Enhancing software development processes with artificial intelligence

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Software Engineering
October 2024
Vili Ståhlberg

UNIVERSITY OF TURKU
Department of Computing

VILI STÅHLBERG: Enhancing software development processes with artificial intelligence

Master of Science (Tech) Thesis, 87 p., 3 app. p.
Software Engineering
October 2024

Artificial intelligence (AI) has emerged as one of the most revolutionary technologies since the internet. While many fields are affected, the field of software development is experiencing a rather transformative impact. With the rise of AI, organizations are presented with a new challenge: adopting artificial intelligence as a part of their software development workflow.
However, the discussion about adopting AI in software development often seems to gravitate towards the use of generative AI as part of the programming workflow, with the rest of the software development lifecycle left unnoticed.
In this thesis we set out to examine the enhancement of software development phases and tasks with AI-based solutions, while also identifying obstacles preventing the effective adoption of AI. We explore this topic with the viewpoint in mind that there likely exists undiscovered potential for enhancement where AI utilization is not as apparent. To achieve this goal, we perform a systematic literature review and conduct interviews to gain insight from experts working in the field of software development. With their opinions and experiences in mind, we draw a holistic overview of AI utilization, and how it could be improved, in the entire software development lifecycle.

Keywords: artificial intelligence, machine learning, software development

Contents

1 Introduction

During the past few years, artificial intelligence (AI) has been the topic of discussion in mainstream media. It has been described as one of the biggest revolutions since the internet. It has affected almost all domains in our lives. AI has even been characterized in popular media as the thing that is capable of taking over many jobs that are currently done by humans [1]. The field of software development is not exempt from this discussion.

Companies across the world that develop software are now adopting artificial intelligence as part of their development workflow. Meanwhile, the demand for high-quality software has also increased, while time frames have tightened to meet the customers' and shareholders' ever-increasing expectations. For a company to stay afloat, they must adapt to this wave of change and the increased expectations. However, this is only the bare minimum. If a company wishes to stay competitive, and even gain a lead against its competitors, they need to seek out ways for effective and unique AI utilization.

To add to this already nuanced and complex matter, the adoption of AI can be a slippery slope for companies. This is the case for adopting AI in any domain, whether it be for enhancing software development or something else. By being exposed to mainstream media, one can get quite an optimistic picture of what tasks AI is capable of performing now and in the near future. Top executives and decision makers in companies are prone to this exposure as well. This could lead to companies investing into AI without truly understanding the true capabilities of the current AI-based solutions that are available.

The goal of this thesis is to shed light on just this very topic: how can we realistically and effectively employ AI across the entire software development lifecycle (SDLC). Not only that, we also want to gauge how Company X is currently utilizing AI as well as how they can overcome obstacles in the effective adoption of AI in software development. With these goals in mind, we have decided on the following research questions:

RQ1 *Which software development phases and tasks can be enhanced using artificial intelligence?*

RQ2 *Which software development phases and tasks benefit the most from the adoption of artificial intelligence based solutions?*

RQ3 *How can Company X improve their current utilization of artificial intelligence to enhance software development?*

RQ4 *How can Company X overcome obstacles in adopting and utilizing artificial intelligence for software development?*

With these research questions in mind, the goal and scope of this thesis is not to produce an exhaustive list of all potential AI enhancement scenarios for software development. Rather, we want to assess the level of enhancement potential through examples and scenarios. For RQ1 and RQ2, the scope of the research is the software development field as a whole. However, for RQ3 and RQ4, we want to specifically examine the questions in terms of Company X and their domain.

The rest of this thesis has been structured so that we are able to explore the topic at hand in an intuitive manner. First, we outline the research design in Chapter 2 to define the research environment as well as provide reasoning and motivation for choosing the research design. Then, we begin the research by reviewing academic literature about software development in Chapter 3 and about artificial intelligence in Chapter 4. After that, we prepare for the research by innovating and exploring different ways of utilizing artificial intelligence in software development in Chapter 5. In Chapter 6 we present the results of the research. Finally, we discuss the results we have gathered as well as answer the research questions in Chapter 7, and conclude the thesis in Chapter 8.

2 Research Design

In order to make meaningful progress in any field of science, systematic research is needed. The acquisition of new knowledge is commonly based on elements like observations, questions, experiments, hypotheses, and analyses, to name a few. In this chapter we will outline the design of the research, what processes are involved, and what methodologies are used and why they were chosen. [2]

2.1 Environment

Before attempting to solve a problem or presenting a solution, the environment should be defined. The research process of this thesis is ultimately connected to Company X and its business problems in developing a one-stop e-commerce service for its customers. Company X is a retailing conglomerate that operates in various sectors including grocery trade, construction, retail technical trade as well as new and used car trade. With that said, Company X has a high threshold for producing reliable and robust software and systems in multiple different domains including consumer-facing applications as well as internal systems. The goal of this research is to provide a holistic view of the various options that are available for enhancing software development processes with AI-based tools in the aforementioned context.

2.2 Data collection

To construct a knowledge base for the thesis, we use two distinct methods to collect data: (1) a literature review of the thesis topics and (2) semi-structured interviews with professionals working in software development. These methods are outlined in this section as well as the motivation for choosing them.

2.2.1 Literature review

As is common in the academic landscape, the basis for this thesis is established via a literature review. It provides a clear context, which is important for a topic like artificial intelligence. Not only does it establish a solid foundation, it allows this study to identify gaps in this topic that may be of interest for further research.

We begin the literature review by exploring academic material about software development. This is done in an effort to outline what exactly is being enhanced in a clear manner. After that, we continue the literature review by defining the second component of the thesis: artificial intelligence, or machine learning to be more specific. Finally, we both (1) explore existing material and (2) innovate based on the material we have reviewed to identify potential ways of enhancing software development with artificial intelligence.

2.2.2 Interviews

To gather real data from actual software developers, interviews were chosen to accompany the literature review as a means of collecting data. Interviewing experts of the field also provides insight, personal experiences, credibility, and a unique perspective to the matter at hand, something that is acquired only by working for many years in the field. Additionally, because we want to gather information from Company X's perspective as well, interviewing Company X's internal software developers appeared to be a good fit.

For this thesis, a semi-structured interview technique was chosen, making the research qualitative. The interviews were structured using questions that provided a theme and a clear structure for the entire interview while still allowing a more open discussion. The interviews are described in more detail in Chapter 6.

3 Software Development

In order to truly understand what is being enhanced, and to provide an answer to any of the research questions, we must define what software development is. Furthermore, we need to understand what explicit phases and tasks software development involves. The aim of this chapter is to provide just that: an understanding of what is involved in the entire software development lifecycle (SDLC). In Section 3.1 we will explore some prominent software development models that have shaped the way we develop software over the years in an attempt to identify distinct developmental phases. After that, in Section 3.2, we will discuss the aforementioned phases to obtain a holistic view of what is being enhanced. Finally, we will attempt to identify a set of distinct tasks that are performed during said phases.

Software development is the process of creating and maintaining systems, applications and other software components by designing, implementing, testing, deploying and maintaining source code. However, software development can be characterized as the set of all activities required to produce software products ranging from conceptualization all the way to the final manifestation of the product, implying that the process is much more convoluted than simply writing code.

This complexity is further compounded by the dynamic nature of technology and the evolving needs of users. The volatility caused by disruptive technologies like artificial intelligence has implications on both (1) what kind of software we develop and (2) what tools, frameworks and models we use to develop software. To navigate through this intricate process, there are distinct models and methodologies that provide developers with a set of guidelines and instructions on how to orchestrate software development. Additionally, these models are commonly organized into explicit phases, each with its own set of tasks and objectives.

3.1 Models

Software development models are used to create a structured and systematic approach to software development. They are almost a necessity when organizing the software development domain into distinct phases that can be executed in different orders and intervals during the entire process depending on the project's needs.

In this section we will explore the diverse landscape of software development models, ranging from mature traditional methodologies like the waterfall model to more modern and iterative approaches such as the agile models.

3.1.1 Waterfall model

Waterfall model originates from the 1970s, and is still widely used in today's software development. Waterfall model is one of the more traditional and linear software development models in existence. As presented in Figure 3.1, it follows a relatively straightforward sequential step-by-step approach, including requirements analysis, software design, implementation, testing, integration, deployment (or installation) and maintenance. After each step, the results of the work are reviewed and verified. Each phase must be completed before moving on to the next one. [3]

According to Vijayasarathy et al., the waterfall model is the most used software development model in the public sector, although it is more common for enterprises in the public sector to utilize a mix of different models [4]. The public sector's favor towards waterfall-based development could be due to its structured approach,

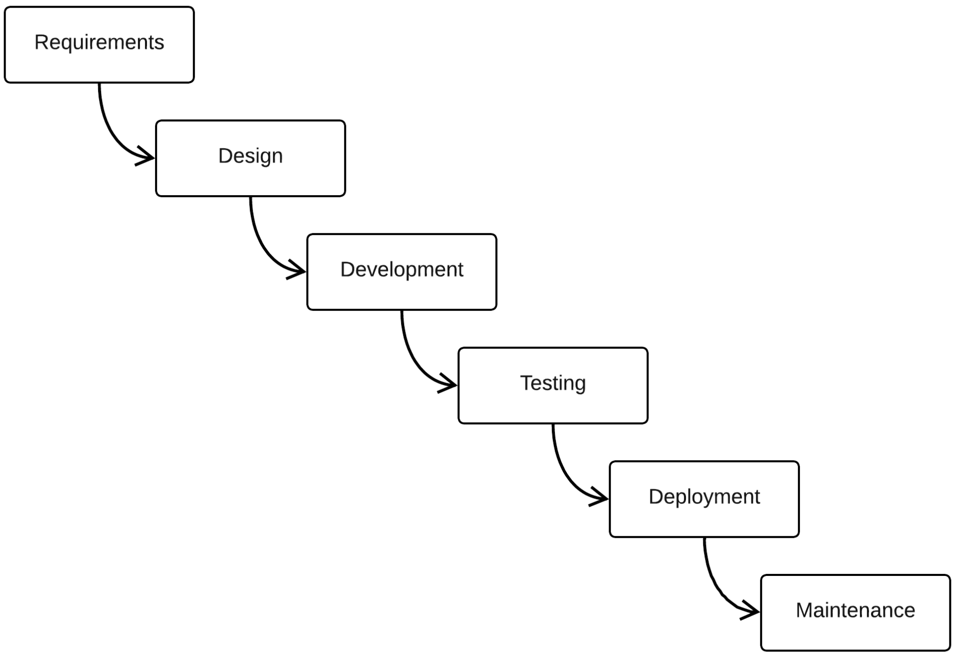

Figure 3.1: Developmental phases of the waterfall model.

facilitating compliance with regulatory standards, and accommodating contractual agreements. Furthermore, the waterfall model provides apparent predictability and planning for budgeting, and emphasizing comprehensive documentation, all of which are crucial considerations in large enterprise software development projects. In reality, it is not uncommon for governmental bodies and other entities in the public sector to go over their budgets, resulting in a disastrous software development project [5].

The waterfall model has been criticized for a number of reasons. Because of its linear nature, there is an innate inability to adapt and respond to a changing environment. In waterfall-based approaches, the development is planned extensively beforehand, meaning that things can be difficult to change during development. Having to fundamentally change things in the middle of development results in wasted work as requirements are thoroughly planned and validated in advance. This means that the requirements need to be discarded and reworked if the environment changes in a meaningful way. Other issues commonly found in waterfall development are high effort and costs for writing and validating documents in each developmental phase, lack of customer and end-user feedback, carrying problems from already finished phases to the following phases, and having to manage large amounts of documents throughout the project. [3]

3.1.2 V-model

Similar to the waterfall model, the V-model life cycle is linear in execution of processes, meaning each phase must be finished before the next phase can begin. However, in this model, testing plays a key role throughout the model. Testing procedures required by the project are developed very early in the life cycle during each of the phases before any implementation of actual code takes place. [6]

We can identify some developmental phases in the V-model. In the V-model, there are phases where requirements are outlined, and an architectural design is produced. There is a clear implementation phase as well. Finally, there is a clear operation and maintenance phase, where the system is maintained, and possibly improved over time. Testing, however, is executed throughout the project life cycle. Different kinds of tests are produced and executed depending on which phase of the model the project is currently in.

The idea behind V-model is to propose, design, and conceptualize the features and parts of the system on the left leg of the "V" shape, while the right leg represents appropriate testing and validation of the system. Each phase of the model has a corresponding process on opposite sides of the "V" shape, as presented in Figure 3.2. For example, the architectural design can be tested with integration testing. [6]

Again, similar to the waterfall model, the V-model is quite rigid and not very responsive to changes in the project and development environment [6]. Additionally, there is little to no feedback from stakeholders or the end-user as no early prototype

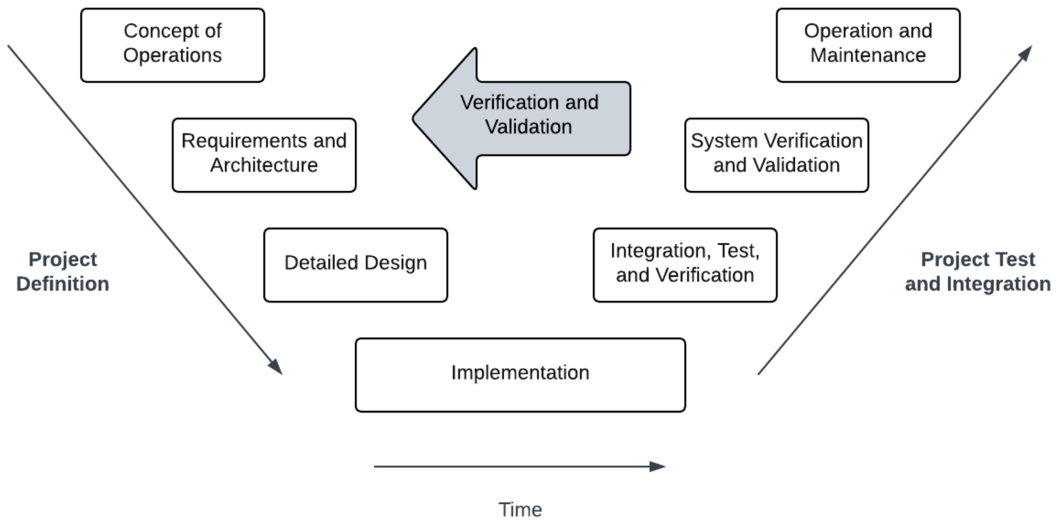
Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

Figure 3.2: Developmental phases of the V-model.

is created in this model of software development. However, compared to a traditional waterfall approach, at least some feedback is gained by testing throughout the development life cycle. With that said, the V-model may be more appropriate in projects that provide back-end functionality, where feedback from the end-users is not as important as, for example, a front-end application [7].

3.1.3 Spiral model

First described by Barry Boehm in 1986, the spiral model is a risk-oriented modified version of the waterfall model that introduces several iterations to the model. As described by Boehm in a later publication, the spiral model can be characterized as a "process model generator" that allows the risks specific to the project to direct and generate an appropriate process model for the project [8].

Some distinct phases can be identified within the spiral model as presented in Figure 3.3. Engineering of requirements takes place at the beginning of each iteration. On each iteration, designing, implementation, and testing also takes place.

Finally, when determined appropriate, the software is deployed to its users. In addition, identification and analysis of risks is performed on each iteration, as well as planning of the next possible iteration.

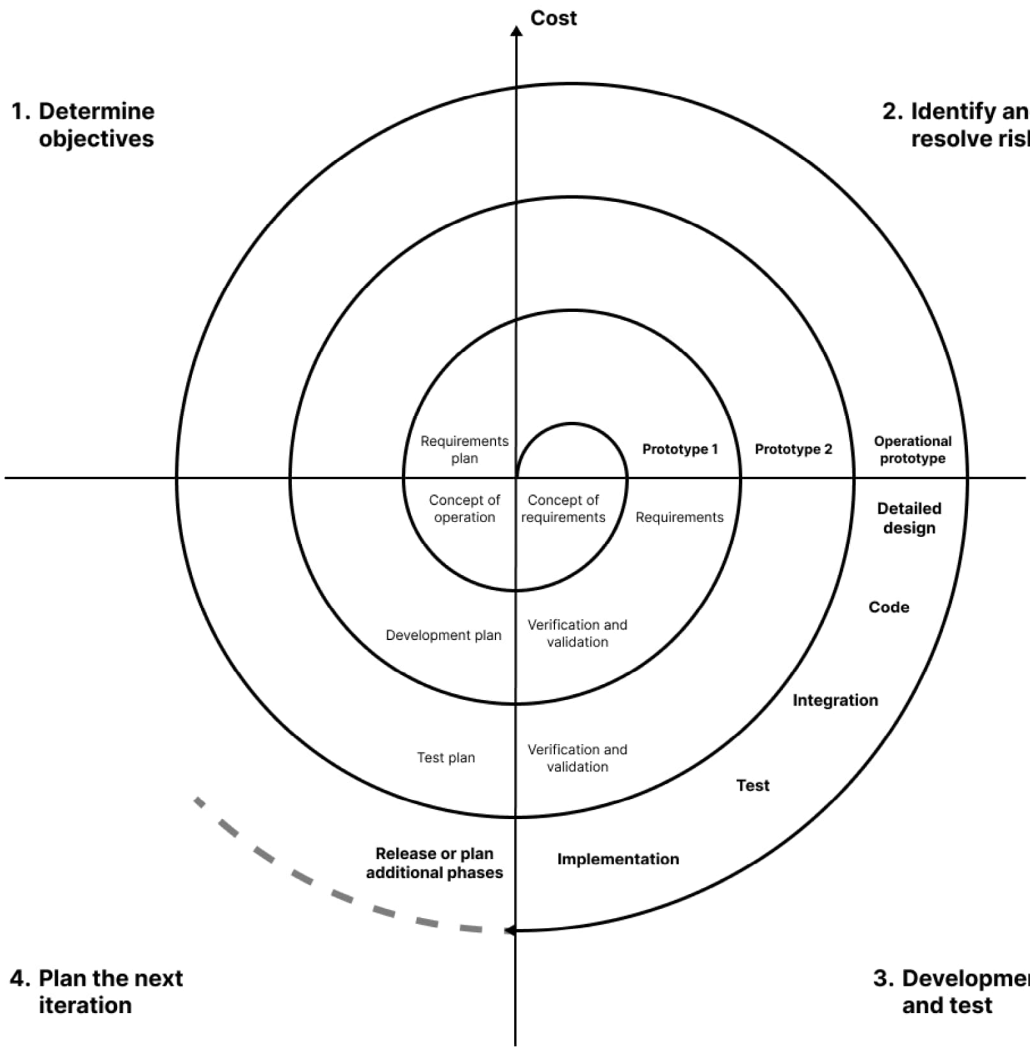

Figure 3.3: Simplified illustration of Barry Boehm's spiral model life cycle.

Unlike the waterfall and V-model, the spiral model introduces prototypes to the development process as each iteration within the spiral advances. Before a prototype is produced, risks are identified and assessed in an effort to manage them robustly. This is one of the key benefits of the spiral model as it allows identification of risks at inception, rather than acknowledging them during development. With its prototypes and iterative nature, the spiral model resembles agile software development models to an extent. [7]

3.1.4 Rapid application development

Rapid application development (RAD) originates from 1992 when James Martin published a paper with the same title. According to Beynon-Davies et al., RAD aims to (1) produce high quality systems, (2) allow quick development and delivery, and (3) lower the costs of development. In its core, RAD utilizes prototyping to produce a source for early feedback to the model. [9]

In a project where RAD is the model of choice, teams of typically four to eight people are formed. This has the implication that all team members must be highly skilled socially as well as in their craft. Teamwork is extremely important, which means that team-building activities like, for example, team dinners are a key part of the model. Additionally, in most RAD projects, joint application development (JAD) workshops take place where developers, potential end-users, and other stakeholders get together to produce requirements. [9]

One RAD implementation, known as dynamic systems development method (DSDM), utilizes five distinct phases. As presented in Figure 3.4, the phases are: feasibility study, business study, functional model iteration, design and build iteration, and implementation. In the feasibility study phase, the project feasibility is assessed and the development model is decided. The business study phase outlines high-level functionality as well as affected business areas. Next, the results of these two phases are used as a baseline for defining high-level requirements for the project. In the next two iterative phases, requirements are defined and a functional prototype is produced and reviewed. Finally, the implementation phase is used to deploy or hand the produced software over to its users or owners. In this model, the term implementation does not refer to actual coding, but rather the implementation of the result of the previous phases where the actual coding takes place. Furthermore, while there is no explicit testing phase in DSDM implementation of RAD, testing is encouraged in the DSDM documentation. [10]

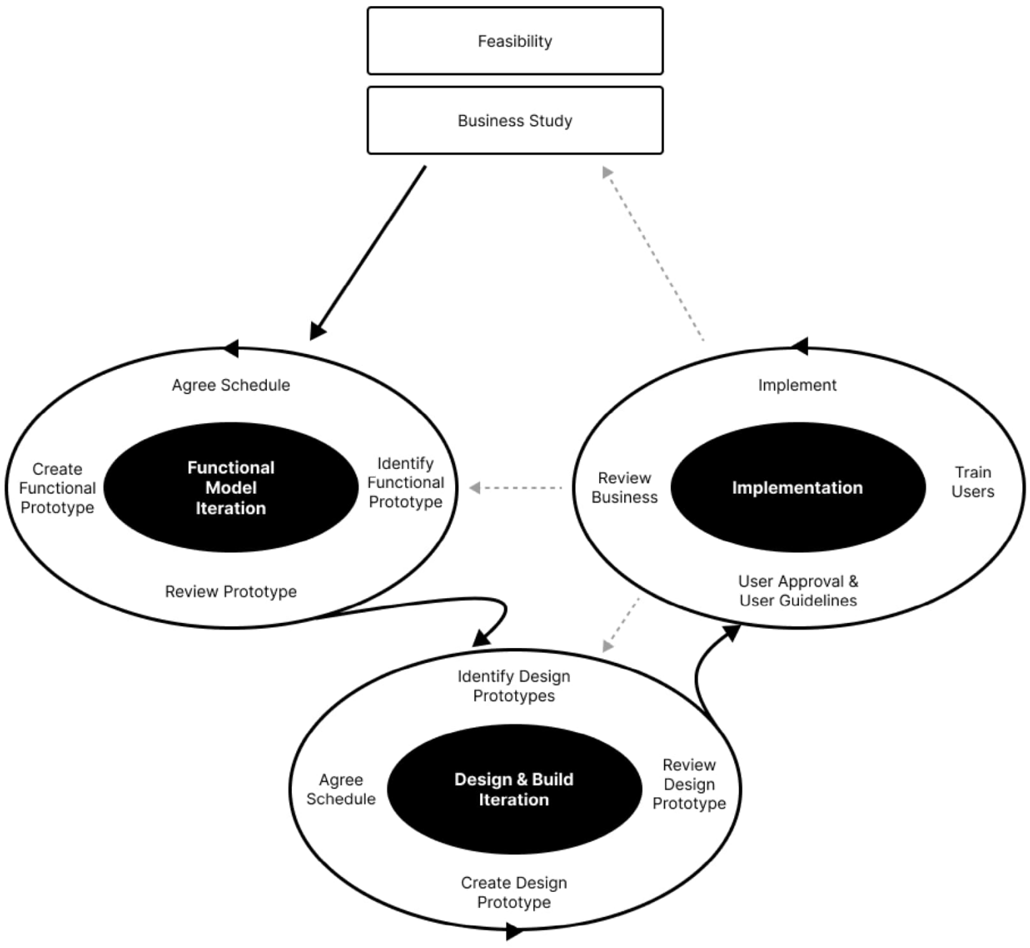

Figure 3.4: The dynamic systems development method, an implementation of RAD.

3.1.5 Agile development methodologies

Agile software development methodologies have emerged as a disruptive response to the growing demand for efficiency in the corporate landscape. In contrast to a traditional waterfall-based model, agile methodologies allow the development process to become more collaborative and adaptive to a changing environment. In addition, the end-user plays a key role throughout the life cycle of an agile model like Scrum. While agile methodologies began from software development, they have since spun out to different industries and are now the norm in many companies outside of software development [11].

The four primary high-level drivers of the Agile Manifesto are: [12]

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Each of these high-level drivers are valued from left to right, meaning the portion on the left take precedence over the items on the right, even though both sides are taken into consideration.

Agile software development can be characterized as utilization of a light-but-sufficient set of rules that guide both the project behaviour and human communication and interaction. In this context, lightness means steerable and adaptive, meaning the project can take a new approach or direction without too much trouble, something that waterfall-based models cannot achieve as easily [13]. This kind of an approach to software development is especially useful in environments that require high adaptability to changing requirements such as the banking industry which is constantly under pressure from regulation [14].

There are many agile methodologies in use like Extreme Programming (XP), Lean startup, and Kanban [13], [15], [16]. However, currently the most used agile methodology is Scrum [13]. The Scrum workflow boils down to three key fundamental elements: events, artifacts, and roles [17]. Next, we will dive deeper into Scrum and the aforementioned elements.

Scrum events, sometimes referred to as ceremonies or meetings, are a set of time-constrained activities that create the iterative and incremental structure of Scrum. The Sprint is a container for all other Scrum events: everything happens during iterative sprints. Depending on the needs of the project, the Sprint can be of different lengths of time, but it should be no more than a month, and it must remain consistent. Sprint Planning initiates the Sprint. Its purpose is to outline the work for the Sprint clearly. Additionally, the primary Sprint Goal is decided. Each day of the Sprint the Daily Scrum is held. As the name implies, this event is held daily, and its purpose is to assess the progress of the sprint as well as bring up any difficulties that developers are facing within a short 15-minute meeting. The final two events are Sprint Review and Sprint Retrospective, which are held at the end of the sprint. The Sprint Review's purpose is to showcase the work completed during the Sprint to stakeholders. The Sprint Retrospective, however, is used for the team to reflect on the Sprint's processes and assess what went well and what did not. The primary idea of the Sprint Retrospective is to improve the process by recognizing both successes and failures in the process. [17], [18]

Artifacts in Scrum can be characterized as scoreboards of the project: they represent work or value. Artifacts are transparently available to everyone in the team, keeping the key information same for everyone. The three Scrum artifacts are the Product Backlog, the Sprint Backlog, and an Increment. The Product Backlog is an ordered list of items required to enhance the product. Items are selected and picked into the Sprint Backlog from the Product Backlog when they are deemed

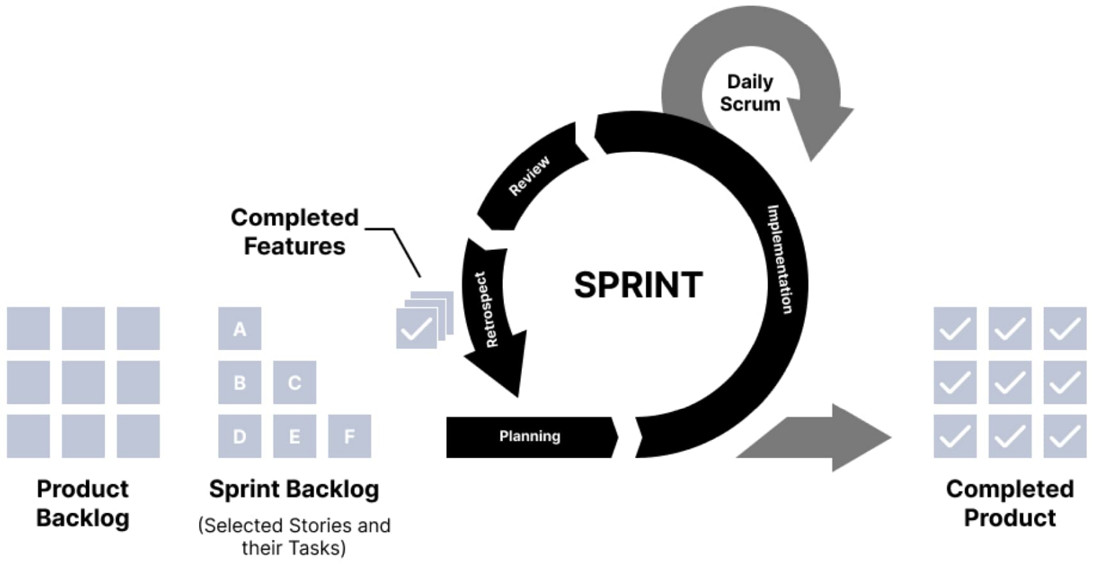

Figure 3.5: The Scrum workflow.

small enough for completion during a single sprint. The Sprint Backlog is a list of items that are going to be implemented during the sprint. Finally, An Increment is an incremental step towards the Product Goal: a description of the future state of the product. [17]

The third and final of the three elements is roles that make up the Scrum Team, a fundamental unit of 10 or fewer people. There are three roles in a Scrum team: (1) Scrum Master, (2) Product Owner, and (3) Developer. In a single team, there is only one Scrum Master and one Product Owner. Developers make up for the rest of the headcount.

The Scrum Master is responsible for establishing Scrum. They also help the Product Owner and the Developers in several ways. For example, they help the Product Owner to manage the Product Backlog and facilitate stakeholder collaboration. For the Developers, the Scrum Master can provide coaching in self-management and overall productivity. The Product Owner is tasked with maximizing the value derived from the Scrum Team's efforts, with methods varying across organizations and individuals. Their responsibilities include effective Product Backlog management, involving the development and communication of the Product Goal, creation of clear Product Backlog items, prioritization of items, and ensuring transparency and understanding of the Product Backlog. Developers in the Scrum Team are dedicated to producing a functional Increment in every Sprint. The Developers usually carry out a variety of tasks, including planning for the Sprint, adapting to obstacles during the Sprint, and producing a functional Increment during each Sprint.

3.2 Phases

While individual companies and developers have their own way of producing software and systems, commonalities can be observed in the way that said entities produce software and systems. With global software development becoming increasingly more common, standardized and widely agreed upon phases have become more common as well [19]. The software development life cycle can be split into distinct phases, each consisting of their own distinct tasks that are essential in producing quality software. However, some overlap is to be expected between the phases.

For this thesis, based on Section 3.1, seven individual phases have been identified. These phases are presented in Table 3.1, each accompanied by a short summarizing description.

In the following sections, we will dive deeper into these phases, outline them, describe the processes they involve as well as how they relate to each other from a holistic viewpoint.

3.2.1 Preliminary analysis

A system or a piece of software is a product, similar to something you can find from the shelf of a store. However, apart from a few exceptions, common software

Table 3.1: Software development phases.

	Name	Synonym(s)	Description
P1	Preliminary analysis	-	Conducting a high-level initial assessment of a project, choosing appropriate development model, and identifying business targets.
P2	Requirements engineering	requirements gathering	Gathering, analyzing, and documenting the needs and expectations of stakeholders and possibly legal bodies.
P3	Software design	architectural design, application design	Conceptualization of the overall architecture and structure of the software system. Involves creating detailed specifications for the system's components, modules, interfaces, and data storage.
P4	Implementation	development, coding	Implementation of the requirements and design of the system by programming.
P5	Testing	validation, verification	Systematically evaluating the software to identify and rectify defects or discrepancies using various testing techniques.
P6	Deployment	installation, activation, release	Involves the release, distribution and integration of the new or updated software to end-users or the client.
P7	Maintenance	management	Ongoing activities to support and enhance the software after deployment such as addressing issues discovered in the live environment and implementing updates or patches.

development models rarely consider the business side of product development, even though they are crucial for the success of the entire project [20].

The very first phase of developing a system, product, or software is preliminary analysis. In order to wade through the following phases, as clear a target as possible should be outlined. In fact, this phase can be recognized in projects outside of the software development domain, as it involves generic project management processes like defining the scope of the project, risk assessment, preliminary cost estimation, and possibly an initial high-level design of the product. It can also be used to identify the target audience as well as the product's feasibility in the market. Additionally, the information gathered during preliminary analysis can be used as a basis for choosing an appropriate software development model.

For this phase to be successful, the entity carrying out the analysis should possess at least some technical knowledge in addition to the specific domain knowledge of where the system or software is to be used. It can be difficult, if not impossible, to estimate project costs or to assess risks from a technical perspective if said entity does not understand the software development domain.

3.2.2 Requirements engineering

Most systems and software components have a list of requirements that are required for the software to fulfill certain criteria, whether it be quality related or a legal requirement. According to Cheng and Atlee, these needs are encapsulated within software requirements, and the determination of these requirements is carried out through the process known as requirements engineering (RE) [21].

Achieving success in requirements engineering entails comprehending the requirements of users, customers, and other stakeholders. Additionally, understanding the context in which the developed software will be applied, engaging in activities such as modeling, analysis, negotiation, and documentation of stakeholders' requirements, ensuring the alignment of documented requirements with negotiated ones through validation, and effectively managing the evolution of requirements is also very important. [21]

It is not uncommon for requirements engineering to be characterized as a difficult process. The space where requirements are defined is rather abstract and not constrained, whereas the software development space is more constrained. Requirements may be defined by individuals that are not experts in software development. Requirement engineering processes can be long and burdensome, resulting in a long, and possibly conflicting list of requirements that no longer serves the end-user. [21]

3.2.3 Software design

In order to produce some kind of a plan for where to begin the implementation of a system, a process called software design should be performed. Software design is the process of an agent creating a specification for a design object with the aim of achieving goals, fulfilling requirements, and utilizing a set of primitive components [22]. This is done in an effort to preemptively solve problems that would otherwise present themselves in the implementation phase, but also to envision the overall architecture of the software. This enables the designer or architect to propose and document both high-level architectural design and low-level component and algorithmic designs for the software.

Software designers and architects may stumble upon a variety of different problems across different software projects. However, it is not uncommon for different software projects to have similar design problems that have either been visited or perhaps even solved in the past. Such solutions are referred to as design patterns. Utilizing design patterns in solving problems similar in nature can help in keeping both the design and architecture of the source code more organized. [23]

Much like requirements engineering, software design can be incredibly complicated and nuanced. Software-intensive systems are inherently complex as they involve not only software but people, computers, and other devices. Additionally, software designers and architects are often bound by constraints like time, budget, and other limitations that must be taken into account when designing reliable software. [24]

3.2.4 Implementation

The implementation phase is where the majority or all of the actual programming work is completed. Depending on the chosen software development model, this phase can be iterative or linear in nature. For example, in Scrum, the implementation phase is treated as a black box, where the unpredictable is expected [13]. As a result, the development goals may change during the phase. However, in a waterfall-based model, the implementation phase strictly follows the project requirements and design, which are defined at the beginning of the project.

Implementation is carried out by developers that use a variety of tools. Typically, developers utilize a code editor, which is software that allows the developer to write software quickly and reliably. Code editors commonly provide a way to easily run interpreters, compilers, or debuggers while writing the code. In addition, many code editors provide language intelligence tools using the Language Server Protocol (LSP). LSP, originally developed for Microsoft Visual Studio Code, allows the developer to check for syntax errors as they write the code. It also enables other programming language-specific features like automatic code completion and syntax highlighting, which significantly enhances both the speed of writing code and the correctness of it. [25]

Source code can be written and modified using a simple text editor that comes with the developer's computer's operating system. However, developers also have the option to opt in for integrated development environments (IDE) which feature a source code editor in addition to many other features like debugging and build tools [26]. Depending on the developer's needs, IDEs can be a necessity or an overkill for certain software development projects.

Software source code is often accompanied by documentation, either being directly included in the source code or provided in external documents. The purpose of code documentation is to improve the understanding of the code logic and how it works. This is useful for both existing and future developers as it allows them to familiarize themselves with the source code faster. Furthermore, it also enhances the maintainability of the code. When a developer needs to adjust a part of the code, or perhaps extend a component, documentation provides context and guidance. This can significantly reduce the likelihood of breaking some part of the system and introducing new errors in the process. [27]

3.2.5 Testing

Having written code or built a part of the system that appears functional, the result ought to be tested and reviewed. Testing can reveal existing faults in the software which can prevent these faults and other issues from slipping through from a development stage to production. Such faults are commonly referred to as bugs in the field of software development. Furthermore, testing also validates the quality of the implemented solution. Additionally, regular code reviews can assist developers in catching bugs, staying consistent and maintaining coding standards within the team, and getting a new perspective.

Depending on the project, different things need to be tested. For example, a mobile application usually has a user interface, which means that some kind of usability and accessibility testing is appropriate to assess how easy or difficult the application is to operate for its users. However, this is not the case with a REST API, which has no visual user interface that the user can interact with. Instead, users of the API interact with it using HTTP requests, which imposes different kinds of usability concerns e.g. the intuitiveness of the API and the format of the result.

To make things arguably slightly simpler, testing can be divided into different levels. To give an example, developers might decide to employ three levels of testing: (1) unit testing, where specific sections or functions in the source code are tested, (2) integration testing, where the interface of a single component or subsystem is tested, and (3) system testing, where the entire system is tested against its requirements. By utilizing this kind of a multi-level testing approach, developers can isolate issues more effectively by narrowing or widening the scope of testing per their needs. Additionally, it enables the identification of issues that are difficult to spot when observing the software solely from a single level or viewpoint. [28], [29]

Even the smallest change in the source code of a large system can produce drastic unwanted results if not tested properly. This is where code testing becomes important. However, the reliability of new code, or changes to existing code, can be improved by arranging code reviews: a process where developers examine their colleagues' code to fix mistakes, ensure quality and improve the overall software development process. Not only does this improve the quality of the code, it fosters collaboration within the team, which can be characterized as a rather crucial aspect of software development.

Similar to the implementation phase, code testing can be done as an iterative and dynamic process by compiling and running the source code, observing its behaviour, making adjustments, and repeating as needed. However, this kind of a manual process is laborious and difficult to scale as the software or system grows larger both in size and complexity. Additionally, as time goes on, organizations tend to demand software developers to produce higher quality and more substance in similar length or shorter delivery cycles, a development that has been driven by evolving technology, increasing customer demand and expectations, and the rise of lean approaches and practices like Agile, DevOps, and CI/CD. This means that the need for faster deployment of production-grade code is growing. [30]

The aforementioned circumstances have led to automated tests that are often ran before shipping any code to production. Just like any other code, these automated tests need to be designed and written to cover important use cases of the system or software, whether it be normal usage or an edge case. Automated tests are often integrated as part of a CI/CD pipeline, where the build process depends on the outcome of the tests: if the tests do not pass, the code will not be shipped to production. This is referred to as continuous testing. The outcome is testing being closely linked to the phase we will explore next: deployment. [30]

3.2.6 Deployment

The deployment phase involves making a software application available for its users in a specific environment. Depending on the system, the deployment phase's tasks can vary significantly based on factors such as the complexity of the software and target infrastructure (e.g. cloud, on-premise, or both). With that said, the deployment phase ought to be thought of as a general process that is tailored to the needs of a specific system. This means that deployment can involve activities that are not directly related to software development like notifying stakeholders of deployment contents prior to deployment [31].

A typical deployment process involves (1) running automated tests and checks, (2) packaging the software and its components, (3) distributing the software to the target environment, and (4) installing it. As this is an overly generalized deployment process, the deployment process may in fact be much simpler for smaller systems but vastly more complex for larger enterprise-level systems. For larger systems, the process likely involves configuration of the server, activation of different services or workflows, and post-deployment testing that is specific to the deployment. [31]

The deployment phase is rarely a one-time process. In reality, software and systems requires constant updating and improving. This means that deployments can be frequent for some systems, which highlights the importance of a refined deployment pipeline. This allows developers to focus on producing quality code rather than simultaneously juggling a variety of things in their mind about the deployment process, whether it be testing, packaging, or something else that is critical for the success of deployments. Additionally, a well-developed deployment pipeline enhances consistency, efficiency, quality, speed, and scalability of the overall development lifecycle.

3.2.7 Maintenance

Software maintenance phase consists of operating, updating, fixing, and improving the system once it has already been deployed at least once. Typically, this phase begins after the first deployment and it continues until the system is deprecated and ultimately taken down. In reality, the maintenance phase is really just a revisit to the previous phases when they are needed. However, the maintenance phase does introduce some new tasks for developers.

The maintenance needs of a system are mostly defined by the system itself. Some systems may require constant monitoring, while others rely on constant configuration and frequent updates to the software code. To give an example, large systems that (1) are critical to society and infrastructure, or (2) need to handle varying and massive amounts of traffic require constant monitoring that the system is up and running. Another example is banking, a heavily regulated sector, which has its software and systems under constant pressure from regulators. Furthermore, different systems and architectures need different kinds of monitoring, whether it be monitoring of CPU or network usage. For example, an on-premise software system, which is self-hosted, is likely to require constant monitoring. However, a cloud-based solution might move the responsibility of infrastructure upkeep to the cloud platform provider, removing it either partially or completely from the development process from the developer's perspective.

3.3 Tasks

By analyzing the software development phases in Section 3.2, we can infer that each phase is ultimately comprised of certain tasks. For example, P4 (the implementation phase) usually consists of a lot of programming which can be completed using a simple text editor that comes with the operating system, or a full-fledged IDE. These tasks are outlined in Table 3.2 in no particular order. The tasks have been identified based on the phases presented in Section 3.2. Again, depending on the chosen software development model, individual tasks may be found in different phases, and they might occur in multiple different phases in varying order. Some tasks can also be split into smaller subtasks.

Table 3.2: Summary of tasks in software development phases.

	Name	Phase(s)	Description
T1	Gathering requirements	**P1**, **P2**	Gathering and documenting software requirements.
T2	Architectural design	**P3**	Architectural design of a system or software using documents or visual illustrations.
T3	User interface design	**P3**	Designing the visual user interface of an application.
T4	Programming	**P4**, **P5**, **P7**	Writing the source code of a system, possibly including automated tests.
T5	Debugging	**P4**, **P5**, **P6**, **P7**	Searching for faults and bugs in the system or software and fixing them.
T6	Deployment	**P6**	Making the system or software available for use to its users.
T7	Version control management	**P4**, **P5**, **P6**, **P7**	Managing different versions of the system or software.
T8	Monitoring	**P6**, **P7**	Monitoring the system's behaviour.
T9	Documentation	**P1**, **P2**, **P3**, **P4**, **P5**, **P6**, **P7**	Documenting important information produced during the entire software development lifecycle.
T10	Project management	**P1**, **P2**, **P3**, **P4**, **P5**, **P6**, **P7**	Management of the software project, including things like prioritization of tasks, project coordination, estimating timelines and tracking the development progress.

It is also worth noting that some tasks may share some of the core activities that make up the task. To give an example, the argument can be made that programming and debugging are the same thing in a different context, as both of them usually involve making adjustments to the software's source code. However, the context in this instance is significant. The goal of the task is not the same, even though they are both making adjustments to the source code. To extend the same example, programming is innately a creative task that focuses on implementation of new features, functionalities, or other improvements. Debugging as a task, on the other hand, is meticulous and problem-solving oriented, and usually it involves the identification of bugs in existing source code and then, once identified, eliminating them.

3.4 Summary

In this chapter, we have taken a holistic approach to software development by exploring software development models and phases found in the models. In addition, we have identified tasks that are performed during said phases. Thus far, we have identified seven phases through analysis of five individual software development models. These phases are executed in various lengths and in different points of the project's lifetime depending on the chosen software development model. Furthermore, we have identified 10 tasks that are performed during the aforementioned phases. Similar to phases, these tasks can be performed at different lengths and order. Some models also prioritize different tasks, and some allow flexible rotation of tasks throughout the system's lifecycle.

With agile methodologies being one of the current predominant ways of composing the software development lifecycle, this thesis pays extra attention to how AI can be leveraged within agile frameworks to maximize efficiency and effectiveness. By focusing on the unique characteristics and requirements of agile practices, the research explores how AI can potentially support and enhance iterative development, continuous integration, and rapid feedback loops.

4 Artificial Intelligence and Machine Learning

In this chapter we lay the groundwork for understanding some of the key topics of the thesis better: artificial intelligence and machine learning. Because AI as a term is very broad and difficult to define, one of the primary objectives of this chapter is to provide the reader with an explicit and concise understanding of what the thesis is examining. Additionally, we define the relevant terms used in this thesis, and dive deeper into machine learning which powers most of the current modern AI technology.

4.1 Artificial intelligence

Artificial intelligence has been the subject of discussion on many forums, interviews, news, and social media platforms. It has also played the role of an antagonist, usually a scourge-like phenomenon, in many popular movies. AI as a topic clearly has the power to grasp the attention of the general audience because it has the potential to disrupt and revolutionize the world. The hype is currently fueled by promises of AI software developers like Devin, autonomous vehicles, and solutions to many of the global issues of the modern world [32], [33]. While some of it is hype, there may be a lump of truth to it.

Being one of the biggest buzzwords of the decade, the term *artificial intelligence* tends to be widely misused [34], [35]. Additionally, discussion on the definition of AI tends to expand into philosophical topics rather than computer science. With that said, while there are many different approaches to defining artificial intelligence in the scientific domain, for this thesis we will be using IBM's concise definition: "Artificial intelligence, or AI, is technology that enables computers and machines to simulate human intelligence and problem-solving capabilities." [36]

4.2 Machine learning

Machine learning (ML) is a field of artificial intelligence that aims to study and develop algorithms that can effectively generalize, deduce, and perform tasks without being explicitly told to do so. Instead, they infer instructions from a preferably large set of examples [37]. As described by Jordan and Mitchell [38], ML focuses on two questions: (1) how can one construct computer systems that automatically improve through experience, and (2) what are the fundamental statistical computational-information-theoretic laws that govern all learning systems, including computers, humans, and organizations?

In the last twenty years, machine learning has undergone dramatic advancements, evolving from an experimental concept in laboratories to a practical and widely used technology in various commercial sectors. As a field of study within artificial intelligence, machine learning has emerged as one of the preferred approaches for developing useful software used in computer vision, speech recognition, natural language processing, robot control, and many other applications [38]. In fact, most of the hype surrounding these AI applications is powered by the quite recent advances ML, particularly in deep neural networks [39].

With respect to this thesis, the main topic of discussion is enhancing software development with machine learning, as the solutions presented in this thesis are based on machine learning. However, future developments in the AI domain are difficult, if not impossible, to forecast with any good certainty. With that said, it is important to understand artificial intelligence as an umbrella term. Future AI-based solutions might not fall under the machine learning field of study at all, but rather belong in some other subfield of artificial intelligence.

4.3 Approaches to machine learning

There are many different ways of classifying and categorizing machine learning. One could, for example, attempt to draw the dividing lines based on what problem the model is attempting to solve. However, we will begin our categorization by dividing machine learning into four distinct approaches of training an ML model: (1) supervised learning, (2) unsupervised learning, (3) semi-supervised learning, and (4) reinforced learning.

4.3.1 Supervised learning

Supervised learning builds and shapes the model by assessing data that describes some input, sometimes referred to as features, that produces some output, also known as labels. The result of supervised learning is a mathematical prediction model that can be used to make predictions on unknown data based on the training it has received. [39]

For instance, let us assume that we would like to create a model for recognizing different animals from pictures. To employ supervised learning, we can provide the algorithm with a dataset that contains pictures of different animals as well as the names for each animal. We can then let the algorithm try to figure out what are the distinct characteristics of each animal. This is referred to as classification: the output variable is categorical, meaning it is part of a discrete set of classes or categories. After training, the model can be tested to verify if the training has

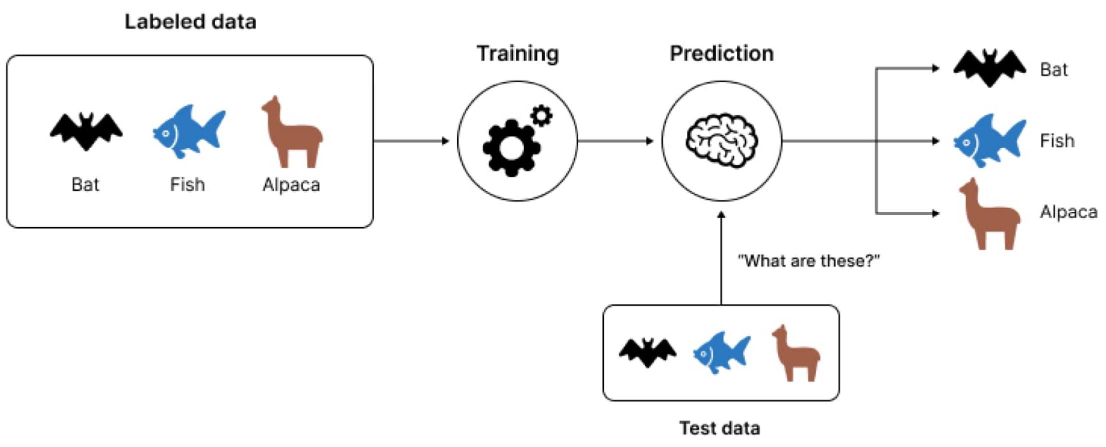

Figure 4.1: Simple illustration of a supervised learning process.

worked. A simple illustration of this process is depicted in Figure 4.1.

Supervised learning is also useful in predicting real or continuous values, where a relationship between two or more variables can be recognized. To give an example, we could attempt to predict the grade of a student based on the hours that student has spent on the studies. We could also try to estimate the price of a home when given parameters about the property and neighbourhood. A problem of this nature is called a regression problem.

By extension, supervised learning is a suitable solution when the goal is to classify, categorize, and perform regression on datasets. On a high level, supervised learning is particularly useful in scenarios where labeled data is available, and the goal is to learn the relationship between input variables or features and corresponding output labels. [40]

4.3.2 Unsupervised learning

Unsupervised learning algorithms are arguably the opposite of supervised learning algorithms. As the name suggests, unsupervised learning is based on self-learning algorithms: they attempt to find structures or commonalities in data that has not already been labeled, classified or categorized. Hence, they are not given any instructions on how to process and work with pieces of data.

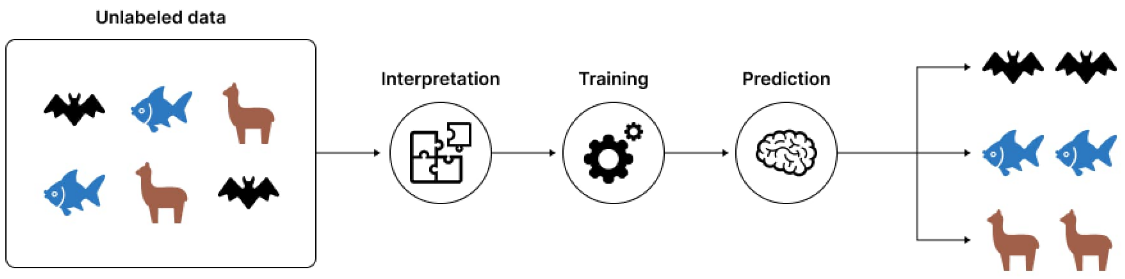

Figure 4.2: Simple illustration of an unsupervised learning process.

Unsupervised learning has different kinds of benefits when compared to supervised learning. To give an example, consider a retail company that wants to deeply understand its customers' behavior in order to launch targeted marketing campaigns, offer personalized recommendations, and optimize product placements in stores. They possess a large dataset about customers' purchase history. By applying unsupervised learning techniques, the company can identify customers with similar purchasing behaviors and group them together. With that said, unsupervised learning is a suitable approach when the goal is to cluster or group similar data points together, reducing datasets to smaller but important subsets, and detecting anomalies in datasets.

4.3.3 Semi-supervised learning

As the name implies, semi-supervised learning combines supervised and unsupervised learning by using both labeled and unlabeled data to train machine learning models for purposes like classification and regression. Technically it is not a distinct approach to training an ML model but rather a combination of two basic approaches. Though it is common to employ semi-supervised learning for similar use cases in which supervised learning could be used, it brings a benefit to the mix: the data does not have to be completely composed of labeled data. [41]

In fact, the primary distinction between supervised and semi-supervised learning is that the former can only be trained using fully labeled datasets. However, the latter uses both labeled and unlabeled data samples in the training process. The added benefit is that the supervised algorithm can be enhanced and supplemented by using an unsupervised learning algorithm in conjunction. Labeled datapoints provide the basis for learning. Then, the unlabeled datasets can be incorporated when the basis is built. [41]

Semi-supervised learning is especially useful in situations where labeled data is scarce, expensive, or otherwise difficult to obtain, but the unlabeled data is abundant. In such cases, a small amount of labeled data in conjunction with a larger amount of unlabeled data can be enough to train a model effectively.

4.3.4 Reinforcement learning

Reinforcement learning is the third basic approach to training a machine learning model. The fundamental concern in reinforcement learning is determining the most optimal actions agents should take in an environment to maximize cumulative reward.

As illustrated in Figure 4.3, learning in this paradigm occurs in three steps: (1) the agent performs actions, (2) receives feedback in the form of rewards or penalties from the environment based on its actions, and (3) adjusts its strategy to maximize cumulative rewards over time. Similar to living organisms, this learning paradigm learns through trial and error while attempting to find the most effective way for achieving the desired outcome.

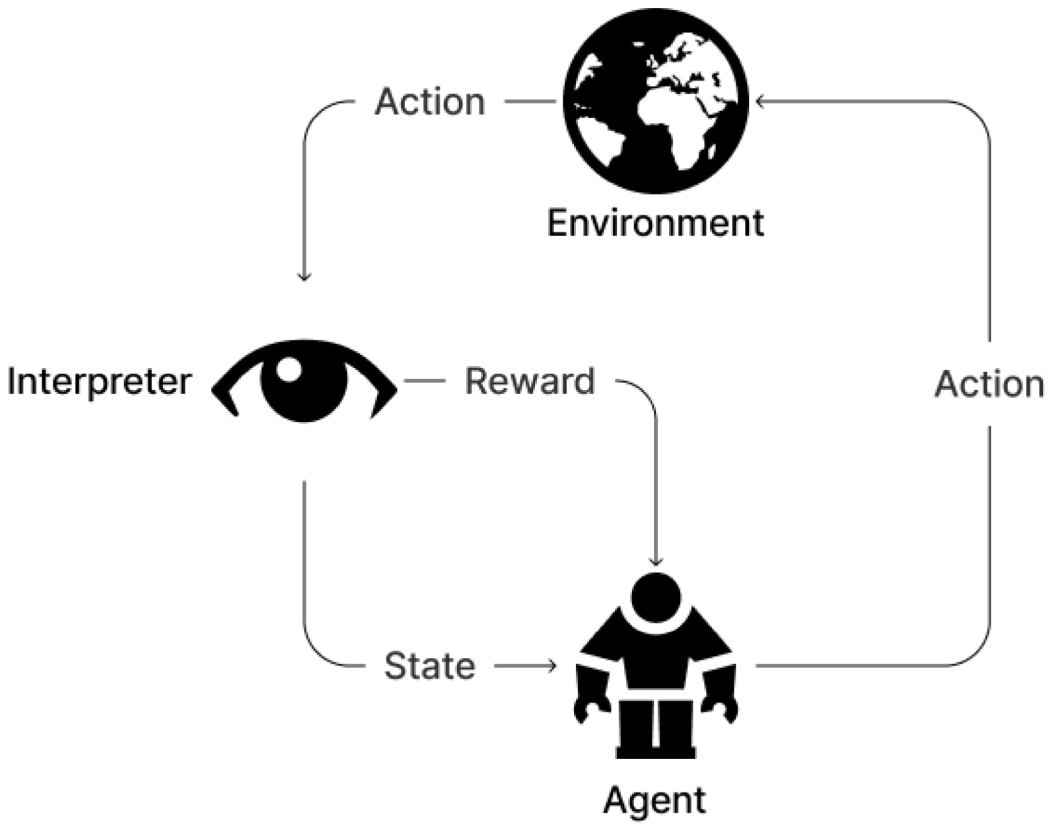

Figure 4.3: Simple illustration of a reinforcement learning process.

4.4 Machine learning models

In machine learning, a model is essentially a mathematical formula that can be used to make predictions for a given data set. To simplify, the model will attempt to guess what comes next. During training, the model will be iteratively shaped to minimize the chance errors when making predictions. Various different types of models can be produced by choosing different types of training methods: In this section, we will discuss four distinct model types: (1) neural networks, (2) decision trees, (3) regression analysis, (4) Bayesian networks.

4.4.1 Neural networks

Recent advancements in machine learning, particularly in the development of artificial neural networks, are behind most of the modern day AI hype [39]. Taking inspiration from the human brain, a neural network is comprised of interconnected nodes or neurons arranged in layers that establish connections between inputs and desired outputs. Through iterative adjustments, the machine learning model is trained to optimize the strength and thickness of these connections, ensuring that provided inputs correspond to the desired responses. This makes neural networks ideal for pattern recognition.

In a neural network, neurons, the core processing units, are arranged into layers. This is illustrated in Figure 4.4. Each layer can perform various different transformations on their given inputs. There are three categories of layers: (1) the initial input layer, (2) intermediate hidden layers, and (3) the final output layer. However, the hidden layers are optional. A network that contains at least two hidden layers is referred to as a deep neural network.

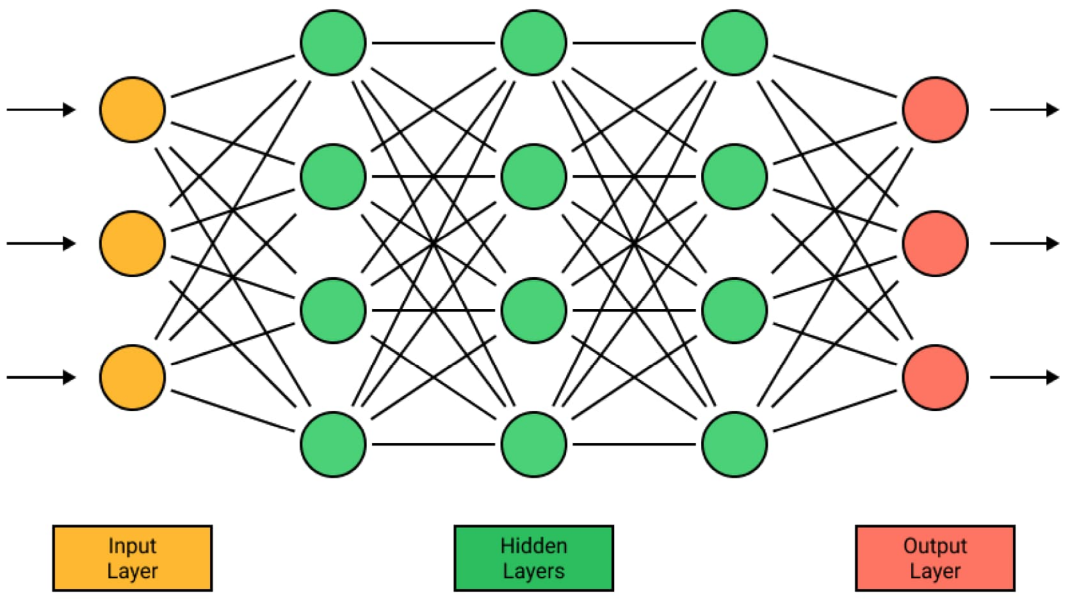

Figure 4.4: The three layers of an artificial neural network.

Let us go through an example that is illustrated in Figure 4.5. Consider a situation where we would like to reliably differentiate an image of a square, a circle, and a triangle from one another. We can create a neural network for achieving this task. The input layer receives an image of one of the three shapes, in this instance, a circle. We provide the network with a 28 by 28 pixel image of a circle to the input layer, giving us 784 individual neurons in the input layer. Each neuron represents a single pixel, denoted by $x_1, x_2, x_3, \dots x_{784}$ in Figure 4.5. As mentioned before, neurons of the first layer are connected to the proceeding layer. Each of these connections is assigned a numerical value to illustrate the strength of the connection, known as weight. In Figure 4.5, weights are denoted by $w_1, w_2, w_3, \dots w_{1566}$. The value of each input neuron is multiplied to the corresponding weights. Then, their sum is given as input to the neurons in the proceeding layer. A bias is added to the sum: an additional parameter that allows the neuron to output values other than zero. Finally, this resulting value, sometimes referred to as the activation, is passed to a threshold function called the activation function. This function determines if the neuron is activated or not. Once a neuron is activated, the data is passed to the neurons of the proceeding layer. This process is repeated until we reach the final output layer, where the result is determined.

4.4.2 Decision trees

Given their simplicity and intelligibility, decision trees are one of the most popular approaches to supervised learning, and they have been used extensively in statistics, data mining as well as machine learning [42]. They are non-parametric and highly flexible [40]. As illustrated in Figure 4.6, decision trees draw conclusions based on a set of observations.

There exists two main types of decision trees: (1) classification trees, and (2) regression trees. While the two have some similarities and are often grouped under

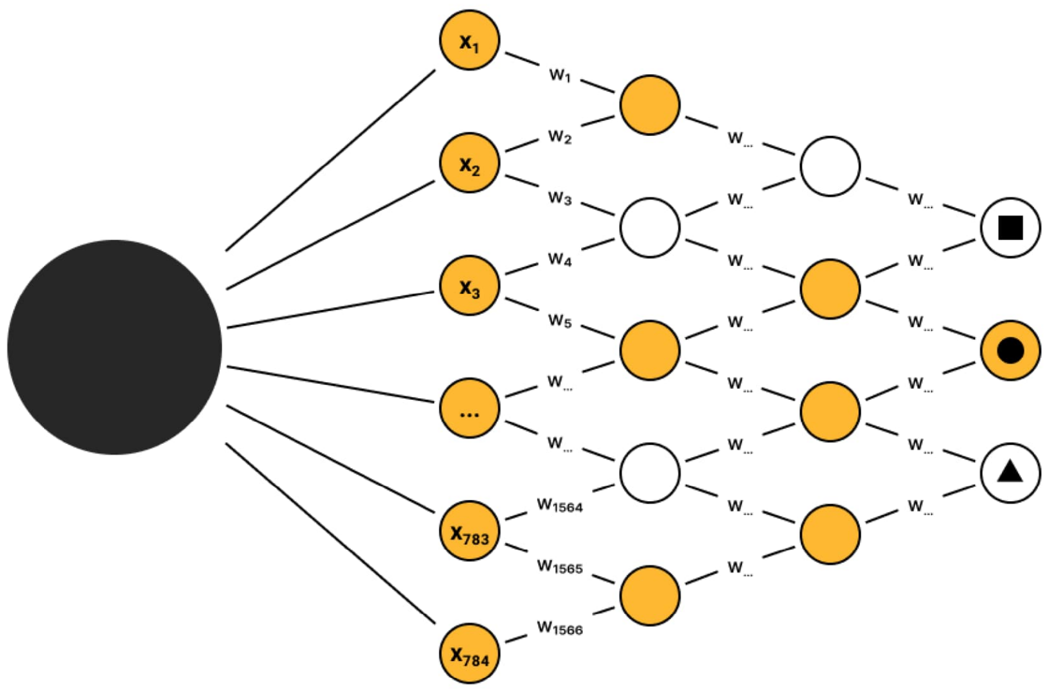

Figure 4.5: An illustration of an artificial neural network that is able to differentiate between a square, a circle, and a triangle.

the umbrella term Classification And Regression Tree (CART), they have their own unique benefits and use cases. To simplify, classification trees are designed for scenarios where the target variable can take a discrete set of values. Within these tree structures, class labels are represented by leaves, while branches symbolize combinations of features leading to these class labels. However, when the target variable can take continuous values, commonly real numbers, the corresponding tree models are referred to as regression trees. Furthermore, the concept of regression trees can be broadened to include any type of object containing pairwise dissimilarities, such as categorical sequences.

To improve prediction accuracy, multiple decision trees can be combined to create a random forest, which employs many different decision trees. If, for example, some of the decision trees in a random forest are not relevant in some instance, they can

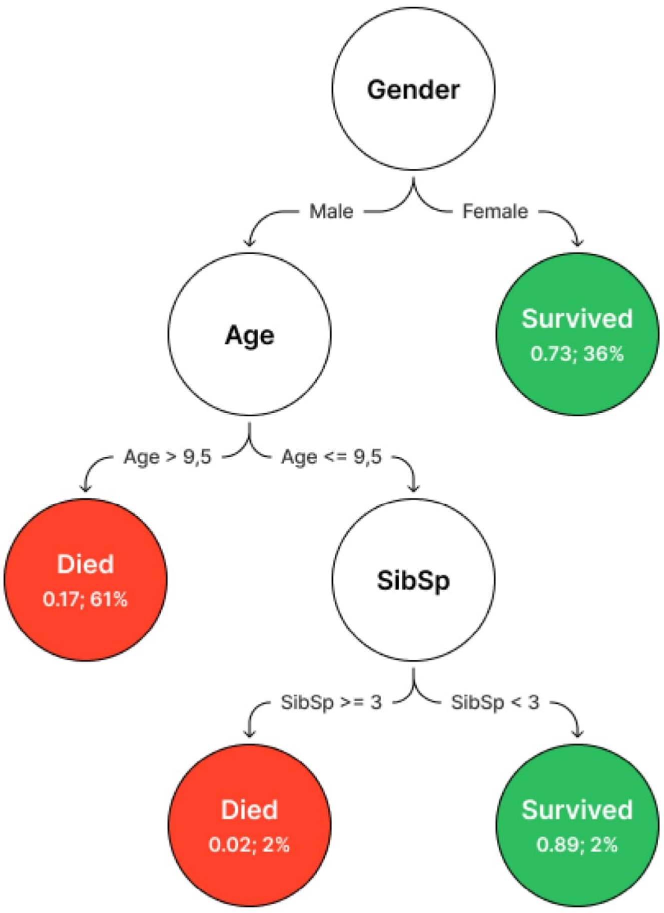

Figure 4.6: A decision tree depicting survival probability of passengers on Titanic, where *SibSp* is the number of siblings or spouses onboard, and the figures in each leaf describe the probability of survival and the percentage of observations in the leaf.

be ignored. This can help with overfitting: an issue that arises when over-complex trees that do not generalize well from the input dataset are grown.

4.4.3 Regression analysis

Regression analysis encapsulates a common behavior that we have briefly discussed in, for example, Section 4.3.1. It is the set of statistical processes used to assess relationships between a dependent variable, also known as an outcome or output, and one or more independent variables, also known as features or inputs. Variables are pieces of data that describe an attribute or characteristic of an object.

The most commonly used form of regression analysis is linear regression, where one attempts to fit a line to best match the given variables. A simplified illustration of this is shown in Figure 4.7. When there is only one independent variable, the regression model is called simple linear regression. However, when there are more than one independent variables, the regression model is called multiple linear regression.

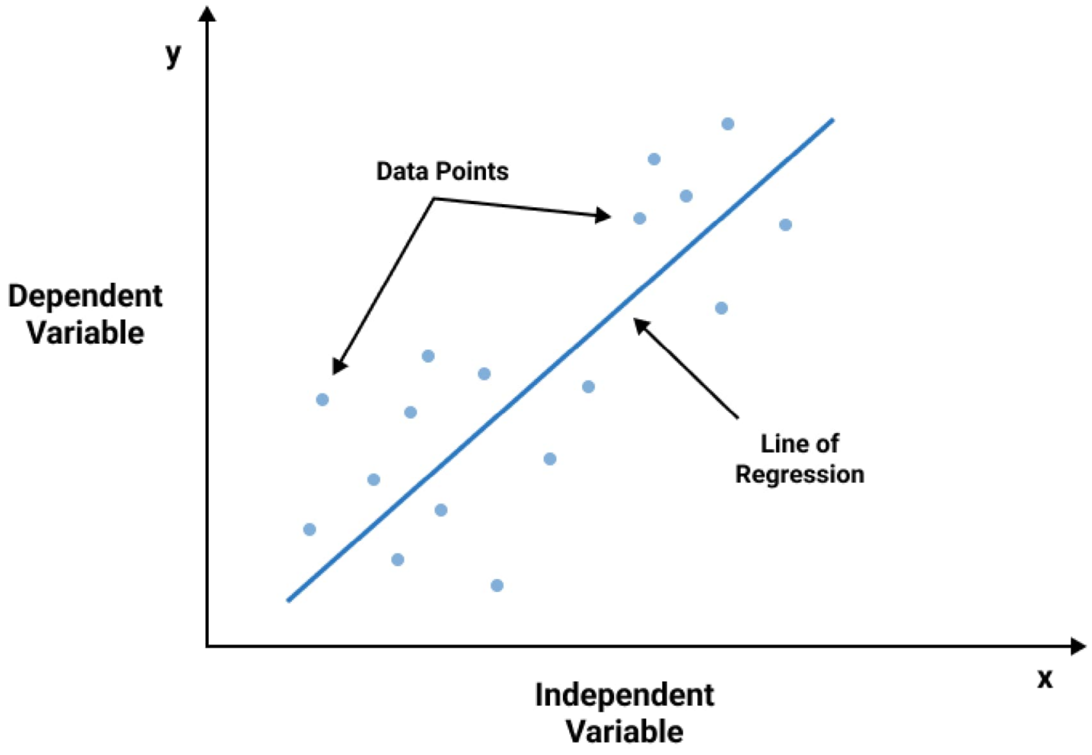

Figure 4.7: Depiction of linear regression.

Once the regression model is trained on a dataset, it can be used to make predictions on the dependent variable's values when given new input data. Evaluation metrics such as mean squared error (MSE), mean absolute error (MAE), or R-squared are often used to assess the model's performance and accuracy.

4.4.4 Bayesian networks

A Bayesian network is a probabilistic graphical model that represents some set of variables and their conditional dependencies using a directed acyclic graph (DAG). In essence, each graph is composed of three elements: (1) nodes, (2) edges, sometimes referred to as arcs, and (3) probability functions. A node is simply a representation of a variable that may be observable, an inferred latent variable, an unknown parameter or some hypothesis. An edge is formed between two nodes, and it represents direct dependencies between the variables. Finally, each node in the network is also accompanied by an associated probability function which quantifies the probabilistic relationship between that node and its parents, nodes that are pointing to it in the graph. [43], [44]

Utilizing a Bayesian network has several benefits. For example, as the model encodes dependencies among all variables, it is capable of handling situations where the input dataset is incomplete and missing some entries. Additionally, Bayesian networks allow one to learn about causal relationships. [44]

Bayesian networks can be useful in predicting causalities [44]. Given an event that occurred, Bayesian networks can be used to predict the likelihood that any one of the possible known variables were the cause of the event, or at least a contributing factor. To give an example, in medicine, one could attempt to determine the contributing factors of a disease using Bayesian networks.

4.5 Summary

In this chapter, we have defined artificial intelligence in the context of this thesis and explored the world of machine learning. We have outlined and compared common approaches to training a machine learning model as well as introduced some of the different types of models that can be produced using different training approaches.

The contents of this chapter have been laid out in an effort to provide an understanding of what goes on under the hood of the AI-based tools and techniques that we will be discussing in the following chapters. In the context of this thesis, we should now have at least an adequate understanding of what powers most of the modern AI-based tools. This is important because, in order to see through the veil of the AI hype, we need to be able to understand not only the potential of these tools but also their limitations in a realistic manner.

5 Utilization of Artificial Intelligence

In this chapter, we innovate and concretize potential utilization cases for AI-based tools and methods in software development. We base each potential utilization on phases and tasks discussed in the Chapter 3. Furthermore, we take into account the current realistic capabilities of ML models as discussed in Chapter 4. The list of scenarios are summarized in Table 5.1.

For each utilization case, we will either briefly go over the task that is being enhanced or the potential issue that the utilization case is attempting to solve. When applicable, we will also introduce some simple examples of each utilization case. These examples are referred to as scenarios: an instance where the utilization might result in a meaningful enhancement. These scenarios are defined in an effort concretely identify potential enhancements that can be discussed in the interviews. The aim of this chapter is not to develop an exhaustive list of scenarios where AI could provide an enhancement. Rather, we aim for a list of scenarios that covers basic software development phases and tasks.

5.1 Source code generation

Arguably, the most famous utilization case of AI is code generation. To be exact, it involves the use of ML models to generate correct and functional software code. These generated snippets of code are contextually based on the current workspace, including the current code base, libraries being used, and the syntax of the pro-

Table 5.1: Proposed AI utilization scenarios for software development

	Category	Target phase(s)	Target task(s)
S1	Source code generation	P4, P5	T4, T5
S2	Software documentation generation	P4, P5, P7	T4, T9
S3	Software documentation generation	P4, P5, P7	T4, T9
S4	AI-driven bug triage and debugging	P5, P6	T10
S5	AI-driven bug triage and debugging	P5, P7	T5
S6	AI-driven software testing and reviewing	P4, P5, P6, P7	T4, T6
S7	AI-driven software testing and reviewing	P4, P5, P6, P7	T10
S8	AI-enhanced CI/CD	P4, P5, P6, P7	T4, T6, T7, T8
S9	AI-driven specification composition	P1, P2	T1, T9
S10	AI-driven specification composition	P1, P2	T1, T2, T9
S11	AI-enhanced agile development	P1, P2, P3, P4, P5, P6, P7	T10

gramming language. Because the underlying ML model is able to make suggestion in the context of the workspace, it has the potential to provide quite accurate and contextually relevant suggestions. In some trivial cases, generative AI can generate fully functional code on its own without any help from a developer. However, at this time, it is mostly considered an assistive tool for developers in the programming task, making suggestions and generating the bulk of code. To give an example, this could mean generating the skeleton for a specific loop structure or a complex if-condition.

Code generation solutions are fairly easy to integrate to an existing workflow without substantial overhead expenses. Modern code editors like Visual Studio

Code and Neovim offer ready-to-go plugins that can be installed with the click of a button. This means that companies and developers are not required to train their own models to get started, but can instead rely on already trained general-purpose programming models. One example of such a model is the OpenAI Codex model, which powers GitHub Copilot [45].

Generally speaking, the process of generating code using a code generation solution like GitHub Copilot is simple: a prompt is given to the model which then responds with an output. For example, the prompt can be a code snippet that is incomplete, and needs to be finished. The model will then proceed to making a prediction about what the developer wants, and responds with an output in the context of the workspace. However, the prompt does not have to be just code. It can be accompanied with a description of what the code should do. In fact, the prompt can be just that: a description of what the output code should do, without any input code snippet. This can be beneficial in a situation where the user providing the prompt is not familiar with the current programming language, does not possess the skills of programming, or is simply not tech-savvy at all. [46]

Scenario 1

A developer would like to increase their productivity by eliminating the writing of repetitive and structurally simple code. They decide to utilize generative AI by either (1) starting to write the to-be-generated code themselves and allowing generative AI to automatically fill out the rest of the code, or (2) write a comment describing the desired functionality of the code and allowing generative AI to write it from scratch. This might potentially increase the speed of development as they have to write less code, and in some instances, juggle less things on their mind when writing code.

5.2 Software documentation generation

In a very similar fashion to code generation, ML models can be used to generate documentation for both new code and existing code bases. In fact, GitHub Copilot can be used to accomplish this task. When given a prompt of some code and an instruction to provide documentation for the code, Copilot will successfully generate documentation, usually in the form of a comment, also known as a docstring. Again, generative AI can be quite useful in this instance as it can help in generating the bulk of the documentation.

To give an example, in some code documentation formats, it can be quite laborious to write the description of each parameter to a function, or provide examples of what the function should output when given certain input parameters. In cases like this, generative AI can enhance and expedite the process of writing descriptive and useful documentation by a significant amount. In addition to Copilot, there currently exists other similar products that specialize in generative AI assisted code documentation like Docify and DocuWriter. [46]–[48]

Scenario 2

A developer is tasked with documenting source code that is currently missing documentation. They are required to document the functionality of functions, input parameters, and output return values. Additionally, the documentation should follow a specific format throughout the source code repository. The developer uses generative AI to complete the task for each function: they provide the function as input and ask the ML model to generate the documentation in a specific format.

It is also worth highlighting that generative AI can be used as (1) an assistive tool alongside the developer to generate the documentation as new code is written, or (2) as a standalone tool to generate documentation for a code base that does not have any source code documentation in the code as docstrings.

Scenario 3

A developer would like to generate API specification documentation for a fairly large REST API. There are automatic REST API documentation tools that are not based on AI, but they require certain annotations to exist in the source code to be able to generate documentation automatically, which have been omitted from the source code. To achieve this, the developer utilizes generative AI to include the annotations in a given syntax for each endpoint in the REST API.

5.3 AI-driven bug triage and debugging

The quality of the produced software is largely dependent on the correct functionality of the software features. This means that not only does the software need to do the intuitively correct action but it must also perform correctly under the hood without any bugs. However, bugs tend to end up in production environments even in the most careful and cautious deployment processes.

When a software bug is located, it is typically placed into a prioritized list of bugs that need to be fixed. After some time, a suitable developer is assigned to fixing the bug, and proceeds to reproduce it, locate it, and ultimately, fix it. The aforementioned series of events can be divided into two distinct activities: software bug triage and debugging.

Software bug triage (SBT) is the process of prioritizing and assigning reported bugs to the appropriate fixers in a timely manner. This is easier said than done. Different developers may be familiar with different code bases, different frameworks and software libraries, and different tools and methodologies. They may be in different stages of their career and possess different levels of expertise. Therefore, it is not feasible to assign a bug to just any developer.

The software bug triage process can be enhanced or fully automated using AI to identify the most appropriate developers for handling each reported bug in a much faster way. A machine learning model can be trained to achieve this task using various types of data such as developer profiles, relevant code repositories, and information about bugs fixed in the past. [49]

Having an automated software bug triage system can be quite a fruitful enhancement for companies. Not only does it help the bug triage managers, who may or may not be tech-savvy, in selecting the appropriate developer for fixing the bug, it will do it much faster. Over time, the model will also accumulate knowledge, and it will likely make predictions more accurately when assigning developers to work on bugs. Additionally, as an automated system, it is innately scalable, making it a well-suited solution for large-scale software projects in the enterprise domain. Overall, such a system will very likely enable companies to minimize costs related to software bug triage dramatically.

Scenario 4

A bug triage team responsible for prioritizing and assigning the fixing of bugs to the right developers is facing a bottleneck in their process. Hence, they need to expedite the process in a reliable way. The team employs a predictive ML model that is able to predict and choose a developer from a pool of developers that is most fit for the task. Additionally, they might employ AI to also assess the priority of the bug, and placing it into an appropriate position in the list of bugs.

Once a bug is assigned to a developer, the developer begins the debugging process. This process can be as simple as fixing an incorrect constant value, invalid variable type conversion, or altering the order of parameters given to a function, which can take less than a minute to fix. However, in more complex cases, debugging a single bug can take any amount of time from days to months to even years.

The process of debugging can be very manual in nature, and it requires a certain skillset from the developer, which also highlights the importance of SBT. Bugs can exist in different ways: whether it be in the syntax of the code or at runtime, the latter of which is arguably more difficult and complex to debug. Different programming languages provide different tools for debugging. Some, for example, print out a more verbose stack trace than others which aids the developer in locating the point where the bug is introduced. Not only does the developer need to be familiar with the language, they need to understand the environment and the system as a whole, including its own specific intricacies and nuances, to be able to debug efficiently.

To make manual debugging easier, an appropriate ML model can be utilized. For example, it can be useful to use an ML model to analyze a stack of error logs to get a more verbose and concise description of the error. Furthermore, the model could also be supplied with the source code in addition to the error logs, allowing the model to both pinpoint the error and provide possible fixes for the error. Even if the model is not able to come up with the correct fix, it can guide the developer towards the actual fix for the error.

Scenario 5

A developer is tasked with fixing a bug. The developer is fairly familiar with the code base, but in this instance, the error resides in a part of the code base that the developer is not that familiar with. To expedite the process, the developer employs an ML model to analyze a fairly large stack of error logs to get a clear description of what is causing the error to occur. Furthermore, the developer also requests possible fixes for the error from the ML model, which provides the developer with a starting point for coming up with a fix.

5.4 AI-driven software testing and reviewing

As discussed in Section 3.2.5, software testing is a crucial part of software development. However, writing tests can be time-consuming and difficult, and software development teams need to deploy fast. AI can be of help in this crucial step.

An appropriately trained ML model can analyze changes made in a code base. Based on the model's analysis, fully functional test files can be generated for covering at the very least some basic unit tests. This removes the bulk of the work from developers, allowing them to spend less time writing repetitive boilerplate tests and to focus on deploying new features and improvements.

Scenario 6

A developer has pushed new code to a code base. To ensure the functionality software now and in the future, software tests need to be written that test the new code. The developer employs an ML model to suggest relevant tests for the changes and generates test files that can be easily integrated to the existing testing scheme using generative AI.

AI could potentially be used for code reviews as well. Similar to the AI-driven bug triage proposed in Section 5.3, generative AI can assist authors in selecting an optimal reviewer from a pool of developers: someone who is well-acquainted with the code base and more likely to identify critical issues in a specific context, whether it be the code base itself, a given programming language, or a specific external package or module. While picking the most suitable code reviewer for a specific code change can be challenging for a human, an ML model has the potential to take a holistic analytical approach by analyzing, for example, the history of code changes and the project's contribution graph to select the most suitable reviewers. If the reviewer is appropriate, they are arguably less likely to overlook the review request and delegate it to someone else. If the reviewer is not appropriate, they might also provide inadequate feedback or neglect a critical issue.

Scenario 7

A developer has pushed new code to a code base. To ensure adherence for coding standards and maintain a high code quality, a code review is to be held. However, a reviewer needs to be selected from a pool of developers. The team employs generative AI to select a reviewer based on their attributes and suitability. This takes the load off from the team needing to manually choose a reviewer for each code change, and has the potential to improve the quality of the code in the long run.

5.5 AI-enhanced CI/CD

The ideal CI/CD pipeline leverages automation of tasks across the entire deployment process. This can include running automated tests, deployment health checks, checking production system logs, or looking at monitoring charts. However, depending on the complexity of the system and the deployment process, this can take a lot of time if done manually by developers. Complex manual deployment processes are also prone to human errors.

To decrease the workload of developers in charge of deployment, AI could be utilized. A potential utilization would be to incorporate AI generated testing techniques, as discussed in Section 5.4, as part of the CI/CD pipeline. This could mean, for example, (1) assessing the results of existing tests, or (2) using AI to generate tests on the fly as new code is deployed. Furthermore, AI could be utilized to monitor system logs for a period of time post-deployment to catch any anomalies that have potentially made their way to production.

Scenario 8

A developer has made changes to a code base and would now like to deploy the changes. The project has an existing CI/CD pipeline that runs unit tests, if any. Upon success, the pipeline merges the code changes into production. The developer would like to improve the pipeline by making it more efficient and reliable. To achieve this, the developer employs generative AI to generate simple unit tests on the fly as part of the CI/CD pipeline, that are then added to the code base and ran upon deployment, including all following deployments. They also utilize AI to assess the production system logs for a certain period of time after deployment, to make sure that any anomalies are caught as soon as possible.

5.6 AI-driven specification composition

Most, if not all, projects produce some kind of documentation throughout the project's lifecycle. Software development projects are no exception. For example, the results of preliminary analysis are often documented, containing information such as the project's end user, risks, budget, and timeline.

As organizations strive to stay competitive, the ability to come up with an initial analysis of a project's feasibility, risks, scope, and potentially its budget become increasingly more valuable. For this purpose, AI could be utilized to produce an initial analysis of the project, providing a basis for further and more thorough analysis performed by humans.

Scenario 9

In the early stages of a new software project, a project team needs to conduct a thorough preliminary analysis to understand the project scope, feasibility, and potential risks. They decide to incorporate AI to enhance their analysis. They leverage AI, trained with historical data on similar projects and topics, to quickly map and estimate the potential risks, as well as assess the feasibility of the product. While it is difficult at this stage, they could also estimate the scope of the project using AI, allowing for better budgeting and timeline estimates.

In the dynamic and competitive landscape of software development, accurately capturing and prioritizing requirements through requirements engineering is essential for delivering features that meet user needs and business objectives. This process could also potentially be enhanced using AI. After gathering initial requirements, AI could be utilized, for example, to generate descriptive and more verbose scenarios for each requirement. This would allow the requirements to be encapsulated in a way that reflects an actual scenario that could take place when a user is using the application. This could potentially improve the accuracy of the requirement when it is eventually implemented during development. Additionally, if proper input data such as stakeholder importance and development costs are available, the requirements could potentially be prioritized using AI.

Scenario 10

A software development team is tasked with creating a new feature for their application. To ensure they capture all necessary requirements, the team decides to leverage AI in their requirements engineering process. After drafting the initial requirements, AI helps the team by drafting detailed requirement specifications or scenarios. Finally, they utilize AI to prioritize the requirements on factors like stakeholder importance, development complexity, and potential impact.

5.7 AI-enhanced agile development

As described in Section 3.1.5, agile development methodologies involve an iterative approach and a rather short feedback loop to best accommodate a rapidly changing environment. Because the change is rapid, it can be difficult to prioritize tasks, manage the backlog, plan for the future, and estimate the length and workload of tasks. Agile methodologies like Scrum also involve quite a bit of routine tasks and ceremonies that can take up a lot of time to prepare and execute, even if done properly. To enhance the process and alleviate the aforementioned issues, AI could potentially be incorporated to the process.

To improve task prioritization and planning, an appropriate ML model could be utilized. The model could be used to analyze historical project data to make a prediction on how much time and resources a single task is going to require. This can help teams working in an agile way to, for example, estimate and plan user stories and sprints more accurately and efficiently. Furthermore, AI could be used to refine the backlog by prioritizing items based on business value and team capacity. This could improve the efficiency of the team, as they have to work less on prioritization, and more on producing and deploying high quality features.

Scenario 11

The Product Owner of a software development team would like to enhance the process of prioritizing user stories before each sprint. The backlog has grown to be quite large over the course of several months, which makes the process even more difficult. To make the process easier, the Product Owner decides to employ AI to (1) help with backlog refinement, and (2) to estimate the work needed for each user story based on the user stories the team has completed in the past.

AI could also be utilized to boost the efficiency of regularly held agile ceremonies like the daily meetings. An appropriate model could be used to assess the progress of each team member to keep everyone up-to-date and to identify potential blockers. The result of the assessment could be delivered to a group team chat before the daily meeting as an automatic status update. This can potentially remove a significant amount routine work from daily meetings, allowing the team to focus on more important topics of the day.

6 Interviews

As mentioned in Section 2.2.2, a semi-structured interview technique was chosen for this thesis. The interviews were conducted individually, allowing in-depth discussion about the topic to occur. The goal of the interviews was to gather not only the opinions but also the experiences of Company X's software developers about utilizing AI throughout the software development lifecycle, with a particular interest in the scenarios presented in Chapter 5. The interviews were an opportunity to assess and validate the scenarios.

Table 6.1: The interviewees

	Role	Experience (years)	Work description	Division
I1	Senior Mobile Developer	7	Mobile application development	Retail trade
I2	Software Engineer	7	Full-stack development (frontend and backend), software design, improving agile development in other teams	Retail trade
I3	Full-stack Developer	8	Web development (frontend and backend)	Car trade
I4	Senior Full-stack Developer	5	E-commerce site development, project management, and software design	Building and technical trade
I5	Senior Full-stack Developer	10	Web development (frontend and backend)	Building and technical trade

The group of interviewees are listed in Table 6.1, including their role in Company X, years of experience in software development, the interviewees' description about their work, as well as the target division of their work in Company X. In the following sections, the interviewees are denoted using the identifiers found in the table's first column, while the researcher is referred to with the letter R.

All interviews took place in July of 2024. The interviews were conducted over a video conference platform utilizing both the audio from a microphone and the video feed from a webcam, with only the audio being recorded. The total of audio that was recorded amounted to approximately 6 hours of material. Hence, to process and analyze the interviews more efficiently, the audio from each interview was transcribed into written English.

The chosen language for interviews was either English or Finnish. Therefore, for some interviewees, the questions presented in this thesis were translated to Finnish. Additionally, some of the quotes and responses given by interviewees have been translated from Finnish to English.

6.1 Interview questions

To gather the desired data and to provide structure to the interviews, a set of 8 questions were used. The questions are outlined in Appendix A. The questions were not visible to the interviewees during the interview: they were verbally presented one at a time. Additionally, for Q4, the scenarios presented in Chapter 5 were provided to each interviewee prior to the interview.

Before going into the questions, each interviewee was provided with an introduction to the thesis topic and goals. To emphasize that the thesis is not inspecting solely the implementation phase of development, but rather the entire software development lifecycle, the phases presented in Table 3.1 were also introduced to each interviewee.

6.1.1 Current utilization of AI

RQ3 aims at understanding how Company X's software developers are currently utilizing AI to enhance software development. The first two questions were used to gain insight on just that. Q1 was used to gather knowledge about how Company X's developers are currently utilizing AI in software development, if at all. It was also an opportunity to find out what specific tools they are using. Q2 aims at finding out how interviewees feel that AI as benefited them.

6.1.2 Scenarios

To gain insight on RQ1 and RQ2, the scenarios presented in Chapter 5 were validated in the interviews with Q3. The goal of the question was to identify scenarios that are both (1) realistically implementable and (2) provide a significant enhancement to the software development lifecycle, whether it be a speed boost in development or an enhancement in the quality of the produced software or product. Additionally, because of the open nature of the question, it enabled interviewees to present their own adjustments and improvements to each scenario, or even present their own scenarios.

In addition to discussing the scenarios, interviewees were asked to rank each presented scenario with a simple score of one to five, one representing a scenario with no enhancement to the software development lifecycle, and five a scenario with a clear enhancement.

6.1.3 Obstacles

RQ4 aims at locating potential obstacles in adopting AI for software development at Company X. The purpose of Q4 is to address this topic directly: it enabled interviewees to present obstacles that they may have faced. However, Q5 was used to recognize potential obstacles that interviewees may not categorize as obstacles by identifying differences in how they utilize AI in different environments.

While it is not explicitly a goal or a research question for this thesis, the final set of questions were used to gauge how software developers at Company X are gaining and sharing knowledge on AI. Arguably, the most recent AI advancements may not be that relevant or important for the average software developer. However, it is important for a company, including its software developers, to stay up-to-date as well as support and enforce the adoption and utilization of modern technologies and tools over the long haul. Failing to do so will likely hinder the company's success as time passes and technology advances. Therefore, the inability to gain and share knowledge of AI enhancements in software development can be characterized as an obstacle.

For the aforementioned reasons, Q6 was used to identify sources of information that Company X's software developers use to gain knowledge on AI, specifically for software development purposes. Q7 and Q8 were used to both (1) recognize how, if at all, information about AI is shared in Company X, and (2) how it could be improved.

6.2 Results

In this Section we will go through the results of the interviews. Similar to the previous section, we will go through each category individually. However, we will discuss the scenarios in further depth, going through each category of the scenarios individually. Finally, the score based rankings of each scenario are outlined in Table 6.2.

Table 6.2: Scenarios ranked from 1 (no enhancement) to 5 (clear enhancement) by each interviewee with scenarios on the Y-axis and interviewees on the X-axis.

	I1	**I2**	**I3**	**I4**	**I5**	Average
S1	4	4	5	4	5	4.4
S2	2	2	4	3	2	2.6
S3	3	3	2	3	1	2.4
S4	3	1	2	2	4	2.4
S5	4	5	4	2	3	3.6
S6	5	2	3	3	5	3.6
S7	2	2	1	4	2	2.2
S8	3	1	3	3	4	2.8
S9	3	4	4	2	4	3.4
S10	3	2	1	2	5	2.6
S11	1	4	1	3	3	2.4

6.2.1 Current utilization of AI

Unsurprisingly, all interviewees utilize artificial intelligence in their workflow to some extent. While there are a lot of similarities in how they currently utilize AI in software development, there are also some clear differences.

All interviewees reported utilizing AI to generate code. The chosen tool was unanimous: GitHub Copilot. However, on occasion, some interviewees used Chat-GPT for the task as well. Some interviewees described going as far as to generate unit tests, while others would not rely on Copilot for anything other than very simple blocks of code. Additionally, interviewees reported to having used AI to summarize what a block of code does as well as summarizing error logs for debugging purposes.

For personal projects (if any), interviewees felt that they are able to use generative AI more freely than when compared to Company X. This is not surprising, as a professional setting puts the threshold of acceptable code higher as well as increases the risk of deploying AI generated code further. Additionally, interviewees are not at risk of supplying generative AI tools and services with copyrighted code or material that they do not legally own when working with personal projects.

6.2.2 Source code generation

To describe source code generation to the interviewees, S1 was used. All interviewees see generative AI to generate code as an enhancement. It was ranked the highest with an average score of 4.4, as presented in Table 6.2. Interviewees also agreed that the benefit of using AI in this way comes solely from saved time.

Interestingly, none of the interviewees reported an increase in code quality when using a tool like GitHub Copilot to generate new code from scratch. On the contrary, some interviewees described a decrease in both quality and correctness of the new code, if AI is used this way in excess. If it is not utilized carefully, the code base can become bloated, containing lots of unnecessary duplication and components that none of the project's developers are familiar with.

Interviewees recognized that AI generated code can be syntactically valid, and even appear as the correct solution, but it does not guarantee its correctness. Hence, all interviewees emphasized the importance of understanding the generated code to avoid errors. For example, I3 described an event where they relied too much on the generated code:

> **I3:**
> There was this one time when I used generative AI to generate [more complex] code. With a quick glance, I thought it looked good on the surface. However, it pretty much took me a full working day to fix the issues that it had caused.

Furthermore, I4 brought up the fact that the level of expertise of the developer matters when using a tool like GitHub Copilot:

> **I4:**
>
> If we are dealing with a junior-level software developer, who is essentially a beginner level developer, there is a risk that they will utilize AI code generation to generate code that they do not fully understand. On the other hand, a seasoned senior developer is more likely to recognize the risk of generating code, especially when its functionality is unclear.

The consensus about the benefits of an AI-based copilot tool was that all interviewees appreciated the on-handedness and easy availability as well as its ability to help reduce typing, especially when dealing with repetitive but trivial tasks and minor coding issues. I4 characterized their view of a tool like GitHub Copilot as an advanced autocomplete rather than something that should be extensively used to generate large and complex parts of the code:

> **I4:**
>
> I view generative AI tools like GitHub Copilot with a certain kind of scepticism. The style of usage, at least for me, is like a fancier autocomplete. It helps you find more elegant solutions to simple problems as well as making your typing a lot quicker.

While having such a tool so easily available and ready-to-go at all times is great, it has its own pitfalls. One problem that may arise is that developers can, over time, become very reliant on the tool. I2 described this in the following manner:

> **I2:**
>
> One thing that I have noticed is that sometimes I stop writing code to wait for [GitHub] Copilot to give me an idea as to how to continue, rather than continuing to think for myself as to what to write next. Therefore, instead of thinking what I am actually trying to accomplish, I rely too much on the tool to tell me how to solve the problem.

As mentioned previously, none of the interviewees saw an improvement in code quality when generating new code from scratch. However, I5 reported utilizing GitHub Copilot, in addition to generate tiny code snippets and small functions, to suggest improvements to the code that they have already written:

> **I5:**
> I would say that I use GitHub Copilot in a third way. What I would do first is that I start by drafting my code, building it and making sure that it works. And then, I will ask Copilot: can you improve my code? So Copilot will improve code that already works rather than making unnecessary noise and then me having to figure out which [generated] code works and which does not.

6.2.3 Software documentation generation

For software documentation generation scenarios S2 and S3, the results were far more mixed than, for example, source code generation. It seems that for Company X's case, generating documentation directly into the code as comments is not necessary. One of the reasons behind this is that they utilize statically typed languages like Java and arguably TypeScript, which effectively eliminates the need for type annotations as comments in the code.

However, the interviewees do not rule out the possibility of using code documentation generation if they were dealing with a statically typed language like, for example, plain JavaScript. In addition, I4 continues the same thought, and mentions an interesting point about managing code documentation with AI:

> **I4:**
> I definitely see this useful with something like writing JSDoc for JavaScript. However, it should always be kept in mind that some AI model has made it, so it should always be taken with a grain of salt. But if we were to maintain some kind of a code documentation solution like for example JSDoc, perhaps AI could be used to pinpoint parts of the documentation where the documentation is no longer up to date.

Another concern with documentation, generated or not, is that it has to be maintained. This can create a lot of work for developers. For this reason, the interviewees prefer to keep documentation very light. Furthermore, the interviewees prefer to simply write clean code:

> **I1:**
> In our case, we have not used AI for this. By its nature, good code is documentation for itself. Basically, we do not have much extra documentation apart from some instructions how to set things up, which is not included in the code.

An interesting point mentioned by I3 was that they see AI as very useful in generating documentation for blocks of code that are large and complex. Rather than inspecting the function manually, they utilize AI by providing it with the function and asking it to explain what the function does. While this generated documentation does not get stored anywhere, it is useful in that moment for the developer to gain an understanding of what the block of code does. I3 described this in the following way:

> **I3:**
>
> If there is an unknown block of code, it is very easy to simply highlight the block of code in the code editor and then ask GitHub Copilot to explain what it does. I think that that has been very useful.

Finally, I2 mentioned that they see a lot of potential value in utilizing AI in migrating software documentation to a single location with a standardized format:

> **I2:**
>
> We use Confluence for a lot of our software documentation needs. I think AI could be very valuable if we could define a template for the Confluence documentation, feed the old material – in whatever format – to the AI, and the AI would automatically generate and upload the documentation to Confluence in the format we want.

6.2.4 AI-driven bug triage and debugging

The interviewees' thoughts on utilizing AI for bug triage, as presented in S4, were mixed. Some interviewees felt that bug triage is not about choosing the right developer, but instead about sharing knowledge among the team:

> **I1:**
>
> In a way, it is more efficient to let someone with more knowledge to fix an issue. On the other hand, if the person with the most knowledge is always fixing some specific bugs, it can deepen the knowledge gap between the developers. Sometimes fixing a bug is a great way to learn about that specific part of the project.

I2 also had some thoughts about the need for bug triage in the first place. They highlighted the importance of establishing clear product or system ownerships within the organization, and the size of teams:

> **I2:**
>
> I believe that there should always exist some kind of a product ownership for software products and systems. With that said, developer teams should not have 20 members, but rather a smaller group that has a deep understanding of the product that they own. For this reason, having AI aid in locating that specific team that owns the product is rather trivial and not useful in my opinion.

They also later continue on the same topic by switching to a more holistic organization-wide view:

> **I2:**
>
> For a much larger organization, I think this could make sense. Maybe not for developers, but on a larger scale. For example, if a bug is reported to some very general department inside the organization, maybe there could be a filter that would automatically perform some sort of triage, create a bug fix ticket, and assign it to the correct developer inside the team that is responsible for that product.

On the more optimistic side, I5 had this to say about AI-driven bug triage:

> **I5:**
>
> If done right, I think that this could save a lot of time. It could help us prioritize the right thing.
>
> **R:**
>
> What if the AI keeps selecting certain developers for certain type of bug fixes, leaving the rest of developers out? Do you think that, if the knowledge gained from fixing certain bugs keeps accumulating to certain developers, it would become a problem?
>
> **I5:**
>
> I think that is a fixable problem. We can tune the algorithm to kind of account for that and distribute the fixes more fairly. For example, if one developer needs to gain knowledge on debugging React, we can assign them more bug fixes related to React.

Interviewees felt that S5 might be more useful than S4. Some interviewees reported to having used AI for debugging purposes to, for example, assess a stack of error logs. For example, I2 had previously used AI for this, as he was not that familiar with the specific programming language. However, they also recognized that AI may not always be able to provide a solution for very specific and complex problems that require knowledge on many different parts of the system. When asked about utilizing AI for debugging, I3 had this to say:

> **I3:**
>
> I think that bug fixing is the sum of many different aspects of the system. Essentially, you have the logs, the code, the functionality, and possibly other things to consider as well. Perhaps it could work, but I have not utilized AI for that.
>
> **R:**
>
> Would you use AI for, for example, debugging a code base that is unfamiliar to you? Or to find out what exactly is causing the error?
>
> **I3:**
>
> That is a difficult question to answer without knowing what the *[AI]* tool should be like. If you consider the debugging process: you get the imperfect log of what is happening, you use that to figure out how to reproduce the bug, when does it occur, and what is the outcome. All of this nuanced context has to be provided to the model or tool. The process is very multifaceted, and requires various different perspectives from the debugger.

6.2.5 AI-driven software testing and reviewing

To assess the potential enhancement of utilizing AI for testing and code reviewing purposes, S6 and S7 were used in the interviews. Again, interviewees were very hesitant in fully relying on AI to generate tests, especially complex ones like end-to-end tests that require knowledge on the entire system. Instead, interviewees felt that it is appropriate to generate simple unit tests that act as a starting point for more complex and nuanced handwritten tests.

> **I5:**
>
> For me, I would say that AI is useful for generating tests as a starting point. From there, you can expand your tests more. But if you fully rely on that, it is kind of dangerous. It is almost like not writing any tests at all.

I1 also had similar thoughts on the matter:

> **I1:**
>
> I think this has a lot of potential. This is something AI could really do. Of course, I am not so satisfied with what AI can do right now, but I think this has some great potential. For example, let's say you are adding a new feature. What we already have is a bunch of tests in the code base. The AI can check them out and take them as context on how to write new tests. We can generate the tests for the new feature, and then we really just need to tweak them to make them complete. I think this would really improve the quality of the code in the project. I don't think this is very difficult for AI, but I just feel that AI is not quite there yet. AI generated code is very general, and not specific to the project.

Most interviewees saw S7 as not that useful, with the exception of I4. They saw code reviews as something that should involve developers regardless of how fit they are for reviewing the code that is being committed. This was for the same reason as to why some interviewees preferred bugs to be assigned without AI: to allow knowledge to be accumulated among the team of developers somewhat evenly. I4, however, felt that AI could help in choosing the right reviewer:

> **I4:**
> I see potential in this. You could use many inputs for the AI like who has been making most code changes lately, who has the most knowledge, or who has the most work on their table right now. It depends on the input, but if done correctly, I see some potential in this.

6.2.6 AI-enhanced CI/CD

To obtain the opinions of interviewees about utilizing AI for enhancing a CI/CD pipeline, S8 was used. The interviewees did not find the automatic generation of tests as part of the CI/CD pipeline as useful. The consensus was that tests should be written prior to any deployment taking place. Additionally, interviewees expressed similar concerns as presented in Section 6.2.5. I2 suspected that having AI automatically merge tests into production code might do more harm than good:

> **I2:**
> We already have some tools at Company X that make automatic changes to code repositories. And in some cases, they have caused harm. I suspect that this would apply to an AI-powered solution as well. If you have a release planned for Monday at 7 in the morning, and your AI hallucinates some error into one of the unit tests it has generated, it can do more harm than good.

However, interviewees generally responded positively to having AI assess production system logs for a period of time after the deployment pipeline has finished. I3 suggested, that this could be an ongoing thing, not just as part of the CI/CD pipeline. However, I5 described a concrete example situation where production logs are getting errors, which are difficult for the developers to interpret, and ultimately, fix:

> **I5:**
>
> I am not sure about the unit test part, but I like the production system logs part of it. Let's say we deploy the system, and all the time there are some kind of alerts. There are a lot of errors in the logs, let's say like error 403 and so on. This goes on for like 10 minutes. No one knows what is causing this 403 error, and it is kind of not human readable. Sometimes we don't know what we are supposed to do with this kind of an error. It would be nice to have some kind of an AI translate to us: what kind of an error is it? Some service might be down, or something like that.

6.2.7 AI-driven specification composition

In general, interviewees had similar thoughts about S9 and S10. For the most part, they felt that AI would not be reliable in generating project-critical documents that are the result of the preliminary analysis and requirements engineering. However, most interviewees thought that AI could be used to generate a starting point for the documents that would contain, for example, the most obvious requirements. But ultimately, they would require the input of a human to be reliable.

Interviewees had similar thoughts about the accuracy of the information provided by the AI. The accuracy is largely dependent on the context provided to the AI, meaning that the results will be much more accurate if AI is given the correct context, as pointed out by I2:

> **I2:**
>
> If we are creating a similar project that we have created in the past, AI will be more useful. But if we are working with a completely new and different idea and concept, it will be more difficult to utilize AI.

Interviewees were also hesitant in utilizing AI to, for example, prioritize requirements. They felt that AI would not be capable of accurately, and with great consistency, to prioritize the requirements. Additionally, it would be difficult to collect the required variables like stakeholder importance, development complexity, risks, and schedule that need to be considered. I1 had this to say about the matter:

> **I1:**
> Development complexity can be evaluated based on the tasks we have achieved before. I just feel that collecting the required data is a challenge here. Things like stakeholder importance, these things cannot be easily documented in a written way. These things require social intelligence from humans, so I would say that AI can give assistance or suggestions, but not really do the job without human involvement.

6.2.8 AI-enhanced agile development

Interestingly, interviewees had wildly different opinions about utilizing AI in agile development as described in S11. Some interviewees felt that AI could not reliably, for example, prioritize tasks as it is a difficult task even for humans to perform. Additionally, some interviewees figured that having any kind of effective prioritization would require a lot of input variables. I1 had this to say about the matter:

> **I1:**
> I could use AI to improve my writing. But prioritization? That requires a lot of input and information which is not technically official, but more so coming from shareholders in meetings, or even a gut feeling how important something is for the value creation. I do not think you can get that kind of data easily. As for the estimation of tasks, I think that is also really difficult. Even for humans estimating development tasks is really difficult. You can estimate roughly, but you never know exactly how much work something will take.

I3 described the value of agile in continuous discussion and improvement within the team, rather than producing the most optimal backlog. They felt that if that is taken away and automatized with AI, it could hinder the productivity of the team:

> **I3:**
>
> I think that this is kind of a double-edged sword. In backlog refinement, the value is gained from the discussion within the team. The discussion and designing part of refinement is more important than the outcome.

However, I2 recognized significant value in utilizing AI in splitting larger tasks in the backlog into sub-tasks as well as estimating the amount of work:

> **I2:**
>
> When I think about backlog refinement, I could see it being useful in cutting up the tasks and estimating the workload. You could have a large task in the backlog, and the AI would split it into sub-tasks. If you have a larger team where no one takes charge or facilitates the discussion, it can be difficult for developers in that team to give estimates for tasks. However, if AI hallucinates some number, even if it is completely wrong, it would at least provoke some discussion in the team.

I2 also reported to having used AI for the purpose of generating tickets and estimating story points in the past, being the only interviewee having done so:

> **I2:**
>
> For the refinement part, I myself have used AI for generating story points for tasks. Tools like ChatGPT have been excellent for that. I was creating the architecture for a notification project, and I used ChatGPT to generate all of the tickets and story points for me, which was really quick. It got almost everything right, I only had to make small adjustments and tweaks to make sure I included the things I wanted, and excluded what I did not want.

6.2.9 Obstacles

The biggest obstacle in adopting AI solutions for software development mentioned by interviewees was the trust issues and skepticism about the reliability of AI-generated material. Currently, utilizing AI for software development requires human monitoring to a large extent, especially if the task is crucial and failing to complete it correctly would cause a great deal of harm. I4 highlighted the fact that because commonly used software development AI tools like GitHub Copilot are trained with material available on the internet. This implies that really great solutions to problems are few, and the models that power these AI tools are trained using material that contains partially or even completely incorrect solutions to problems:

> **I4:**
>
> For me, the skepticism towards these AI tools comes from the lack of proper training material. There aren't that many top notch solutions available for all coding problems on the internet. With that said, the question is that can these AI models perform?

Additionally, interviewees felt that one major barrier for adopting AI-based solutions, especially fully automated ones, to the software development lifecycle is the ML models' lack of ability to account for Company X's individual workspace or code base. The currently available models are, for the most part, very general. They are not specific to Company X. I5 had this to say about the topic:

> **R:**
>
> Do you think that AI would be able to solve more complex problems that require knowledge of an entire code base?
>
> **I5:**
>
> I believe not at the moment. I mean, with the current state of copilots, it is not possible because it can only read the current file you are working with. It cannot access the whole repository for example. That is clearly a problem. It can only solve problems that are open to the internet and have been solved in the past.

Another obstacle mentioned by an interviewee is the sheer size of Company X. Company X is a large, mature and dominant company that wishes to remain that way. This implies that implementing new and disruptive AI solutions throughout the company is going to be costly and time-consuming. AI is also very immature and likely to evolve rapidly over the following years. I2 described their thoughts on the matter as follows:

> **I2:**
>
> Well, Company X being a publicly traded company, it has a responsibility towards shareholders that have an equity stake in the company. The funds of the company and its shareholders need to be protected, we can't just throw money to the next big innovation without careful thought and planning.

Another issue mentioned by interviewees was data privacy and legal concerns. However, as pointed out by I4, it is not really a concern anymore, because Company X has been able to obtain licensed versions of commercialized AI products that do not use company data to train the underlying models:

> **I4:**
>
> Understandably, one thing that has caused concern, as we have seen [in media], is that some data would leak out because some commercialized ML model uses the company data as input data to train the model. For this reason, we are understandably limited to using the AI products that have been licensed, where the input data is not used for training the model.

7 Discussion

The goal of this chapter is threefold. First, we aim to interpret the results presented in Section 6.2, and explore what observations we can make based on the thesis topic while reflecting on the information we have gained from our literature review. Secondly, we assess the observations, and discuss what concrete steps Company X can take to apply the knowledge gained from this thesis. Finally, we use this chapter to address the research questions we have set for this thesis.

The impact of AI-based solutions on the entire SDLC is discussed in Section 7.1, which addresses RQ1 and RQ2. To address RQ3, the current utilization of AI at Company X, and the potential improvement of it, is discussed in Section 7.2. Furthermore, to address RQ4, obstacles and drawbacks of AI are discussed in Section 7.3, with potential solutions presented when applicable. To summarize, the purpose of this chapter is to serve as a bridge between the literature review and the interviews, offering holistic insights about the research and its results.

7.1 Impact on SDLC

Based on the research material, all phases described in Table 3.1 appear to be enhanceable with artificial intelligence to some extent, addressing RQ1. However, the level of enhancement appears to vary wildly per each phase. For example, P1, P2, P3 could potentially be enhanced to a rather small degree, while P4 and P5 could be enhanced quite a bit. The remaining phases, P6 and P7 could be enhanced moderately. Similar variation can be recognized in tasks outlined in Table 3.2, with the exception of T3, which did not come up during any of the interviews. Arguably, this could be because of two reasons: (1) no scenario was proposed for the task specifically, and (2) all interviewees were strictly software developers, not user interface designers. However, this does not imply that T3 cannot be enhanced with AI-based tools.

Next, addressing RQ2, we will discuss the level of enhancement for phases and tasks. It is clear that some phases and tasks are easier to enhance, while some are more difficult and not as apparent. Additionally, some enhancements appear to provide greater enhancements than others, making them more beneficial to each phase and task as well as the entire SDLC. To gauge the level of potential enhancement, we will consider two factors: (1) the open discussion, opinions, and examples expressed by the interviewees, and (2) the rankings given for each scenario by interviewees. While the rest of this section goes into further detail, an overview of AI enhancement in SDLC is presented in Figure 7.1.

Given the interview discussions, P4 and P5 appear to be the phases that benefit most from adoption of AI-based solutions in the SDLC. Interviewees clearly felt that AI-based tools like GitHub Copilot provide a significant enhancement to the aforementioned phases. Especially T4 and T5, when occurring within these phases, appear to gain a vast enhancement from AI utilization. Interviewees did not only discuss S1, one of the scenarios targeting these phases and tasks, in a rather positive manner, but also ranked it with the highest average score of 4.4, as depicted in Table 6.2. One reason behind this could be that it is already adopted and frequently used at Company X, and therefore it is an obvious enhancement from the interviewees' perspective. Additionally, the aforementioned phases and tasks tend to be one of, if not the most, time consuming phases and tasks of the entire SDLC.

On the other hand, P1, P2, P3, were not seen as phases that would benefit sig-

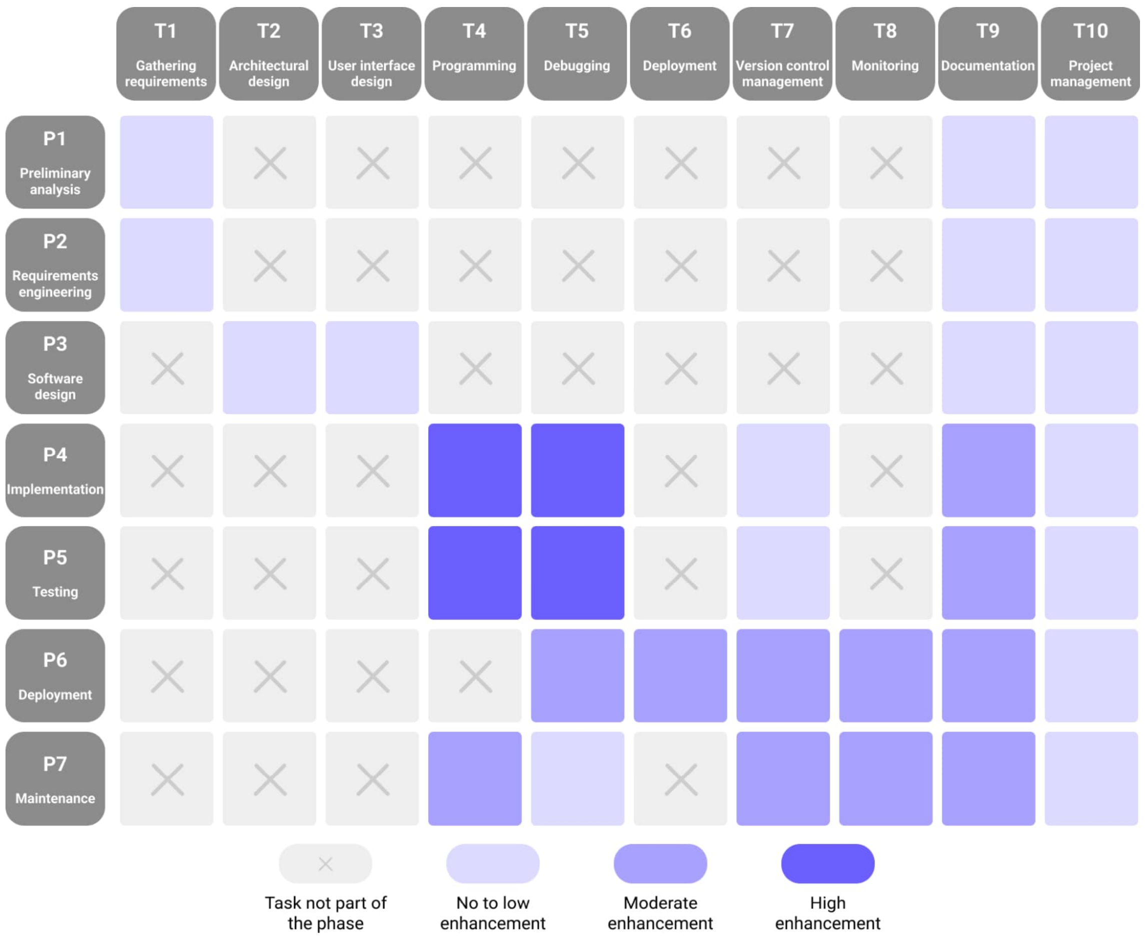

Figure 7.1: A holistic visual overview of AI enhancement in SDLC with phases 1-7 on the Y-axis and tasks 1-10 on the X-axis.

nificantly from the adoption of AI-based solutions. The same applies to T1 and T2, which generally take place within the aforementioned phases. This was largely due to the interviewees' skepticism towards the reliability of AI-based tools to account for the necessary context such as the opinions of stakeholders, potential risks, unforeseen circumstances, and architectural requirements. Rather, tools like Microsoft Copilot and ChatGPT were seen as assistive tools that could speed up those phases and tasks in a generic way, like providing summarizations of lengthy documents and writing aid. However, interviewees were open to the idea of using AI to create a starting point for some of the documentation and planning-intensive phases P1, P2, and P3. This is reflected by the rankings of the targeting scenarios S9 and S10, which received an average ranking of 3.4 and 2.6, respectively.

Reflecting on the interviews, phases P6 and P7, as well as tasks T6, T7, T8, T9, and T10 appear to benefit from AI enhancement moderately. Interestingly, in most of the interviews, the interviewees innovatively discussed realistically implementable AI-enhanced systems and workflows that have the potential to enhance the aforementioned phases and tasks, but have not been implemented yet at Company X. These were systems that were not captured by the scenarios presented in Chapter 5. These AI-powered solutions appear to be many, as different interviewees came up with very different kinds of suggestions as to what they could be. Examples of what the interviewees came up with are presented in Section 7.2. Furthermore, these phases and tasks are not as commonly enhanced in the software development field by AI-based tools or automation as, for example P4, P5, T4, and T5. This makes the potential enhancements not as apparent and even somewhat unfamiliar to the interviewees. Therefore, because of the sheer amount of different utilization cases, it is not out of the question that these phases and tasks could in fact benefit significantly from AI enhancement.

7.2 Improving current utilization of AI

To directly address RQ3, the current utilization of AI for software development at Company X is largely revolving around T4 and T5 with most of the utilization happening in P4 and P5. Based on the interviews, it appears that little to no attention is currently given to the other phases and tasks outlined in this thesis. Arguably, one of the reasons behind this could be that utilizing AI for actual programming part of SDLC is one of the most apparent and publicly discussed ways one can utilize AI for software development. However, based on the findings of this thesis, there likely exists a substantial amount of undiscovered potential in utilizing AI for the other software development phases and tasks as well.

In the case of Company X, the most significant improvements could be gained from examining areas within the SDLC where capabilities of AI are not currently being leveraged. By this we mean the examination and enhancement of both phases and tasks, but also procedures related to the SDLC that are specific to Company X. Discussion related to this very topic came up during the interviews. In addition, concrete examples of what the enhancements could be were also a topic of discussion.

One example would be building an AI-powered production monitoring system that would examine the production system logs and alert developers in certain circumstances. This system could also provide a small description of the problem, making the debugging process faster. It would also add an extra layer of defence for projects that are business-critical, and require careful and continuous monitoring.

While the interviewees' opinions were not unanimous on the matter, another example would be using AI to automatically generate a base layer of simple unit tests for new projects or features that could then be improved upon by the developers. Naturally, the blueprint for these tests would need to be strict and simple enough for AI to be capable of generating them reliably.

Furthermore, triage-related enhancements were also discussed during the interviews, and they received mixed opinions as well. However, there could be some potential in building, for example, an AI-powered pipeline where bug fix tickets are automatically created and assigned to the correct team that has product ownership of some software system or product. This could potentially decrease the time between the detection and recognition of a bug, and the fixing of it.

To handle the migration of older documentation to a newer format, one interviewee proposed a process where old documentation is converted to a new format using AI. Building on that, the process could potentially be extended to something else, like documentation generation based on the contents of a pull request, which could be automatized to be done in all future pull requests as well. For GitHub code repositories, this would mean the automatic generation of a Markdown README file.

From a project management viewpoint, agile development could potentially be enhanced using artificial intelligence. One interviewee described utilizing AI for managing and estimating the size backlog items by (1) splitting large tasks into smaller sub-tasks and (2) estimating the workload using AI, and even reported to having done so in the past. While some interviewees had contradicting opinions, such a solution could prove to be an enhancement if implemented well and used moderately.

7.3 Obstacles and drawbacks

To address RQ4, we need to consider the literature review we have done in Chapter 4 regarding artificial intelligence. Based on the review, we know that the algorithms that power AI-based tools like GitHub Copilot are simply aiming to predict what is the next most likely piece of a pattern. Given this information, we can make the assumption that the algorithms, and their ability to make correct predictions, are very highly dependent on the training material. Additionally, context is crucial for machine learning algorithms because it provides rather essential information to the model that allows it to make more accurate predictions. Without proper context, the model is more likely to find superficial patterns in data that lead to incorrect generalizations. This is one of the key obstacles of adopting AI, and it was recognized on a high level by the interviewees. One way to combat the context problem is to provide the necessary context. This could mean allowing the model to analyze, for example, a code repository prior to utilizing the model to generate new code in the context of the repository.

One drawback that came up during the interviews was the fact that over-reliance on AI to generate code can lead to code repositories that contain poor quality code.

In addition, excessively relying on AI could lead to reduced critical thinking, erosion of core programming skills, and reduced learning opportunities for developers using it over time. Especially junior-level developers are prone to these pitfalls as they are still learning to a great degree, and should not overly rely on AI to solve problems for them. This issue is also emphasized by the fact that currently developers are using AI in very different ways. To combat this nuanced issue, Company X has different options. One option is to standardize and regulate AI usage for code generation to meet certain guidelines. Another option is to educate developers on how assistive AI tools like GitHub Copilot can be used optimally to speed up the writing process while maintaining control and high quality.

The large size of Company X makes the adoption of new technologies and innovations more difficult than when compared to, for example, a startup. A small company can move fast, and quickly adopt new technologies as well as tear them down as they come and go. However, this is not the case for a large company. As pointed out by the interviewees, Company X should exercise caution when investing into new and disruptive solutions like AI. Committing to the wrong thing will likely have a long-term negative economic effect on the company. Fortunately, there are ways that Company X can utilize to work around this issue. For example, Company X can launch small internal proof of concepts that test a particular AI-based solution in some selected small component within the organization. This could be a specific product, feature, or a team. Then, once the proof of concept is completed, they can choose to let go of the idea, or extend it into another component within the organization. Another option is to consult experts that have adopted AI in a similar setting as Company X. This way Company X can learn about the adoption of AI in a similar setting without doing so themselves, and learning everything from scratch.

8 Conclusions

In this chapter, we conclude the thesis by directly addressing the research questions presented in Chapter 1. We also want to shed light on the limitations of the research and provide suggestions for further research. We do this in an effort to (1) improve transparency and credibility as well as (2) provide a potential direction for further research.

In this thesis we set out to explore the world of enhancing software development phases and tasks with AI-based solutions. To accomplish this, we first laid out the basis for the thesis by reviewing contextually important academic literature on software development as well as artificial intelligence and machine learning. Then, we innovated potential enhancement scenarios, and held semi-structured interviews with software developers working at Company X. Finally, we interpreted and discussed the results of the interviews, and drew our conclusions based on the research.

8.1 Remarks

Beginning by addressing RQ1 and RQ2, the research indicates that all phases and tasks of the SDLC can be enhanced by artificial intelligence to some degree. However, the extent of enhancement varies significantly between the phases and tasks. Phases P4 and P5, along with tasks T4 and T5, benefit the most from AI tools like GitHub Copilot, which interviewees rated highly, especially for time-intensive activities. In contrast, phases P1, P2, and P3, as well as tasks T1 and T2, were viewed as less suited to AI enhancement, largely due to the complexity of contextual factors such as stakeholder input and architectural decisions. However, AI was seen as useful for generic support like writing aid. Phases P6 and P7 and tasks T6–T10 showed moderate potential for AI enhancement, with interviewees proposing creative AI-enhanced solutions that have yet to be implemented at Company X. This indicates possible future enhancements for the taking. Overall, AI shows great promise in certain areas of the SDLC, particularly in repetitive or time-consuming tasks, but its full potential is unknown.

Next, we will address RQ3. Currently, AI is primarily used in phases P4 and P5 and tasks T4 and T5 at Company X. This means that the usage is mainly focused towards the implementation of software and tests. The remaining phases and tasks, however, appear to receive little to no attention, likely because of AI's role as a coding assistant is widely recognized and frequently in the spotlight, leaving other potential utilization scenarios in the dark. However, this research suggests substantial untapped potential for AI enhancement across the entire SDLC. Interviewees proposed several AI-driven improvements, including AI-powered production monitoring systems for enhanced monitoring and faster debugging, automated generation of simple unit tests, and an AI-based triage pipeline for managing and directing bug fixes. Additional ideas included AI-assisted documentation migration and generation as well as agile development enhancements through AI-based task estimation and backlog management.

Addressing RQ4, Company X is facing some obstacles in the adoption of AI that could be mitigated. To overcome obstacles in AI adoption, Company X has several options to choose from. First, they can mitigate the risk of low-quality AI-generated code and the erosion of developers' critical thinking due to excessive AI usage by standardizing the usage itself, and educating their developers on optimal practices. Additionally, to address challenges related to their large size, Company X can start with small internal proof-of-concept projects to test AI solutions on specific components, features, or teams before scaling up. This allows the company to test and experiment without fully committing to any specific AI solution. Another option is to consult with experts who have successfully adopted AI in a similar large-scale setting, helping Company X learn from experiences outside of the company and avoid potential missteps.

8.2 Research limitations

The research done in this thesis is not without its limitations. One clear limitation of the research is the lack of interviewees. The opinions of five developers is not sufficient enough to capture the general consensus among all software developers, perhaps not even the consensus among Company X's developers. However, it does provide some direction and insight as to what the consensus might be. Additionally, having only five individual interviewees means that the opinions of those developers are emphasized in this thesis. With that said, it is not out of the question that the selection of interviewees contains developers that have wildly different views on the topic when compared to the majority of developers, which could certainly skew the results. All interviewees also represent a rather similar, albeit common, type of software development: full-stack and mobile development. Therefore, this research could be improved by inclusion of a more diverse background of software developers. Finally, including interviewees who are not explicitly software developers, but work within the software development lifecycle, could be useful. An example of such interviewees could be people working with user interface design related tasks.

Arguably, another limitation is that the selection of interviewees is limited to Company X's employees. Software development varies from company to company, meaning that things may be done differently in other companies than they are at

Company X. This certainly has an effect on the results. However, while this can be interpreted as a limitation for RQ1 and RQ2, it is an essential factor for examining RQ3 and RQ4, which are targeting Company X specifically.

Finally, the list of scenarios presented in Chapter 5 is not exhaustive. It does not contain every possible scenario for AI enhancement in software development. However, it does capture common elements in software development that could benefit from enhancement. What the ideal list of enhancement would look like is going to vary from company to company. That list would also be likely to change as time goes on as the capabilities of AI increases.

8.3 Further research

As pointed out in Section 8.2, this research would likely benefit from additional interviewees from different types of software development. Additionally, it would be beneficial to increase the sheer amount of developers interviewed to better gauge the experiences and opinions of more developers.

For Company X specifically, this research could be extended by examining potential enhancements in software development phases and tasks that are specific to Company X. At the time of writing, the focus of the research could be in identifying practical enhancements by automating things where traditional automation, like rule-based if-then systems, are not capable of completing the task, but the solution is within reach with the aid of an AI-based tool.

However, as time goes on, AI and ML models are very likely to make significant advancements in what they are capable of solving. Future research should be adjusted to accommodate those advancements. One interesting direction to take would be to research how companies could better incorporate their own context into the utilization of AI, as one of the biggest stumbling blocks in AI adaptation and utilization is the lack of context.

References

[1] M. Egan. "AI is replacing human tasks faster than you think". Online; accessed September 22nd, 2024. (2024), [Online]. Available: `https://edition.cnn.com/2024/06/20/business/ai-jobs-workers-replacing/index.html`.

[2] G. R. Marczyk, D. DeMatteo, and D. Festinger, *Essentials of research design and methodology*. John Wiley & Sons, 2010, vol. 2.

[3] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development", in *Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009. Proceedings 10*, Springer, 2009, pp. 386–400.

[4] L. R. Vijayasarathy and C. W. Butler, "Choice of software development methodologies: Do organizational, project, and team characteristics matter?", *IEEE software*, vol. 33, no. 5, pp. 86–94, 2015.

[5] T. Rajala and H. Aaltonen, "Reasons for the failure of information technology projects in the public sector", *The Palgrave handbook of the public servant*, pp. 1075–1093, 2021.

[6] N. M. A. Munassar and A. Govardhan, "A comparison between five models of software engineering", *International Journal of Computer Science Issues (IJCSI)*, vol. 7, no. 5, p. 94, 2010.

[7] N. B. Ruparelia, “Software development lifecycle models”, *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[8] B. W. Boehm and W. J. Hansen, “Spiral development: Experience, principles, and refinements”, 2000.

[9] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope, “Rapid application development (rad): An empirical review”, *European Journal of Information Systems*, vol. 8, no. 3, pp. 211–223, 1999.

[10] G. Coleman and R. Verbruggen, “A quality software process for rapid application development”, *Software Quality Journal*, vol. 7, pp. 107–122, 1998.

[11] D. K. Rigby, S. Berez, G. Caimi, and A. Noble, “Agile innovation”, *Bain & Company Brief*, 2016.

[12] “Manifesto for Agile Software Development”. Online; accessed February 14th, 2024. (2001), [Online]. Available: `https://agilemanifesto.org/`.

[13] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis”, *arXiv preprint arXiv:1709.08439*, 2017.

[14] D. T. Llewellyn, “Financial technology, regulation, and the transformation of banking”, in *The European Money and Finance Forum (SUERF) conference: Financial Disintermediation and the Future of Banking Sector*, 2018.

[15] R. F. Bortolini, M. Nogueira Cortimiglia, A. d. M. F. Danilevicz, and A. Ghezzi, “Lean startup: A comprehensive historical review”, *Management Decision*, vol. 59, no. 8, pp. 1765–1783, 2021.

[16] E. Brechner, *Agile project management with Kanban.* Pearson Education, 2015.

[17] “The 2020 Scrum Guide”. Online; accessed February 19th, 2024. (2020), [Online]. Available: `https://scrumguides.org/scrum-guide.html`.

[18] M. Sliger, “Agile project management with scrum”, Project Management Institute, 2011.

[19] R. Jain and U. Suman, “A systematic literature review on global software development life cycle”, *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 2, pp. 1–14, 2015.

[20] C. Ebert, “The impacts of software product management”, *Journal of systems and software*, vol. 80, no. 6, pp. 850–861, 2007.

[21] B. H. Cheng and J. M. Atlee, “Research directions in requirements engineering”, *Future of Software Engineering (FOSE’07)*, pp. 285–303, 2007.

[22] P. Ralph and Y. Wand, “A proposal for a formal definition of the design concept”, in *Design Requirements Engineering: A Ten-Year Perspective: Design Requirements Workshop, Cleveland, OH, USA, June 3-6, 2007, Revised and Invited Papers*, Springer, 2009, pp. 103–136.

[23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[24] P. Freeman and D. Hart, “A science of design for software-intensive systems”, *Communications of the ACM*, vol. 47, no. 8, pp. 19–21, 2004.

[25] R. Rodriguez-Echeverria, J. L. C. Izquierdo, M. Wimmer, and J. Cabot, “Towards a language server protocol infrastructure for graphical modeling”, in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018, pp. 370–380.

[26] K. Muşlu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, “Speculative analysis of integrated development environment recommendations”, *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 669–682, 2012.

[27] E. Aghajani, C. Nagy, O. L. Vega-Márquez, *et al.*, “Software documentation issues unveiled”, in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 1199–1210.

[28] M. A. Umar, “Comprehensive study of software testing: Categories, levels, techniques, and types”, *International Journal of Advance Research, Ideas and Innovations in Technology*, vol. 5, no. 6, pp. 32–40, 2019.

[29] J. Lee, S. Kang, and D. Lee, “Survey on software testing practices”, *IET software*, vol. 6, no. 3, pp. 275–282, 2012.

[30] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation.* Pearson Education, 2010.

[31] M. V. Mäntylä and J. Vanhanen, “Software deployment activities and challenges - a case study of four software product companies”, in *2011 15th European Conference on Software Maintenance and Reengineering*, 2011, pp. 131–140. DOI: `10.1109/CSMR.2011.19`.

[32] S. Wu. “Introducing Devin, the first AI software engineer”. Online; accessed March 21st, 2024. (2024), [Online]. Available: `https://www.cognition-labs.com/introducing-devin`.

[33] Tesla, Inc. “Autopilot”. Online; accessed March 21st, 2024. (2024), [Online]. Available: `https://www.tesla.com/autopilot`.

[34] “Artificial Intelligence: Defining an Often-Misused Term”. Online; accessed December 28th, 2023. (2023), [Online]. Available: `https://whattheythink.com/articles/108945-artificial-intelligence-defining-often-misused-term/`.

[35] “AI is a Buzzword. Here Are the Real Words to Know”. Online; accessed January 8th, 2024. (2023), [Online]. Available: `https://medium.com/the-`

generator/ai-is-a-buzzword-here-are-the-real-words-to-know-47b1e0a324a7.

[36] IBM. "What is artificial intelligence (AI)?" Online; accessed March 21st, 2024. (2024), [Online]. Available: https://www.ibm.com/topics/artificial-intelligence.

[37] E. Brynjolfsson, D. Li, and L. R. Raymond, "Generative ai at work", National Bureau of Economic Research, Tech. Rep., 2023.

[38] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[39] E. F. Morales and H. J. Escalante, "A brief introduction to supervised, unsupervised, and reinforcement learning", in *Biosignal processing and classification using computational learning and intelligence*, Elsevier, 2022, pp. 111–129.

[40] T. Jiang, J. L. Gradus, and A. J. Rosellini, "Supervised machine learning: A brief primer", *Behavior therapy*, vol. 51, no. 5, pp. 675–687, 2020.

[41] IBM. "What is semi-supervised learning?" Online; accessed March 30th, 2024. (2024), [Online]. Available: https://www.ibm.com/topics/semi-supervised-learning.

[42] X. Wu, V. Kumar, J. Ross Quinlan, *et al.*, "Top 10 algorithms in data mining", *Knowledge and information systems*, vol. 14, pp. 1–37, 2008.

[43] S. H. Chen and C. A. Pollino, "Good practice in bayesian network modelling", *Environmental Modelling & Software*, vol. 37, pp. 134–145, 2012.

[44] D. Heckerman, "A tutorial on learning with bayesian networks", *Innovations in Bayesian networks: Theory and applications*, pp. 33–82, 2008.

[45] "GitHub Copilot - Your AI pair programmer". Online; accessed December 28th, 2023. (2023), [Online]. Available: https://github.com/features/copilot.

[46] E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, “Code generation using machine learning: A systematic review”, *Ieee Access*, 2022.

[47] “Docify AI – Code Comment & Documentation Tool”. Online; accessed December 28th, 2023. (2023), [Online]. Available: `https://docify.ai4code.io/`.

[48] “DocuWriter.ai - #1 AI Code documentation tools”. Online; accessed May 18th, 2024. (2024), [Online]. Available: `https://www.docuwriter.ai/`.

[49] N. K. Nagwani and J. S. Suri, “An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review”, *International Journal of Information Management Data Insights*, vol. 3, no. 1, p. 100 153, 2023.

Appendix A Interview questions

Current utilization of AI

Q1: Do you currently utilize AI in software development? If so, how? Any specific tools you are currently using?
Q2: What kind of benefits does AI provide for you?

Scenarios

Q3: Given these scenarios, how useful and implementable do you find each scenario?
List of scenarios provided to each interviewee prior to the interview.

Obstacles

Q4: Are there any obstacles that you have faced at Company X in adopting AI, and if so, what kind of obstacles?
Q5: When using AI in software development, do you use it differently in Company X's projects than you have in your personal projects or past projects with other companies?
Q6: How have you gained knowledge about AI based tools?
Q7: Are you currently sharing information with other Company X developers, and if so, how?
Q8: How could sharing information among developers at Company X about artificial

intelligence be improved?

Appendix B Notice of AI usage in the thesis

Artificial intelligence has been utilized in the process of writing this thesis. However, AI has been utilized only as an assistive writing tool, not as a means of generating content for the thesis. The usage has been strictly limited to generating ideas for how to phrase things, improving grammar and the content already written by the author, and generally refining the expression of ideas as a whole. The core arguments, analysis, and research presented in this thesis are authored by the researcher, with AI serving only as an assistive tool for writing.