to leaks if shared with AI (Coutinho et al., 2024). As Mbizo et al. (2024) inform, in most cases it is not clearly disclosed whether AI tools might use human inputs as training materials subsequently compromising any sensitive data shared with AI.

The second major concern with the training data of AI tools is if it is up to date (Moroz et al., 2022). Moroz et al. (2022) clarifies that the varying quality of the training data, which might include open repository data from inexperienced developers, might very well lead into AI tools suggesting deprecated solutions. Mbizo et al. (2024) add that insufficient and deprecated suggestions are likely to cause technical debt requiring maintenance and updating in near future lowering the overall value of the code. For this Moroz et al. (2022) suggest that the training data should have an expiry date after which the AI would not use it as a basis for its suggestions. Wong et al. (2023) even express an idea of “synthetic” training data constructed purely for the purpose of training LLMs for a strict context. On the other hand, depending on the context this would most likely require significant human efforts in creating the training data. Other more minor concerns with AI training data include worries of intellectual property rights and harmful language and ideologies (Moroz et al., 2022).

When working with AI tools, a continuously growing concern is relying too heavily on AI tools outputs instead of one’s own abilities (Moroz et al., 2022; Weber et al., 2024). In other words, this means using AI as the primary source for code production without sufficient reviews. This can result in bugs ending up in production code as developers don’t fully understand their code (Moradi et al., 2023). Another caveat with overreliance in AI tools is ending up in a spiral of lengthy debugging sessions costing more time than coding without AI tools altogether (Vaithilingam et al., 2022; Weber et al., 2024). Contrary to consensus in the field, Moradi et al. (2023) state that the code produced by AI tools is often easier to debug than entirely human written code. The studies by Zhou et al. (2025) and Vaithilingam et al. (2022) on the other hand, show AI tools users reporting a feeling of losing control over their own code. With this the respondents implied that debugging AI produced code ended up costing more time and effort than writing the code on their own. A solution for this problem by Zhou et al. (2025) is a practice where AI is used in either planning or coding but not both. This approach allows developers to either collaborate with AI to brainstorm ideas or oversee that AI-generated code effectively meets its intended purpose.

The growing reliance on or misuse of AI tools in programming education has garnered significant attention. As of now, AI tools can effortlessly complete simple to moderately complex assignments, enabling students to pass introductory courses with minimal effort. However, this approach undermines the development of essential programming skills and conceptual understanding. A common tendency among students learning to code is to prioritize quickly completing assignments over engaging in deep, meaningful learning. In addition to this, teachers or assistants rarely have the time and capability to review every submission to verify they are not AI creations. Thus, programming education is at a turning point where new ways of learning need to be invented. (Prather et al. 2023)

In examining the challenges and limitations of AI tool adoption, it is notable that study participants frequently report difficulties in incorporating these tools into their daily workflows and understanding their effective use (Mendes et al., 2024; Zhou et al., 2025; Weber et al., 2024). The study by Zhou et al. (2025) lists technical problems as the biggest challenge with AI tool usage. This means inconveniences like an IDE incompatible with AI tools like Copilot or AI tools not responding correctly for user inputs. Findings by Kemell et al. (2024) complement this by explaining that users had difficulties in having correct files open to give Copilot the needed context for generating code. They also found out that having IDE open for too long starts to cause issues with the AI tool.

Another difficulty in integrating AI tools as part of daily workflow is to learn how to use them effectively (Mendes et a., 2024; Weber et al., 2024). The study by Weber et al. (2024) points out that less experienced participants struggled to generate well-integrated quality code with AI tools, whereas more experienced users had a clear idea of providing enough contextual information for AI to make it generate optimal code. Another key aspect of effectively leveraging AI tools involves developing the ability to understand when to disregard AI generated suggestions (Kemell et al., 2024). Lastly, something users need to learn with AI tools is effective prompting and how to provide the tools with enough contextual information (Kemell et al., 2024). Kemell et al. (2024) note that users often struggled to determine whether suboptimal AI outputs were a result of the quality of their own prompts.

A technical limitation of AI tools lies in their ability to comprehend and produce code for newer and less used technologies (Pudari and Ernst, 2023;

Zhou et al., 2025). This issue may manifest as "hallucinated" or outdated code elements, such as APIs or libraries, or even entirely non-functional or erroneous code (Liang et al., 2024; Zhou et al., 2025). As Zhang et al. (2023) explain, one of the most used AI tools for programming, GitHub Copilot, is used most of the time with the same technologies and thus, naturally has a larger knowledge base on them. For coding languages these include JavaScript, Python and C# and for technologies almost half of the time Copilot is used for Node.js. This highlights that LLMs struggle to perform effectively with technologies for which they have limited or no training data. Kemell et al. (2024) also name keeping up with the continuous emergence of new technologies as a challenge for AI tools.

Among the various limitations observed in the adoption of AI tools for software development, resistance to change emerged as a significant barrier in certain cases. In one of the pilot groups in the study by Kemell et al. (2024), one participant exhibited apparent scepticism towards LLMs like Copilot, dismissing its autocomplete feature as insufficient for coding and finding its chat functionality inadequate. Even after months of use, they remained unconvinced of Copilot's value, perceiving it as a suboptimal tool (Kemell et al., 2024). This resistance highlights the broader challenge of user adoption, where effective LLM utilization demands effort and context provision, without which the tools' benefits may remain unrealized. While this aligns with other limitations, such as difficulties in prompt crafting or handling newer technologies, data from all the case studies by Kemell et al. (2024) indicate that such pronounced resistance is in the end relatively rare.

A key consideration in the discourse on AI tool usage in software development is the ethical implications of their application. The biggest talking points regarding the ethical aspects are knowing what data has been used for training the AI, understanding the importance of quality control regarding AI produced code, privacy issues as well as harmful biases and data ownership (Akbar et al., 2025; Fjeld et al., 2024; Pashchenko, 2023). As Pashchenko (2023) explains, there are still many unanswered questions regarding the legislation concerning data ownership, intellectual property rights of code as well as author's rights, copyrights and patents. Pashchenko (2023) underscores that as AI tools become more pervasive, appropriate legal frameworks and ethical guidelines are needed to protect both individuals and companies from potential misuse of data and to ensure responsible usage of AI tools. When discussing software engineering research, Akbar et al. (2025) highlight the critical need for responsible AI tool usage, noting, for instance, that customers may be unaware or unprepared for their sensitive information

being shared with AI systems. This is generalizable to all fields of AI use and underscores the ethical responsibility of developers to ensure transparency and safeguard data privacy when integrating AI into software development processes.

Contrary to many other articles, Kemell et al. (2024) also analyse the limitations of AI tools usage on an organizational level. The challenges and limitations encompass previously discussed issues, including data privacy and intellectual property rights, as well as concerns related to the broader adoption of AI tools within the organization such as costs and efficiency. Organisations are naturally interested in evaluating the benefits achieved with buying AI tool licenses. The difficulty in measuring software developers' productivity, as previously discussed, combined with these factors, presents a significant challenge. Consequently, it is inherently challenging to assess which AI tools provide the greatest value in terms of benefits gained relative to their licensing costs. The continuous emergence of new technologies in AI tools as well as programming is also something that concerns organisations when adopting AI tools as a part of developers' daily work. (Kemell et al., 2024)

**Table 2. Summary of identified challenges and limitations in AI tool usage**

<b>Challenge or limitation</b>	<b>Reference</b>
Weak outputs from AI	Finnie-Ainsley et al., 2022; Kuhail et al., 2024; Liang et al., 2024; Mendes et al., 2024; Moradi et al., 2023
Weaknesses in AI training data	Coutinho et al., 2024; Fjeld et al., 2024; Mbizo et al., 2024; Mendes et al., 2024; Moroz et al., 2022; Prather et al., 2023; Wong et al., 2023
Overreliance on AI	Moradi et al., 2023; Moroz et al., 2022; Vaitilingamn et al., 2022; Weber et al., 2024; Zhou et al., 2025
Compromises the learning of junior developers	Prather et al., 2023
Difficulty in adopting, using, and learning AI tools	Kemell et al., 2024; Mendes et al., 2024; Weber et al., 2024; Zhou et al., 2025
AI tools' poor performance with less common technologies	Kemell et al., 2024; Liang et al., 2024; Pudari and Ernst, 2023; Zhang et al., 2023; Zhou et al., 2025
Resistance to change	Kemell et al., 2024

Ethical concerns	Akbar et al., 2025; Fjeld et al., 2024; Pashchenko, 2023
Organizational challenges	Kemell et al., 2024

### 2.2.3 Impacts in productivity

This chapter delves into the perceived effects AI tools have had on developers' productivity. The effects are a sum of the benefits, use cases and challenges related to AI tool usage. AI tools significantly enhance developers' productivity; however, to fully comprehend this complex issue, the various methods by which productivity is improved must be identified (Solohubov et al., 2023).

The most significant and obvious way AI tools improve developers' productivity is by expediting various routine tasks' completion (Liang et al., 2024; Zhang et al., 2023; Ziegler et al., 2024). The first and most obvious way of speeding up development is automatic code generation which is extremely useful in generating repetitive or boilerplate code allowing developers to spend most of their time solving more complex problems within the code (Liang et al., 2024; Pangavhane et al., 2024; Prather et al., 2023). Another way of saving time with AI tools is that they eliminate the need for spending time googling solutions for various problems (Pudari and Ernst, 2023). As Kemell et al. (2024) explain, GitHub Copilot users are able to search for information within their IDE with Copilot. Other ways of speeding up development with AI tools include writing documentation (Moroz et al., 2022), writing tests (Weber et al., 2024), as well as helping to understand code written by others (Mbizo et al., 2024). With sufficient practices in using AI tools, the user can speed up their development avoiding caveats like "the debugging rabbit hole" (Prather et al., 2023). Stavridis and Drugge (2023) observed an approximate 30% increase in code production velocity with AI tools, whereas Weber et al. (2024) found that participants using GitHub Copilot accelerated all their daily work tasks by 35% overall.

The usage of AI tools also offers support in productivity from the social standpoint (Vaithilingam et al., 2022; Zhang et al., 2022). As previously discussed, Yilmaz et al. (2016) and Graziotin et al. (2013) explain that the productivity of a developer also has a social dimension where pleasure and positive mindset in everyday work play a major role in one's productivity. Zhang et al. (2023) investigated developers' attitudes toward using AI tools in their work, revealing that developers find greater enjoyment in their tasks when using AI tools, with GitHub Copilot eliciting the most positive responses among the tools evaluated. Vaithilingam et al. (2022) and Prather et

al. (2024) found that developers enjoy AI tool usage to such an extent that they are willing to employ them despite occasional erroneous suggestions, which can lead to extended debugging sessions. Pudari and Ernst (2023) propose that AI tool usage may alleviate developers' pressure to deliver code quickly, yielding long-term benefits for their mental well-being and social stance towards work. Smit et al. (2024) reported that 80 % of respondents mentioned an increase in satisfaction towards working with AI tools

AI tools can enhance productivity by recommending technologies and facilitating the expansion of an individual's knowledge base (Coutinho et al., 2024; Liang et al., 2024; Stavridis and Drugge, 2023). As Faraj and Sproull (2000) emphasize, the critical role of expertise in software development, as a form of knowledge work, underscores the value of AI tools, since expertise in programming can lead to substantial improvements in program efficiency, potentially by order of magnitude. AI tools have the capability to aid one's individual learning of new technologies, coding languages or techniques (Liang et al., 2024). Furthermore, this allows one to become more efficient in their daily work through improved expertise, thus improving productivity. Moroz et al. (2022) highlight the learning advantages of AI tools, noting that they also enable individuals in junior roles to more effectively adapt to their work responsibilities. AI tools can substantially support inexperienced developers in learning and enhancing their coding skills, though they also place significant responsibility on these developers to use the tools appropriately to avoid hindering their own learning progress by relying too much on these AI tools (Mbizo et al., 2024; Prather et al., 2023).

Another way to look at improved productivity is that AI tools help developers produce higher quality solutions in an efficient manner (Zhang et al., 2023). As software quality is nowadays seen as a major factor in project success (Maurer and Hellmann, 2013), efficient quality code production can be seen as part of productivity. AI tools are capable of analyzing requirements and forming tests for code based on them (Kemell et al., 2024; Moroz et al., 2022). This allows developers to design tests before writing the actual code facilitating test driven development - a framework commonly associated with high quality code and good coding standards (Kemell et al., 2024; Moroz et al., 2022). Other previously discussed ways of improving project quality with AI tools include writing documentation and refactoring old code to fit modern quality standards (Mendes et al., 2024).

Stavridis and Drugge (2023) propose that, given adequate contextual information and sufficiently detailed requirements, artificial intelligence may

eventually possess the capability to autonomously develop comprehensive systems. Unlike most of the scientific literature regarding AI tool usage in software development, the study by Özpolat et al. (2023) evaluates ChatGPT as an individual operator to create an enterprise resource planning (ERP) solution all by itself. The common approach in scientific studies is to analyze AI tools, mainly Copilot, in programming but this study utilized AI capabilities in each step of the software life cycle. Özpolat et al. (2023) investigated the application of ChatGPT across five distinct domains: requirements elicitation and analysis, system design, programming, testing, as well as deployment and maintenance guidance. Their study employed a series of targeted questions to steer ChatGPT toward optimal performance in these areas. Despite some challenges this study highlights that AI can work extremely well in all software engineering domains, not only programming. Lastly Özpolat et al. (2023) suggest that AI tools could have an even larger impact in non-coding software engineering activities compared to the aid they provide in coding. The takeaway from this study is that AI tools are already capable of creating a working large-scale system with the guidance of an experienced software engineer. This highlights that AI tools still have huge amounts unrealized potential yet to be studied.

Scientific literature highlights the transformative role of AI tools in enhancing software developers' productivity by optimizing workflows, making work more engaging, and improving software quality. By automating routine tasks, increasing job satisfaction, and supporting personal development, these tools enable developers to work more efficiently and creatively (Kuhail et al., 2024). Furthermore, AI tools contribute to producing higher-quality code and align with modern development's best practices (Zhang et al., 2023). They also have the potential to influence all phases of the software development life cycle allowing for new and innovative use cases (Özpolat et al., 2023). Collectively, these advancements underscore AI tools' capability to redefine productivity in software engineering, balancing efficiency with meaningful developer engagement.

**Table 3. Summary of identified productivity benefits**

Productivity benefit	References
Speeding up work tasks	Kemell et al., 2024; Liang et al., 2024; Mbizo et al., 2024; Moroz et al., 2022; Pangavhane et al., 2024; Prather et al., 2023; Pudari and Ernst, 2023; Stavridis and Drugge, 2023; Weber et al., 2024; Zhang et al., 2023; Zhou et al., 2025

Increase of comfortability and stress release	Prather et al., 2023; Pudari and Ernst, 2023; Smit et al., 2024, Vaithilingamn et al., 2022; Zhang et al., 2023
Increase in knowledge base	Coutinho et al., 2024; Liang et al., 2024; Mbizo et al., 2024; Moroz et al., 2022; Prather et al., 2023; Stavridis and Drugge, 2023
Increase in quality of work results	Kemell et al., 2024; Mendes et al., 2024; Moroz et al., 2022; Zhang et al., 2023

### **3 Research methods**

The research approach I chose for this thesis is case study. As Yin (2009) states in his book “Case Study Research: Design and methods”, case studies are often suitable in situations where the questions asked are “how” and “why”. In their article about case study guidelines, Rueson and Höst (2009) add that, case study is also a suitable research method in situations where a contemporary phenomenon is studied in its natural real-life context. This study aims at understanding and explaining the capabilities of new AI assistant tools and technology. It is very typical that these types of situations are approached with a case study (Perry et al., 2006).

This chapter defines the studied case. The study setting, scope and research methodology are also explained in detail. Furthermore, this chapter explains the data collection methods for this study as well as provides clear explanations on why these methods were chosen. Lastly, this chapter explains how the collected data is analysed.

#### **3.1 The case**

The unit of a company in my case study, referred to as case company, is a company founded in 2016 that develops robotic process automation (RPA) for its customers. Later, in 2022 the case company became a unit of a larger Finnish IT company. In short, the case company develops solutions for customers to automate repetitive and monotonous tasks to allow employees to work with more complex tasks. This case company employs 40 developers. The developers are divided into development teams but most of the work is done individually. Overall, the case company consists of a diverse pool of professionals educated in a multitude of technologies and methodologies used in the field such as Kanban, Scrum and Python programming.

The case company produces RPA solutions for Finnish customers with repetitive business activities that can be automated using open-source technologies and frameworks. A typical workflow of a project in the case company is such that first the task to be automated is defined clearly enough that each step can be automated in code level. Then once the definition and a cost estimation are approved by the customer, the development work can begin. A vast majority of these projects are ones that had just one developer working on them individually. Finally, a project is logged complete once a test run in production environment is passed.

Technologically all these projects are implemented at least partially with Python and its many open-source libraries like Selenium, pywinauto and PyAutoGUI. The responsibility for the technological implementation and overall code architecture rests with the assigned developer, however, a code plan review is conducted by another developer who is not otherwise involved in the project. Most of the time these projects are developed by only a single developer.

The case company has a carefully planned out strategy, part of which is to utilize AI tools to their full potential in everyday work to improve working productivity. Part of this strategy was introducing GitHub Copilot as part of developers' daily workflows in January of 2024. Copilot was first tested with a test group of 10 developers for approximately a month. After that it was introduced to company wide use in a monthly developer info meeting. Developers in the case company are advised to utilize Copilot as much as is needed to increase productivity.

All in all, this provides an excellent case for my study as it allows me to research the different techniques regarding AI tool usage and their effects on project metrics and developers' efficiency. The activities the employees in the case company face, range from occasional definition tasks and planning to outright programming. Furthermore, since developer productivity is influenced by different contextual factors and backgrounds, the various tasks ranging across multiple industrial sectors enables me to evaluate productivity in a diverse setting, strengthening the study's reliability.

### **3.2 Case study scope**

The scope of the case study, or the case itself, is the effects of an AI tool's, GitHub Copilot's, usage has had on developers' daily work and productivity as well as project metrics in the case company. The diverse nature of the case company's developers' tasks provides interesting insight into Copilot's capabilities in designing entirely new solutions in comparison with their capabilities in helping developers in expanding existing systems with new or changed requirements. I have decided to include all willing participants from the case company who work with development related activities. All these individuals have access to GitHub's Copilot, which is the focus of this study in terms of AI tools. In addition, many free LLM based tools like ChatGPT are available for these individuals. This set of willing participants helps me in obtaining meaningful data to analyse the AI tools perceived effects on productivity. Furthermore, the participants allow me to analyse and link

different use cases for AI tools with different ways of increasing productivity. This scoping helps to understand the effects of AI tools in developers' everyday work and their linkage with perceived productivity, which is the objective of this case study.

The case company employs expert judgment as their main effort estimation technique. It is usually carried out by a project manager who carefully evaluates the project. Furthermore, project managers usually also consult developer's opinion on the estimate to make sure they are satisfied with it. Archival Jira data could potentially offer insights into the accuracy of these estimations, though researching this is not the intent of this study.

### **3.3 Research methodology**

This study aims at understanding the different use cases for AI tools in software development as well as understanding their effects in productivity and efficiency. This means that this study is researching contemporary phenomena in their natural context which is widely used as a description for when a case study could be used (Runeson and Höst, 2009; Wohlin, 2021; Yin, 2009). Yin (2009) also adds that a typical characteristic in case study settings is that behavioural events cannot nor should not be manipulated. This holds true for this study as AI tools usage in software development and its effects are indeed a complex phenomenon. In the study scope there is no manipulating the behaviour events to guarantee unbiased and truthful analysis of Copilot usage among developers.

Following Yin's (2009) definition, my case study is an embedded single case study. This denotes that a single case is studied throughout multiple data collection methods. In my study this materializes through the questionnaire, archival analysis and observations of actual working situations. Subsequently, the collected data is triangulated to ensure the validity of the findings and to facilitate a comprehensive analysis of this complex phenomenon increasing the construct validity of this study. A hypothesis for this study is that the introduction of GitHub Copilot has increased developers' productivity.

The topic of AI tool usage in software development is, as mentioned, a complex phenomenon that depends on multiple contextual factors such as developers' work experience, nature of job and available tools. This implies that this research should be done with empirical methods (Runeson and Höst, 2009). An analytical paradigm does not fit my study as the case is a complex real-life situation where human factors and interaction between technology

and humans play a large role (Runeson and Höst, 2009). To rule out other options of research methods, controlled experiment is also not suitable for this study as the phenomenon in this study is very hard or impossible to study in isolation (Runeson and Höst, 2009). Especially the aspects of effects in productivity are deeply connected with the project development context.

A possibility of researching how my proposed best practices with AI tools would work in a real-life context is intriguing but I have decided to leave it out of the scope of this study to keep it more manageable for a master's thesis. Further research possibilities along with other discussion can be found in chapter 5.

### **3.4 Data collection**

Following the defined case study methodology, multiple data collection methods were employed to collect data from various viewpoints. This data includes questionnaire with open and closed ended questions, observations of daily working routines with GitHub Copilot and archival analysis of Jira data from previous projects. The data was collected in two phases where the questionnaire was the initial phase and observations as well as archival analysis followed in phase two. This was done to ensure maximum usefulness for the observations and to understand how to best analyse the Jira data.

#### **3.4.1 Questionnaire**

The primary data collection method for this study is a questionnaire on daily working habits regarding the usage of GitHub Copilot. This data collection method was chosen as I saw it as the most efficient way of gathering as much data from the case company's developers as possible, whilst not making the research process excessively long and time consuming. The questions were designed to get insight from developers regarding their Copilot usage from multiple viewpoints. These included use cases, effects on productivity as well as challenges and limitations. The questionnaire also included various questions yielding quantitative data. Out of these, the questions 14 to 16 were the same as in the study by Kuhail et al. (2024) to gain comparable data between the studies. These questions mapped out respondent's opinions on Copilot's trustworthiness and effectiveness. The questionnaire invitation was sent to all 40 of the case company's developers. The entire questionnaire is provided as appendix 1.

### **3.4.2 Observations**

Observations were conducted to gain deeper insights into the real-world application of GitHub Copilot in software development within the case company. While the questionnaire provided a broad overview of developers' experiences and routine practices, direct observations allowed for a more nuanced understanding of how Copilot is integrated into daily work. Furthermore, they helped in understanding how the perceived productivity is achieved through concrete practices with utilizing Copilot. By closely monitoring developers in their working environments, I was able to analyse their interactions with AI pair programming tool in Copilot and assess the extent to which this tool influenced coding efficiency, problem-solving approaches, and general productivity.

The observational phase focused on key behaviours, including how often developers used Copilot, the nature of tasks where Copilot was most beneficial, and the ways in which Copilot shaped development practices. Additionally, these observations shed light on limitations and challenges with Copilot usage. The participant selection was based on the AI use cases and limitations reported in the questionnaire. This way it was possible to select participants with different routines with Copilot. The number of developers willing to participate also limited the number of observations. The observations were held as a Teams call where the entire session was recorded for further analysis. Participants were directed to perform their tasks as if the observer was a trainee who they were introducing to daily workflows explaining their rationale for using Copilot each time they utilized it.

### **3.4.3 Archival analysis**

The archival analysis method was selected to determine whether developers' perceived changes in productivity are reflected in actual project metrics such as delivery times and relative effort estimation error, a metric presented in the study about effort estimation errors by Jørgensen and Molokken-Ostvold (2005). Relative effort estimation error describes the proportion by which the actual required working hours deviated from the estimated effort. In the article by Jørgensen and Molokken-Ostvold (2005) the used metric is mean relative error but, in this context, it is more interesting to calculate the relative error for each Jira task's effort estimation and then visualize that as a box plot dividing the dataset into times before and after using Copilot. This is done to analyse also metrics like median instead of purely focusing into the mean relative error.

As many of the potential metrics used to measure developers' productivity, introduced in the study by Oliveira et al. (2020) are not tracked in the case company, the idea is to measure project lead times and relative error for effort estimations as metrics for developer productivity. The motivation for the use of these metrics is that if the introduction of GitHub Copilot has increased developers' productivity, projects would not last as long and would not require as much effort in comparison to the effort estimations. The analysed archival data includes information on the original effort estimates, actual working hours spent and key project milestones such as start and completion dates.

The projects on the case company are presented as tasks in Jira. These tasks contain relevant information such as project effort estimation and logged working hours. These tasks have several different types but the relevant types for this study are "RPA task" and "RPA expansion". Essentially these are projects where either a new RPA solution is created, or an existing one is significantly changed. The tasks also contain a status of the work such as "in progress" or "closed". These are values that are logged to Jira task cards. The tracking of project metrics is mostly manual. Details like used hours, effort estimation and task type are manually logged values. The logging of project start and end dates can also be considered manual as they are logged when a Jira task is created and when it is closed respectively. The logging of working hours is done weekly, and the effort estimations are logged to Jira tasks before the development phase of the project can start

All this data was collected through Jira systems own API, where data from all tasks is available in Json format. This allows the project data to be easily handled and analysed with Python where the data is easy to display with several open-source libraries supporting various formats in displaying numerical data.

### **3.5 Data analysis methods**

This sub-chapter explains how the collected data was analysed. The gathered data is both qualitative as in open questions from the questionnaire and observations as well as quantitative from the archival analysis and numerical questions from the questionnaire. This is done to ensure strong analytical paradigm in data analysis (Yin, 2009).

### **3.5.1 Questionnaire**

The questionnaire yielded both qualitative and quantitative data from Copilot usage in the case company with open and closed ended questions respectively. In both, the empty or “unable to answer” answers were removed from the analysed data. The qualitative answers were then coded thematically and analysed for reoccurring themes. For the quantitative questions, which were adapted from prior studies, the means and standard deviations were calculated and subsequently compared to the results of the original study in which they were used. In these five step Likert scale questions the strongly agree option was counted as five and strongly disagree as one.

### **3.5.2 Observations**

Instead of traditional observation techniques like writing notes directly after the observation, I saved the observation session as a Teams call recording and carefully analysed the results later to understand the motivation behind each Copilot use case as well as the benefits the participants perceived. The recordings were analysed carefully through the viewpoint of the themes from the thematic analysis of the qualitative questionnaire answers. This was done to gain deeper understanding of Copilot usage through practical use cases in actual programming tasks. I also analysed the participants’ behaviour to recognize any problems in Copilot usage. This provided a more comprehensive understanding of Copilot usage than the relatively short answers from the questionnaire.

### **3.5.3 Archival analysis**

The data analysis of the archival data was very straight forward with basic data analysis methods in Python code. The key idea behind this archival analysis was to analyse the projects’ lead times, and relative error for effort estimations for the Jira tasks. The data was analysed comparing time before the introduction of Copilot and after it. Initial plans for data analysis included separating the evolution of relative effort estimation error based on the Jira task type but that was excluded from the analysis because the final dataset contained only 16 entries for “RPA Expansion” tasks. The archival analysis was done to better understand whether these project metrics show improvements after the introduction of Copilot.

The vast amount of project data from previous projects posed a challenge in filtering erroneous data out of the dataset. The initial retrieved dataset

contained 571 data points some of which had one or more key features missing. These data points were removed from the dataset as, for instance, it is impossible to measure the relationship of realized and estimated effort if realized effort in task's logged hours is missing. The final dataset had 118 data points. The earliest data point in the dataset dates to November 2022, while the most recent one was recorded in May 2025. Within this range, the pre-Copilot period accounts for about 13 months, and the post-Copilot period makes up roughly 17 months. Table 4 summarises the analysed project metrics and their relation to features of the dataset

**Table 4. Definitions of project metrics**

<b>Project metric</b>	<b>Calculation formula</b>
Relative effort estimation error	$\frac{\text{Logged working hours} - \text{Estimated working hours}}{\text{Logged working hours}}$
Project lead time	Project end date – Project start date

## 4 Results

This chapter presents the empirical data collected through the case study, forming a critical foundation for the subsequent analysis and discussion. The data and findings are systematically organized into three distinct chapters, each corresponding to a specific data source: questionnaire responses, observational findings and archival Jira records. This triangular structure facilitates a comprehensive examination of the diverse set of evidence, ensuring clarity and coherence in the presentation of the study's outcomes.

### 4.1 Questionnaire

The questionnaire results are presented in this subchapter. First section 4.1.1 covers the results from the quantitative questions of the questionnaire. The results of the thematic analysis of the qualitative questions responses are presented in section 4.1.2. The questionnaire was deployed for a month and out of the 40 developers in the case company, 16 answered. In the beginning of the survey, the respondents were asked about their demographic and working experience in software development and programming in general including personal projects. After the demographic and working history related questions, the respondents were asked about their experiences with Copilot as well as their trust towards the help it offers.

#### 4.1.1 Quantitative answers

The demographical questions at the beginning of the questionnaire identified that 12 of the respondents were software robotics developers, whereas 2 reported working as standard software developers and the final 2 reported being lead developers in the case company. In addition to their job description, the respondents were also asked about their technical expertise within and outside the scope of their job description. The most common programming skills outside the scope of the job description were full-stack and web development skills. Table 5 presents the working experience and overall programming experience of the respondents.

**Table 5. The experience of the respondents**

	<b>Working experience in software development</b>	<b>Personal programming experience</b>
< 2 years	5 (31.2 %)	0 (0.0 %)
2 to 5 years	8 (50.0 %)	8 (50.0 %)

6 to 10 years	3 (18.8%)	7 (43.8 %)
11 to 16 years	0 (0.0 %)	1 (6.2 %)
> 16 years	0 (0.0 %)	0 (0.0 %)

**Table 6. Q7: How often do you use Copilot**

<b>Response option</b>	<b>Amount (%)</b>
Several times a day	9 (60.0%)
Daily	2 (13.3%)
Weekly	4 (26.7 %)
Monthly	0 (0.0 %)
Not at all	0 (0.0 %)

**Table 7. Q8: Copilot has had positive effect on your productivity**

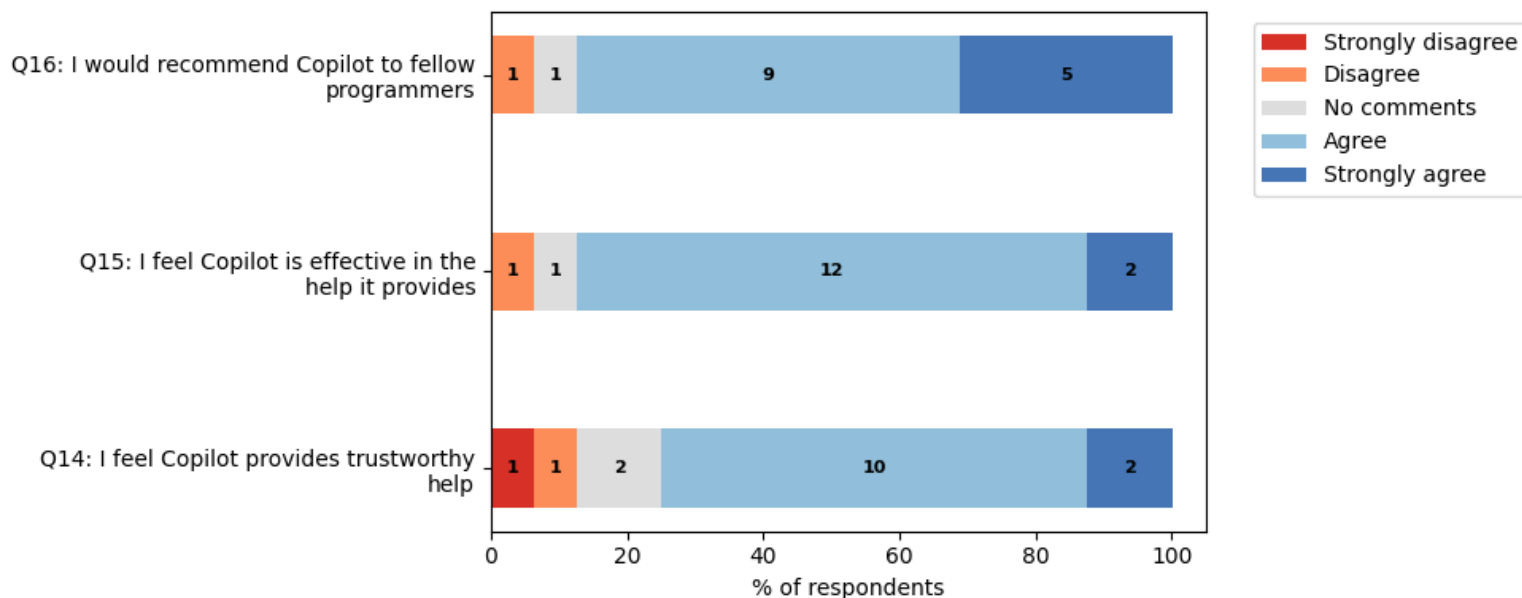
<b>Response option</b>	<b>Amount (%)</b>
Very positive	6 (37.5 %)
Positive	7 (43.8 %)
Neutral	3 (18.7 %)
Negative	0 (0.0 %)
Very negative	0 (0.0 %)

**Table 8. Q11: Have you benefited from using Copilot**

<b>Response option</b>	<b>Amount (%)</b>
Strongly agree	7 (43.7 %)
Agree	7 (43.7 %)
No comments	2 (12.5 %)
Disagree	0 (0.0 %)
Strongly disagree	0 (0.0 %)

The next section of the study focused on identifying the different use cases and limitations associated with Copilot usage. It also explored respondents' perspectives on how Copilot has influenced their work productivity, the frequency of its use, and the extent to which they have benefited from it. In conclusion, questions 7, 8 and 11 show that the respondents mostly use Copilot

in a daily basis and feel like having benefited from it and having become more productive with it.



**Figure 2. Respondents trust in Copilot**

Figure 2 presents the spread of the answers from the questions regarding trust in Copilot from the study by Kuhail et al. (2024). All in all, the respondents reported relatively high trust in AI tools. The results show that the respondents showcased high trust in AI tools with mean answer being 3.92. The standard deviation for the answers was also relatively small being 0.84, meaning that the responses were relatively consistent.

#### 4.1.2 Thematic analysis

The answers from the qualitative question 9, 10, 12, 13 and 17 were coded with the following codes: Copilot use case, Effects in productivity, Challenge in copilot usage, Reason to avoid Copilot usage and improvement idea. These codes were primarily derived from the content and intent of questions 9, 10, 12, and 13, which directly prompted participants to provide responses aligned with these categories. For example, the code Copilot use case corresponds closely with question 9, which asked participants to “list concrete ways of how and why you use Copilot in your daily work.” However, responses to other questions also occasionally included relevant insights that fit this and other codes. With the analysis of the following codes, several themes emerged. These themes and their frequency are presented with examples in tables 9 to 12.

**Table 9. Use cases for Copilot**

<b>Theme</b>	<b>Frequency</b>	<b>Example</b>
Generating boilerplate code	12	“I allow copilot to autofill things such as function-content, docstrings, queries etc. and modify the results as needed”
Generating logic code	9	“Copilot alleviates cognitive tasks, e.g., completing a demanding algorithm”
Searching for information	8	“I often ask copilot something instead of going to google/documentation”
Code review	7	“I use GitHub's Copilot PR review. Copilot can also provide the feedback faster than humans”
Learning or helping with new things	5	“Figuring out syntax for new modules/libraries”
Debugging	5	“Debugging errors, by giving a code snippet and the error message”
Writing documentation	5	“Creating documentation for code”
Writing tests	5	“Writing unit tests, generating test data”
Brainstorming ideas	3	“Brainstorming about how to implement certain features”
Explaining code	3	“I can ask for an explanation why a solution works with an immediate response”

The most common use case for Copilot in the case company is generating both logic and boilerplate code based on the responses. Other major use cases were code reviews, searching for information, learning new things and helping at different stages of project development.

**Table 10. Copilot's effects on productivity**

<b>Theme</b>	<b>Frequency</b>	<b>Example</b>
Time saving	9	“It has increased my productivity by taking care of most of the trivial work leaving me the crucial core design and decisions”
Eliminates monotonous work	8	“It gives your own brain some more capability, when you don't need to search for every small thing yourself”
Increases working motivation	5	“It affected my productivity in a positive manner and has improved my work satisfaction as it helps me to meet my deadlines and supports my work”

Helps increasing knowledge base	4	“Taking up new skills or libraries feels a bit faster now”
---------------------------------	---	--

Aside from the themes, the question of “Explain how Copilot usage has affected your productivity or mindset in working? Why?” also elicited several answers such as “positively” with no explanation or motivation. Only one respondent reported negative effect on productivity that being that working independently feels significantly more difficult after using copilot for long periods of time.

**Table 11. Challenges regarding copilot usage**

Theme	Frequency	Example
Insufficient code generation	15	“Copilot suggests deprecated libraries and strategies that don't really work”
Difficulties in taking responsibility or fully understanding Copilot's generations	7	“I do not trust the answers If Copilot gives me code and I know too little to verify its safety or correctness”
Usage takes too much time and impairs individual thinking	5	“Copilot just does not always serve you with your way of working and sometimes doing without it is just easier”
Needing to learn how to better use Copilot	4	“Sometimes I may have given up on trying to use copilot for some operations due to lack of prompting skills”
Leads to long debugging sessions	4	“Using Copilot's code suggestions has many times lead to me debugging what feels like someone else's code, when I could have written the same code myself with less effort”
Contextual issues	3	“Copilot does not always understand the problem definition if it is complex”

By far the most common challenge in using copilot was insufficient code generation meaning that Copilot was not capable of producing good enough code for the task at hand. Among the more frequently reported challenges were users' discomfort with utilizing the code generated by Copilot, as well as instances where its use disrupted the development workflow and resulted in

time costs. The less common challenges included users not being familiar with and needing to learn how to use Copilot effectively as well as fearing that using it would lead to long debugging sessions. Lastly 3 respondents reported that a challenge with using Copilot is the technological context of the case company's solutions. In concrete this means that Copilot does not know the entire technological context it is working in and thus cannot produce valid code suggestions.

**Table 12. Reasons to avoid using Copilot**

Theme	Frequency	Example
Not trusting Copilot's capabilities on the current task	11	"Often times Copilot might not work the way you wish, so there is still lots of double checking of results"
Fearing long debugging sessions	5	"Sometimes the use of Copilot might take up more time than coding without it, because I get stuck refining my code and continuously checking whether my solution is the best possible"
Existing better tools for the use	3	"Its coding capabilities are far behind tools like Cursor I use for my free time hobby projects. Also, I feel ChatGPT as a tool far better than Copilot"
Fear of becoming dependent on AI	3	"I am too reliant to its use and my own coding skills and technical confidence has begun to weaken"
Contextual matters	3	"Virtual machines are a bit slow for copilot use"
Not knowing how to effectively utilize copilot on the current task	3	"I struggle to explain issues to Copilot"
Security concerns	3	Avoid using copilot "when dealing with sensitive customer data"
Copilot interrupting otherwise good workflow	2	"The inline completions would always kind of interrupt my own train of thought"

The reasons to avoid using Copilot showed similar themes to challenges in its usage. The most common ones were the previously mentioned not trusting Copilots capabilities and fearing that using it would lead to long debugging sessions. Some previously unmentioned themes included security concerns, having better alternative tools for the task and the fear of becoming too dependent on Copilot and AI in general. In examining the reasons for not using

Copilot, it is noteworthy that six respondents saw little to no reason to avoid using Copilot.

The questionnaire did not really elicit many ideas on how to improve the usage of Copilot except that GitHub is bringing Model Context Protocol (MCP) servers to be part of Copilot's offering. Two respondents mentioned this as an interesting prospect.

## 4.2 Observations

The data collected through observations of Copilot usage in daily work was analysed through the viewpoint of the questionnaire answer codes of Copilot use cases, limitations and reasons to not use it as well as effects on productivity. In total I sent invitation to 5 potential participants, but in the end, there were only two willing developers participating to these observation sessions lasting one hour each. These participants were relatively experienced developers having also reported being familiar with GitHub Copilot in the questionnaire. Overall, the observations helped in understanding and recognizing the codes in the natural setting during actual work.

The most common observed use case for Copilot was automated code generation. It was utilized for several different purposes. Participant 1 used Copilot's automated code generation for creating boilerplate code and autocompleting short code snippets with Copilots suggestions, whereas participant 2 was a little more hesitant in utilizing automated code generation. They used it for generating boilerplate code into an empty file, but they did not let Copilot edit actual code files but rather inspected Copilots suggestions in a separate chat window.

The participants also utilized Copilot for code refactoring, cleaning and documentation. They justified this by noting that such tasks are relatively simple and well-suited to Copilot's capabilities, allowing them to complete repetitive and less engaging work more efficiently. Participant 1 had also configured a custom shortcut in their editor to use Copilot in creating commit messages. They also mentioned that shortcuts to these actions are crucial in saving time. Participant 2 expressed that Copilot has the ability to significantly increase the quality of the code as most of the time they only need to write the starting sequence of a doc-string and Copilot already offers one with high quality description of the function or method.

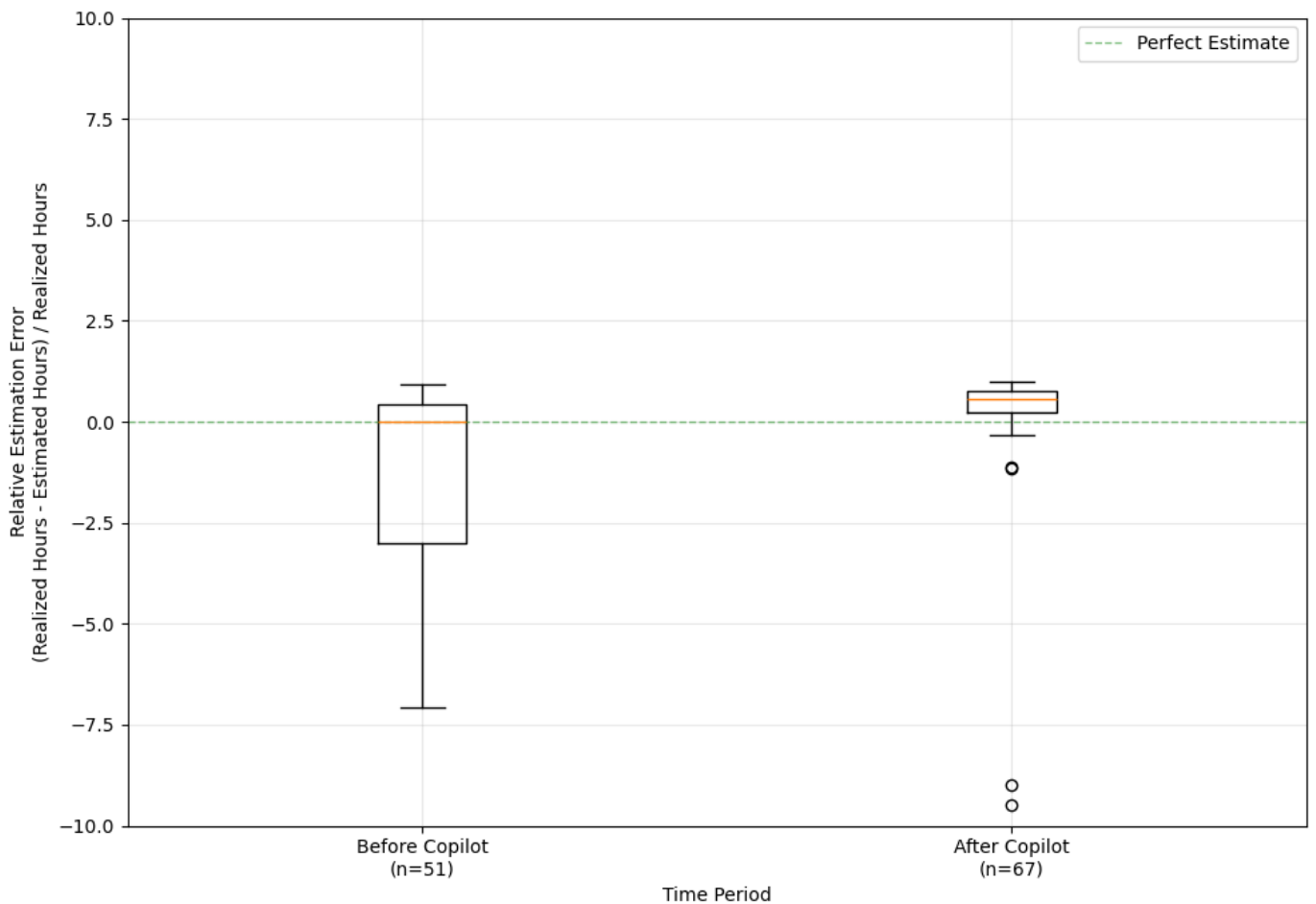
Participant 2 used Copilot to generate test data for their code. While doing this they voiced a concern of double or triple debugging Copilot's outputs. With this they meant situations where the actual code is generated with Copilot and the test and possibly even the test data is also created with it. This leads into situations where the developer needs to figure out whether the bug lies in the actual code or in the tests if everything is generated by Copilot. Participant 2 explained that this could lead to extremely long debugging sessions and was also concerned that using Copilot like this would lead to developers losing control of their own code.

For challenges, participant 1 did not really express any, but rather expressed concern related to Copilot usage as well as providing examples when they prefer not to use it. In the beginning they explained that in more challenging tasks they prefer to use their own personal licence to an LLM provided by Anthropic named Claude, which they currently see as the most prominent LLM for coding. They explained that Claude seems to be more effective in solving coding related problems than the LLMs available for Copilot. They also stated that utilizing the LLM in the browser was as fast as opening a chat window inside their code editor. Both participants highlighted a cautious approach toward AI tools, expressing limited trust in their output and preferring to perform problem-solving independently rather than relying on AI-generated suggestions. Participant 1 also expressed concern about the excessive use of AI tools for overly simple tasks, arguing that it is inefficient given the computational resources and energy consumption involved

The amount of observed productivity effects is limited by the length of the observation session. No major effects can be observed in only one hour, however some considerations regarding productivity were made. Participant 1 demonstrated that having a well-defined framework for integrating AI tools into their workflow enabled them to accelerate the completion of work tasks. All in all, participant 1 had clear use cases for Copilot and knew its limitations. This allowed them to streamline their work without having to continuously evaluate the contexts in which using Copilot would be beneficial. Participant 2 on the other hand, showcased that productivity can be significantly increased when AI instantaneously takes care of easy and monotonous parts of the coding like doc-strings, imports and logging.

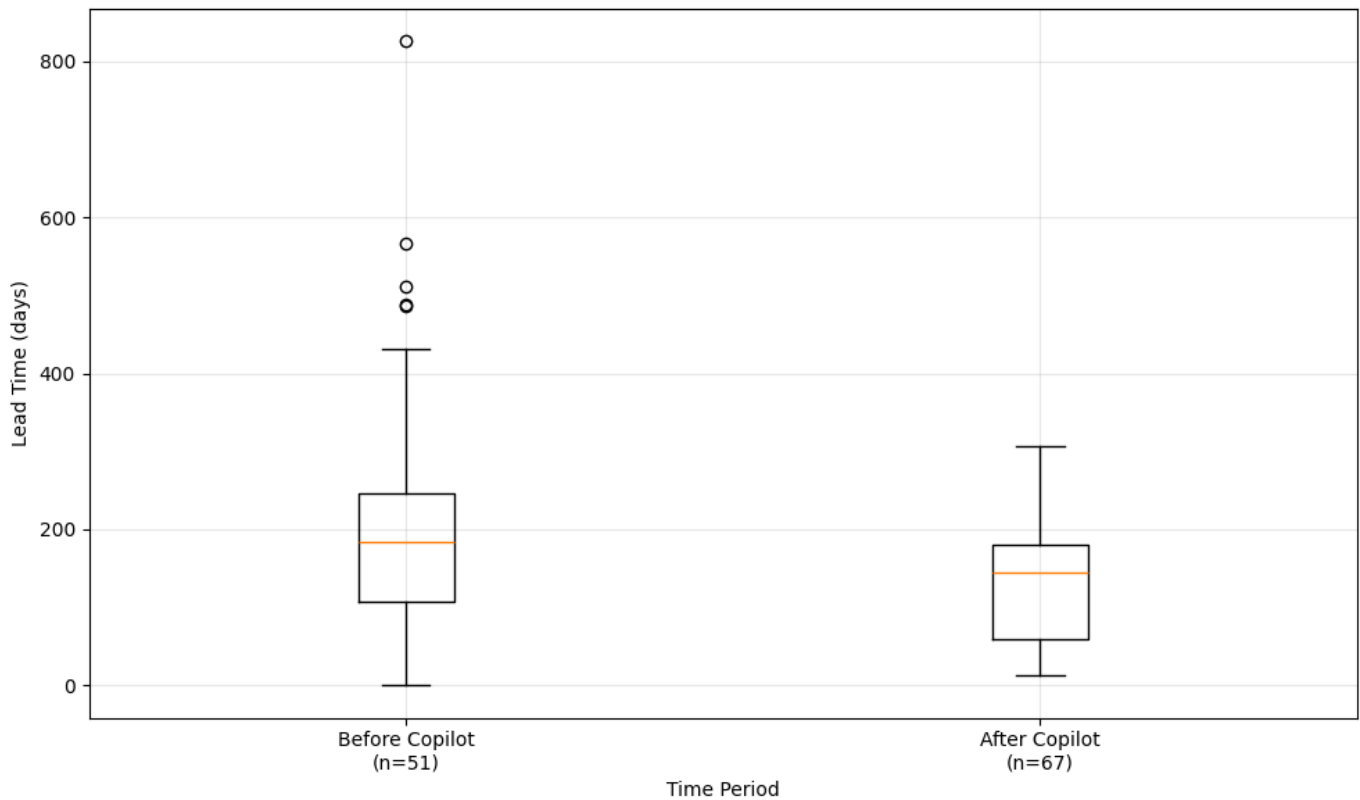
### 4.3 Archival Jira analysis

The examination of archival Jira data produced a range of findings. The retrieved dataset's data points represent Jira tasks with key details including estimated effort, logged working hours, task type, as well as task start and completion dates. Entries missing these attributes were excluded from the analysis. Additionally, data points with unusually high manually logged values, such as logged hours exceeding 50 times the estimated effort, were also removed to ensure data reliability. The remaining dataset contained the information of 118 Jira tasks. The data was visualized by creating a boxplot of relative effort estimation errors before and after the introduction of Copilot. This visualization is presented in figure 3. Figure 4, on the other hand, illustrates the project lead times before and after Copilot usage.



**Figure 3: Box Plot of relative estimation errors before and after the introduction of Copilot**

Figure 3 illustrates the change in relative effort estimation errors from time before Copilot to time after integrating it to daily workflows. This figure illustrates that contrary to the hypothesis of relative effort estimation errors decreasing, the median relative error has increased by 0.561 from -0.011 to 0.550. Normality tests indicated that the data distributions for both periods deviate substantially from normality, with notable skewness and kurtosis. The skewness for the dataset from time before using Copilot is -1.084 with kurtosis of -0.373, whereas the dataset from the time of using Copilot has skewness of -4.643 and kurtosis of 20.947. These non-normal distributions justify the use of a non-parametric Mann-Whitney U test. Overall, the results indicate that, rather than improving estimation accuracy, and thus developer productivity, the integration of Copilot appears to have been associated with greater estimation variability and inflated relative errors across tasks. A Mann-Whitney U test was used to assess the statistical significance of this change, yielding a p-value below 0.001, which confirms that the difference is statistically significant. The effect size, calculated at -0.694, suggests a medium practical effect.



**Figure 4: Box plot of project lead times before and after the introduction of Copilot**

Figure 4 presents the mean project lead times before and after the introduction of Copilot. This means the number of days between the creation and closure of Jira tasks. The data presented in figure 4 shows that median project lead time has decreased by ~21.8% after the introduction of Copilot. The project lead time datasets were also non-normal with pre-Copilot dataset having skewness of 1.515 and kurtosis of 2.823 and post-Copilot dataset having skewness of 0.193 and kurtosis of -0.913, again justifying the usage of Mann-Whitney U test. The Mann-Whitney U test shows that the p value for this change is 0.003, indicating once again a statistically significant change. The effect size for this change is 0.709 indicating a medium effect size.

## 5 Discussion

This chapter answers the research questions and encourages discussion around them. The answers are formed by triangulating the findings of the case study and comparing them to the findings from the literature review. This chapter also offers recommendations for best practices for AI tool usage within software development. Areas for future research are also discussed.

### 5.1 AI tools' impact on developers' work

The emergence of AI tools has significantly changed the daily work of software developers. Using AI tools comes with both benefits and limitations. The benefits of AI tools extend to both the practical aspects of work as well as individuals' mindsets toward their work. The limitations, however, are not necessarily rooted in the technical capabilities of the AI but rather stem from improper usage. An essential part of avoiding the challenges in AI usage is to know when it is appropriate or meaningful to use it.

The findings from the case study and the literature review show largely similar findings in terms of AI tool use cases and their limitations and challenges. In particular, the use cases were mostly the same in both. The only item identified in literature review that was not present on the case study findings was AI suggestions on tools and workflows. In both the literature review and the case study, the main use cases for AI tools were mostly coding related activities like code generation, reviewing or improving code and creating tests and documentation. Also, information retrieval was mentioned as a major use case.

Analysing the different challenges of AI tools usage showed that the participants of the case study were more concerned over the technical capabilities of the AI tools, whereas the literature review voiced more concerns of information security and ethical usage of AI tools. This might be because the case study was scoped to handle only GitHub Copilot, which is designed with organizational information security measures that prevent data from being shared externally. In contrast, several AI tools examined in the literature review were open-source and lacked comparable guarantees regarding data protection. Given these constraints, participants encountered greater difficulties in integrating Copilot into the organizational context compared to the more flexible integration scenarios described in the literature.

Comparing the questions related to trust towards Copilot borrowed from the study by Kuhail et al. (2024) showed similar results between the case study and scientific literature. The respondents mostly trust Copilot and would be willing to suggest that to fellow programmers. Comparing the results shows that the respondents in the case study questionnaire reported a slightly higher trust in AI tools (mean = 3.92) than those in the study by Kuhail et al. (2024) (mean = 3.65), on a five-point Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). The standard deviation for the answers was also quite similar between the studies: 0.84 in the case study and 0.75 in the study by Kuhail et al. (2024).

To conclude RQ1, AI tools show significant potential to transform the daily work of software developers. This is achieved by automating and helping with routine tasks. Despite the many benefits of these tools, their usage also poses some challenges. The responsibility for optimal AI tool usage and outcomes lies with the developer and their ability to understand and make informed use of the technology. After all, a developer is always responsible for their own code, regardless of whether it was written with AI tools or not.

## **5.2 AI tools' effects on developer productivity**

In terms of AI tools' effects on developers' productivity, the findings from the literature review and the case study align relatively well. The findings were mostly similar, but scientific literature also saw increased quality in work results as one effect on productivity, which as Oliveira et al. (2020) note, developers often associate to productivity.

A major difference between scientific literature and the case study findings emerge in the archival analysis. The analysis revealed that the median relative effort estimation error increased by approximately 56 % after the introduction of Copilot, indicating that estimation accuracy declined significantly. The Mann-Whitney U test confirmed this change as statistically significant with medium effect size. At the same time, mean project lead times decreased by approximately 40%, suggesting that projects were completed more quickly overall. This was also proven as statistically significant change with medium effect size with a Mann-Whitney U test. This points to a situation where task completion has accelerated but variability in estimation reliability has grown.

Overall, the evolution of project metrics presents a complicated picture. While developers appear to complete tasks faster, the estimations of required

effort have become less accurate. However, these findings should not be interpreted as direct evidence of decreased productivity. The archival Jira dataset, as previously discussed, contains several limitations that weaken its suitability for drawing strong conclusions. The project data is largely manually logged, creating room for human error, such as missing or inaccurate time entries. Furthermore, effort estimations are typically defined early in the project planning phase and might remain unchanged in Jira even when project scope or requirements evolve. As a result, deviations between estimated and actual hours reflect the project dynamics rather than real changes in productivity. Moreover, the presence of skewed data undermines the statistical validity of the findings, as outliers and unbalanced project categories can bias mean values and reduce the comparability between the two time periods. Given these factors, the archival analysis should be interpreted primarily as indicative rather than conclusive, suggesting trends in workflow evolution rather than definitive evidence of a decline in developer productivity. Thus, it could be argued that the effect that Copilot has had on developers' productivity is not comprehensively reflected on these project metrics.

To conclude RQ2, developers themselves in the case study, as well as scientific literature, report an increase in productivity after integrating AI tools to their daily workflows. The archival analysis on the other hand showed mixed results. On one hand the project lead times had decreased indicating an increase in productivity, but on the other hand the relative effort estimation errors had grown indicating a decrease in productivity. All in all, AI tools effects in developers' productivity cover different aspects of software developers' work including accelerating work tasks such as programming, increasing comfortability regarding work, increasing code quality and relieving stress, as well as aiding in increasing developers' knowledge base. These are fundamental aspects of a software developer's work, contributing to an increase in productivity as well as professional growth and long-term well-being. The potential productivity gains are often realized when AI tools are used cautiously, maximizing their benefits while ensuring that critical control remains with the developer. The observations demonstrated that learning optimal AI tool usage techniques is crucial for realizing their full potential. Testi

### **5.3 Best practices for AI tool usage**

After studying the use cases, challenges and productivity gains of using AI tools in software development, I have formed some best practice suggestions regarding AI tools usage. These suggested practices are guidelines from both scientific literature and empirical findings of the case study. Their goal is to

help software developers maximize the benefits of AI tools while avoiding common problems.

One of the most important practices is to use AI tools for well-defined and relatively simple tasks. These tasks should be ones that the AI tool is known to be capable of solving. In practice, this means using the tools to automate tasks such as generating boilerplate code, writing documentation, or suggesting commit messages. As demonstrated in the observations by participant one, when AI tools are used like this, they have the capability to meaningfully reduce the time spent on monotonous work without interfering with core logic or decision-making.

Second guideline is that all AI-generated output should be treated as a suggestion rather than a final solution. As stated, developers are always responsible of their own code. Thus, the code should be critically reviewed, tested, and fully understood before it is integrated into a project. This way developers maintain the control over the codebase. The case study observations confirmed that the greatest productivity gains occurred when AI was used to streamline small, routine tasks that required little to no decision making or logical reasoning. This way developers maintained ownership over more complex problem-solving. This is not to say that AI should not be used in problem solving but rather that it is to be used with caution.

The literature review and questionnaire answers also presented several behavioural patterns that tend to reduce the value of AI tools or even introduce risks. These include asking AI to solve too large or complex problems in a single prompt, over relying on AI for architectural or logical decisions, and copying outputs without understanding them. Blindly trusting AI generated code can lead to bugs, loss of control, and technical debt. Developers should also remain informed about data privacy and avoid sharing sensitive project information with external tools.

In summary, AI tools can be a powerful aid when used cautiously with clear boundaries. The key is to keep control of the development process and not experiment with the tools too much. Understanding the limitations of AI tools is essential to utilize the potential they have. AI tools need to be used in structured and well-planned ways to ensure consistent and reliable outcomes.

## 5.4 Future work

There remains substantial potential for future research on the use of AI tools in software development. Currently, a large portion of scientific literature on the topic focuses on LLMs' performance in programming related problems rather than on how they could be used. Furthermore, most of the literature focusing on AI tools usage are scoped only to programming, often overlooking other phases of the software development life cycle.

A promising direction for future research would be to study the role and benefits of AI tools in all stages of the software development life cycle as Özpolat et al. (2023) did. Such an investigation could provide insights into whether AI tools have similar potential in technical and cognitive dimensions of software development activities beyond programming. As the findings by Özpolat et al. (2023) imply, AI tools have potential to provide significant help in each step of the software development life cycle.

Another direction for potential future research would be to study developer productivity with AI tools through code-based metrics like amount and size of commits over time. These metrics would likely provide a clearer illustration of AI tools' effects on developers' code creation capabilities and productivity than the metrics used in the case study. Naturally, studying the evolution of the code quality would also provide additional value for the potential study.

In addition to their technical applications, AI tools may also influence the social and collaborative dynamics within software development teams. Coutinho et al. (2024) suggest that the use of generative AI tools can indirectly affect communication and collaboration by providing quick access to information and facilitating knowledge acquisition. They state that these capabilities may enhance team interactions by enabling members to share insights more efficiently and align their understanding toward shared objectives. However, while AI tools can support individual developer's productivity and team collaboration, there is also a risk that outsourcing all social interaction to AI tools may lead to silos forming. When developers increasingly turn to AI tools for assistance rather than engaging with colleagues, the opportunities for knowledge sharing and learning are decreasing. As AI tools become further integrated into daily workflows, it is essential to investigate the implications of their usage in terms of teamwork and communication.

## 6 Validity

While this study was designed to provide robust insights into the effects of AI tools usage on software development, some threats to validity remain despite best efforts to address them. To evaluate these, the classification framework proposed by Runeson and Höst (2009), which builds on Yin (2009), widely used in software engineering case studies is adopted. In this framework validity is analysed from four viewpoints: construct validity, internal validity, external validity, and reliability. Although originally developed for positivist research, Runeson and Höst (2009) argue it can be adapted for flexible, interpretive studies like this one, making it a suitable framework for this thesis. Below, each aspect is discussed from the viewpoint of this thesis.

### 6.1 Construct validity

Construct validity examines whether the study's measures and concepts align with the intended research focus. In this case this means, the use of LLM-based AI tools and their impact on developers' daily work and productivity. The first potential threat is that participants interpret these concepts differently from the study's definitions. For example, while the thesis specifies LLM-based tools, some developers might have included other AI functionalities relevant to their work, in their responses. This is a possibility especially as AI features and tools are becoming increasingly integrated into development environments. This could distract the focus from the intended tools.

Furthermore, productivity, as discussed in Chapter 2, is a complex and abstract concept, consisting of both technical and social factors. The study relies on self-reported perceptions from the questionnaire, objective metrics from Jira archival data, and observational insights, but these may still not fully capture the complex nature of productivity. As discussed, the productivity of a software developer cannot be completely measured to a score. To mitigate this, the questionnaire included the mindset towards working as part of productivity. Also, findings were validated through triangulation of multiple data sources. While these steps reduce the risk, differing interpretations of productivity among participants remain a threat to construct validity.

### 6.2 Internal validity

As explained by Runeson and Höst (2009), internal validity concerns the ability to establish causal relationships between variables while ruling out alternative explanations. As this study is exploratory and descriptive in aiming

to understand how AI tools influence software development rather than to prove strict causation, it is less susceptible to internal validity threats. However, a risk arises in the archival analysis of the Jira data. The analysis compares project metrics before and after the introduction of Copilot in the case company in January 2024. The risk here is that these project metrics are influenced by a wide variety of other factors and Copilot's direct influence is difficult to isolate. Furthermore, the Jira data itself does not represent the entire picture of developers' productivity as factors such as code quality cannot be determined from these numbers.

In addition to external factors, the archival Jira data itself presents limitations that affect the strength of the internal validity of this thesis. As previously discussed, the project data is largely manually logged, which introduces potential inaccuracies. Effort estimations, that sometimes are defined early in project planning, may also remain unchanged even when project circumstances evolve, which might lead to inconsistency between estimated and realized values that do not necessarily reflect actual changes in productivity. Furthermore, the dataset shows some skewness, which can distort statistical comparisons by amplifying or masking trends. These characteristics mean that the archival analysis should be viewed as indicative rather than conclusive, providing supportive evidence rather than a definitive measure of Copilot's impact on developer productivity.

To address this, the study also uses contextual data from the questionnaire and observations to interpret archival results cautiously. It is also clearly stated that the archival data is simply indicative and does not establish causation. While the triangulation of data sources also strengthens the findings, future research could employ more controlled methods to isolate these effects. For this study, the selected approach provides a reasonable foundation for discussing project metric's evolution and potential AI tool impacts.

### **6.3 External validity**

External validity addresses the generalizability of the study findings beyond the case scope. In this thesis the focus was on AI tool usage in a single unit of a Finnish IT company specializing in RPA development. The external validity risks of this study concern the applicability of the findings to other settings in the field, such as teams developing different types of software, using alternative AI tools, or operating in different organizational environments. Another factor in external validity is the number of participants included in the case study. The questionnaire was answered by 16 of 40 developers in the

case company. Furthermore, the observations had only two volunteers participating. While the case study provided valuable insights into the usage of AI tools, this raises the question of whether this group is an unbiased representation of the developers in the case company. Thus, the results should be interpreted with caution and understanding that the research findings represent a subset of the case company's developers rather than the full company. The external validity of the findings could be strengthened with further research with larger participant groups.

To the external validity risks, this study also presents a comprehensive literature review identifying similar patterns of AI tool usage across various contexts, increasing the transferability of the findings. Moreover, there is a lot of variation between tasks within the case company ranging from designing new RPA solutions to maintaining existing ones. This offers insight relevant also in other software development settings. The findings are not universally generalizable however, they provide practical implications for organizations adopting similar tools in comparable domains. Nevertheless, caution is advised when applying these results elsewhere. Future studies with multiple cases could enhance generalizability.

## **6.4 Reliability**

Reliability evaluates the reproducibility and consistency of the study's findings in similar conditions (Runeson and Höst, 2009). Reaching high reliability is demanding in a study like this where participants are unique and organizational context is very specific. A key threat arises from the inherent subjectivity of the data collection and analysis. The data analysis in particular relies at times on the researchers' subjective interpretation. For example, different observers might interpret developers' interaction with AI tools differently. As mentioned, the questionnaire answers also contain some inherent subjectivity.

To enhance reliability, this study provides detailed descriptions of the employed data collection and analysis methods. This increases transparency and allows other researchers to replicate the study in a similar context. The use of multiple data sources further strengthens confidence in the findings through triangulation. However, the thematic coding of the questionnaire responses and observation data, carried out by a single researcher, introduces some risk of bias. Involving multiple coders could improve reliability, but the nature of a master's thesis rules out such possibilities. Despite this, the methodological rigor and triangulation efforts support the study's reliability.

## 7 Conclusion

This thesis researched the usage of AI tools in software development as well as its effects on software developers' productivity. A review of the existing scientific literature was conducted to assess the current state of research in this field. Furthermore, the embedded single case study was carried out to understand the usage of GitHub Copilot through empirical evidence from a natural software development environment.

The findings of the literature review as well as the case study questionnaire and observations showed that there are already several established use cases for AI tools in software development enabling increase in developer productivity. These productivity gains included faster task completion, reduced cognitive load in routine tasks and increase in knowledge base as well as increased enjoyment in work. The productivity gains were achieved by utilizing AI tools' capabilities in automated code generation, code reviews, information retrieval and automated documentation writing.

However, this master's thesis also highlights several limitations and challenges in AI tool usage based on the findings from the literature review and case study questionnaire. These include occasional overreliance on AI to generate code, difficulties in debugging unfamiliar output, and difficulties in learning effective prompting strategies. Furthermore, technical constraints, such as compatibility issues with IDEs and limited performance in rare or company specific technologies were reported.

In conclusion, AI tools have the potential to increase developers' productivity and satisfaction when used with appropriate caution. To fully realize the potential offered by these tools, organizations need to invest in both the tools as well as training how to optimally use them in the organizational context. As AI tools evolve rapidly, further research is needed to explore their broader impacts and potential across the entire software development lifecycle and their long-term effects on learning and team dynamics.

## References

- Abbass, H. (2021). "Editorial: What is Artificial Intelligence?" *IEEE Transactions on Artificial Intelligence* 2(2): 94-95.
- Akbar, M. A., A. A. Khan and P. Liang (2025). "Ethical Aspects of ChatGPT in Software Engineering Research." *IEEE Transactions on Artificial Intelligence*.6(2):254-267.
- Boehm, B. W. (1984). "Software Engineering Economics." *IEEE Transactions on Software Engineering* SE-10(1): 4-21.
- Boynnton, P. M. and T. Greenhalgh (2004). "Selecting, designing, and developing your questionnaire." *Bmj* 328(7451): 1312-1315.
- Brooks, J. F. "Essence and Accidents of Software Engineering." *IEEE Computer*. April, 1987. vol. 20: pp. 10-19 : ill. includes bibliography. -- See also *Information Processing '86* edited by H.J. Kugler; and Brooks, F.P. 'The Mythical Man-Month,' Addison-Wesley, 1978.
- Chapetta, W. A. and G. H. Travassos (2020). "Towards an evidence-based theoretical framework on factors influencing the software development productivity." *Empirical Software Engineering* 25(5): 3501-3543.
- Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph and G. Brockman (2021). "Evaluating large language models trained on code." *arXiv preprint arXiv:2107.03374*.
- Coutinho, M., L. Marques, A. Santos, M. Dahia, C. França and R. de Souza Santos (2024). "The Role of Generative AI in Software Development Productivity: A Pilot Case Study." *Proceedings of the 1st ACM International Conference on AI-Powered Software*: 131-138.
- Cui, K. Z., M. Demirer, S. Jaffe, L. Musolff, S. Peng and T. Salz (2024). "The Productivity Effects of Generative AI: Evidence from a Field Experiment with GitHub Copilot." *An MIT Exploration of Generative AI*
- Faraj, S. and L. Sproull (2000). "Coordinating Expertise in Software Development Teams." *Management Science* 46(12): 1554-1568.
- Finnie-Ansley, J., P. Denny, B. A. Becker, A. Luxton-Reilly and J. Prather (2022). *The robots are coming: Exploring the implications of OpenAI codex on introductory programming*. ACM International Conference Proceeding Series.
- Fjeld, E. K., & Brynn, R. (2024). *AI in Software Development and Its Potential Influence on Accessibility Compliance*. *Studies in health technology and informatics*, 320, 493–500.
- GitHub (2025). "What is GitHub Copilot?". Github. Available at: <https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot> (Accessed: 17 June 2025).

- Graziotin, D., X. Wang and P. Abrahamsson (2013). Are happy developers more productive? The correlation of affective states of software developers and their self-assessed productivity. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 14th International Conference on Product-Focused Software Process Improvement, PROFES 2013
- Jiang, E., E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai and M. Terry (2022). Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models. Conference on Human Factors in Computing Systems - Proceedings.
- Jørgensen, M. (2004). "A review of studies on expert estimation of software development effort." *Journal of Systems and Software* 70(1-2): 37-60.
- Jørgensen, M., and Molokken-Ostfold, K. (2005). Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software engineering*, 30(12), 993-1007.
- Kemell, K., M. Saarikallio, A. Nguyen-Duc, P. Abrahamsson. (2024) Adoption of Large Language Models in Software Organizations - a Multiple Case Study. Available at SSRN: <https://ssrn.com/abstract=4964930>
- Kuhail, M. A., S. S. Mathew, A. Khalil, J. Berengueres and S. J. H. Shah (2024). "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools." *Science of Computer Programming* 235.
- Liang, J. T., C. Yang and B. A. Myers (2024). A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. Proceedings - International Conference on Software Engineering.
- Maurer, F. and T. D. Hellmann (2013). People-centered software development: An overview of agile methodologies. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). International Summer Schools on Software Engineering. Vol 7171: 185-215.
- Mbizo, T., G. Oosterwyk, P. Tsibolane and P. Kautondokwa (2024). Cautious optimism: The influence of generative ai tools in software development projects. Annual Conference of South African Institute of Computer Scientists and Information Technologists, Springer.
- McConnell, S. (1996). Rapid development: taming wild software schedules. [online] O'Reilly Online Learning
- Mendes, W., S. Souza and C. D. Souza (2024). "You're on a bicycle with a little motor": Benefits and Challenges of Using AI Code Assistants. Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering. Lisbon, Portugal, Association for Computing Machinery: 144-152.
- Meyer, A. N., T. Zimmermann and T. Fritz (2017). Characterizing Software Developers by Perceptions of Productivity. International Symposium on Empirical Software Engineering and Measurement.

- Mohammad, A. F., B. Clark and R. Hegde (2023). Large Language Model (LLM) & GPT, A Monolithic Study in Generative AI. 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE).
- Moradi Dakhel, A., V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais and Z. M. Jiang (2023). "GitHub Copilot AI pair programmer: Asset or Liability?" *Journal of Systems and Software* 203: 111734.
- Moroz, E. A., V. O. Grizkevich and I. M. Novozhilov (2022). The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. Proceedings of the 2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2022.
- Oliveira, E., E. Fernandes, I. Steinmacher, M. Cristo, T. Conte and A. Garcia (2020). "Code and commit metrics of developer productivity: a study on team leaders perceptions." *Empirical Software Engineering* 25(4): 2519-2549.
- Pangavhane, S., G. Raktate, P. Pariane, K. Shelar, R. Wakchaure and J. N. Kale (2024). AI-Augmented Software Development: Boosting Efficiency and Quality. 2024 International Conference on Decision Aid Sciences and Applications, DASA 2024.
- Pashchenko, D. S. (2023). "Early Formalization of AI-tools Usage in Software Engineering in Europe: Study of 2023." *International Journal of Information Technology and Computer Science* 15(6): 29-36.
- Perry, D. E., S. E. Sim and S. Easterbrook (2006). Case studies for software engineers. Proceedings of the 28th international conference on Software engineering. Shanghai, China, Association for Computing Machinery: 1045–1046.
- Prather, J., B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley and E. A. Santos (2023). "“It’s Weird That it Knows What I Want”: Usability and Interactions with Copilot for Novice Programmers." *ACM Transactions on Computer-Human Interaction* 31(1).
- Pressman, R. S. (2005). *Software engineering: a practitioner's approach*, Palgrave macmillan.
- Pudari, R. and N. A. Ernst (2023). "From copilot to pilot: Towards AI supported software development." *arXiv preprint arXiv:2303.04142*.
- Runeson, P. and M. Höst (2009). "Guidelines for conducting and reporting case study research in software engineering." *Empirical Software Engineering* 14(2): 131-164.
- Sarrion, E. (2023). *Exploring the Power of ChatGPT : Applications, Techniques, and Implications*. New York, NY, Apress.
- Seaman, C. B. (1999). "Qualitative methods in empirical studies of software engineering." *IEEE Transactions on software engineering* 25(4): 557-572.

- Smit, D., H. Smuts, P. Louw, J. Pielmeier and C. Eidelloth (2024). "The impact of GitHub Copilot on developer productivity from a software engineering body of knowledge perspective." AMCIS 2024 Proceedings
- Solohubov, I., A. Moroz, M. Y. Tiahunova, H. H. Kyrychek and S. Skrupsky (2023). Accelerating software development with AI: exploring the impact of ChatGPT and GitHub Copilot. CTE. CTE (2023)
- Sommerville, I. (2016). Software engineering. Boston, Pearson.
- Stavridis, A. and A. Drugge (2023). The Rise of Intelligent System Development: A Qualitative Study of Developers' Views on AI in Software Development Processes. Umeå University, Faculty of Social Sciences, Department of Informatics.
- Vaithilingam, P., T. Zhang and E. L. Glassman (2022). Expectation vs Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems. New Orleans, LA, USA, Association for Computing Machinery: Article 332.
- Weber, T., M. Brandmaier, A. Schmidt and S. Mayer (2024). "Significant productivity gains through programming with large language models." Proceedings of the ACM on Human-Computer Interaction 8(EICS): 1-29.
- Wohlin, C. (2021). "Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study?" Information and Software Technology 133: 106514.
- Wong, M.-F., S. Guo, C.-N. Hang, S.-W. Ho and C.-W. Tan (2023). "Natural language generation and understanding of big code for AI-assisted programming: A review." Entropy 25(6): 888.
- Yilmaz, M., R. V. O'Connor and P. Clarke (2016). "Effective Social Productivity Measurements during Software Development - An Empirical Study." International Journal of Software Engineering and Knowledge Engineering 26(3): 457-490.
- Yin, R. K. (2009). "Case Study Research: Design and methods" 4th ed. Thousand Oaks, CA: Sage. Vol 5.
- Zhang, B., P. Liang, X. Zhou, B. Zhang, P. Liang, X. Zhou, A. Ahmad and M. Waseem (2023). "Demystifying Practices, Challenges and Expected Features of Using GitHub Copilot." International Journal of Software Engineering and Knowledge Engineering 33(11n12): 1653-1672.
- Zhou, X., P. Liang, B. Zhang, Z. Li, A. Ahmad, M. Shahin and M. Waseem (2025). "Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow." Journal of Systems and Software 219.
- Ziegler, A., E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam and E. Aftandilian (2024). "Measuring GitHub Copilot's Impact on Productivity." Commun. ACM 67(3): 54-63.

Özpolat, Z., Ö. Yıldırım and M. Karabatak (2023). "Artificial Intelligence-Based Tools in Software Development Processes: Application of ChatGPT." *European Journal of Technique (EJT)* 13(2): 229-240.

## A. Questionnaire

### AI tools and productivity

Hello!

This questionnaire will focus on AI tool usage in software development and how these tools have affected your daily work.

It is fully up to you to decide what experiences you want to share. The key is to share helpful info on how AI tools impact productivity and transform your work, directly and indirectly. In this context we are especially interested in GitHub Copilot in software development related activities for example coding and code planning.

Note: The term Copilot refers to GitHub's Copilot in this questionnaire. If you wish to share experiences with other AI tools please mention it in the answer so that the used tool can be linked with the shared experience.

In the following pages, you will be asked questions about experiences with Copilot. There are no right or wrong answers and everyone has their own opinion. Now, please gather your thoughts about Copilot usage and take as much time as you need to fill out this questionnaire.

When you are ready, click next

Next

## AI tools and productivity

Let's start with demographic questions

### 1. Name

### 2. Job description

### 3. Age

- 18-30
- 31-40
- 41-50
- 51-65
- >65

Previous

Next

## AI tools and productivity

Next up, Programming background

### 4. Working experience in software development

- <2 years
- 2 to 5 years
- 6 to 10 years
- 11 to 16 years
- >16 years

### 5. Programming experience (including personal projects)

- <2 years
- 2 to 5 years
- 6 to 10 years
- 11 to 16 years
- >16 years

### 6. Additional programming skills beyond the job description?

Previous

Next

## AI tools and productivity

Then some questions about AI tools and their relations to productivity

### 7. How often do you use Copilot in your work

- Several times a day
- Daily
- Weekly
- Monthly
- Rarely or not at all

### 8. How do you feel Copilot has affected your working efficiency or productivity?

- Very Positive
- Positive
- Neutral
- Negative
- Very Negative

### 9. List the concrete ways of how and why you use Copilot in your work.

**10. Explain how Copilot usage has affected your productivity or mindset in working? Why?**

**11. I feel like I have benefited from using Copilot in my work**

- Strongly agree
- Agree
- No comments
- Disagree
- Strongly disagree

**12. Explain the possible limitations and challenges you face while using Copilot in your work**

**13. Any reasons you would avoid using Copilot at work?**

Previous

Next

## AI tools and productivity

Trust in AI tools

### 14. I feel Copilot provides trustworthy help

- Strongly agree
- Agree
- No comments
- Disagree
- Strongly disagree

### 15. I feel Copilot is effective in the help it provides

- Strongly agree
- Agree
- No comments
- Disagree
- Strongly disagree

### 16. I would recommend Copilot to fellow programmers

- Strongly agree
- Agree
- No comments
- Disagree
- Strongly disagree

Previous

Next

## AI tools and productivity

Now we have arrived to the end of this questionnaire. Here you have one last opportunity to discuss any other experiences with Copilot you find interesting

**17. Any other important or interesting experiences with Copilot? Anything else you would like to mention?**

Previous

Submit