Sini Vänskä

CONTINUOUS DEVELOPMENT OF AI: ADOPTION CHALLENGES

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2021

ABSTRACT

Vänskä, Sini
Continuous development of AI: Adoption challenges
Jyväskylä: University of Jyväskylä, 2021, 97 pp.
Information Systems, Master's Thesis
Supervisor: Abrahamsson, Pekka

The topic of this master's thesis is the challenges related to the development of artificial intelligence when development takes place using the method of continuous software engineering. Technologies involving artificial intelligence are widely used in various industries and are expected to grow in importance in the future. However, the development of artificial intelligence differs considerably from traditional software and system development, as the purpose of the current program is to create an artificial intelligence system that predicts the future. The development of artificial intelligence is a step-by-step process in which the concept of an artificial intelligence system created is taught to make predictions about test data, which is implemented in the existing system. The business environment is rapidly changing, as innovations, technologies, and practices can revolutionize industries and processes. The frameworks used to develop artificial intelligence have not undergone the same evolution as traditional software and systems development, which have evolved from so-called heavy development models to agile development models. Continuous software engineering is the latest agile method in software development that aims to make the product lifecycle one continuous deployment cycle. The purpose of this dissertation is to specify the challenges that the use of continuous software engineering in the development of artificial intelligence may pose. The study was conducted as an empirical qualitative interview in which participants worked on artificial intelligence application development projects. The study results show that the introduction of continuous improvement is associated with the challenges posed by the nature of artificial intelligence and the communication of developers.

Keywords: artificial intelligence, continuous software engineering, agile, agile development

TIIVISTELMÄ

Vänskä, Sini
Tekoälyn kehittäminen jatkuvalla tavalla: käyttöönoton haasteet
Jyväskylä: Jyväskylän yliopisto, 2021, 97 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Abrahamsson, Pekka

Tämän pro gradu tutkielman aiheena on tekoälyn kehittämiseen liittyvät haasteet, kun kehittäminen tapahtuu jatkuvan kehittämisen menetelmää käyttäen. Tekoälyä sisältäviä teknologioita käytetään laajasti eri toimialojen prosesseissa, ja tulevaisuudessa sen merkityksen oletetaan kasvavan. Tekoäly kehittäminen eroaa kuitenkin huomattavasti perinteisestä ohjelmisto- ja järjestelmäkehityksestä, sillä nykyhetkessä toimivan ohjelman sijaan tarkoituksena on luoda tulevaisuutta ennustava tekoälyjärjestelmä. Tekoälyn kehittäminen on vaiheittainen prosessi, joissa luotu tekoälyjärjestelmän konsepti opetetaan tekemään ennustuksia testidatasta, jonka jälkeen se implementoidaan varsinaiseen todelliseen järjestelmään. Nykyinen liiketoimintaympäristö on nopeasti muuttuva, sillä uuden innovaation, teknologiat ja toimintatavat voivat mullistaa toimialoja ja prosesseja. Tekoälyn kehittämiseen käytetyt viitekehykset eivät ole käyneet läpi samanlaista evoluutiota kuin perinteisen ohjelmisto- ja järjestelmäkehityksen vastaavat, jotka ovat kehittyneet niin sanotuista raskaista kehittämismalleista ketteriin kehittämismalleihin. Jatkuva kehittäminen on ohjelmistokehittämisen uusimpia ketteriä menetelmiä, joka pyrkii tekemään tuotteen elinkaaresta yhden jatkuvan käyttöönoton syklin. Tämän tutkielman tarkoitus on eritellä haasteita, joita jatkuvan kehittämisen käyttö tekoälyn kehittämisessä voi aiheuttaa. Tutkimus suoritettiin empiirisenä laadullisena haastatteluna, jonka osallistujat työskentelivät tekoälysovellusten kehittämisprojekteissa. Tutkimuksen tulokset osoittavat, että jatkuvan kehittämisen käyttöönottoon liittyy erityisesti tekoälyn olemuksen ja kehittäjien kommunikoinnin aiheuttamia haasteita.

Asiasanat: tekoäly, jatkuva kehittäminen, agile, ketterä kehittäminen

FIGURES

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

When a human child is born, they will slowly learn how to touch their toes, how eating dinner eases the feeling of hunger, and how pressing a light switch turns on the room's lights. We do not always even always consider the smallest of our daily tasks to be intelligent. Many of us "automatically" start making coffee after waking up and do not need to think about our actions, while pouring water into the coffee machine and measuring the ingredients. However, completing the preparation of morning coffee or learning a new task is a complex process in the human mind that might seem impossible to replicate with a lifeless object. Yet this idea that inanimate objects doing human tasks have fascinated people for many centuries. However, it has been only a few decades that we have been able to create the first inventions that can complete our actions that required the involvement of the conscious human mind before.

Computer systems have a significant role in our everyday life, both in work and in leisure. For example, we use a smartphone for calling, checking email, as a calendar, and for watching streaming services, to name a few. Before the product can be used in our devices, each application and software has gone through a complex, multi-phased development process, where an idea evolves and develops into a usable product. This is by no means an easy process to complete: The software products are more and more complex and distributed globally. The users, both people, and organizations need customized products that fulfill their individual needs, and the developers need to adapt to the rapidly changing minds of their customers. On top of this, the competition between the software producers is tough. One's development processes need to be effective and efficient to keep up with the competitors, or else, they might lose the race. System development has changed from a static development process with a clear goal to a vaguer group of tasks that is open to changes.

Continuous software engineering is a recently evolved development practice, where the development happens in a stream of actions, that combines the development with business strategy and planning, and operation (Fitzgerald & Stol, 2017). This research aims to study if it is possible to combine continuous practices with the development of artificial intelligence. Artificial intelligence is

developed in three separate ways. The most common is supervised learning where the training data and the "right answer" are accessible. In unsupervised learning, the systems learn by trying to find the common structure in the data on its own. The third, so-called reinforcement learning means that the system evolves by learning in a sequence that leads it to a given goal. (Mikkonen, et al., & Männistö, 2021). Still, the process requires a lot of testing and data sets created for the training, and still, the product can fail to fulfill its requirements, and the resource estimation is difficult (Srinivasan & Fisher, 1995).

Artificial intelligence has become an integrated part of our daily lives, although we are not always aware of it. It has a potential to free us from a variety of routine tasks, and thus enable more automatic operation and change the industry processes. However, the modern business environment requires the ability to adapt quickly to changes, and this can be difficult with structured development processes that are used to train systems. Some ideas of continuous software engineering are adapted in the development of artificial intelligence, but there is little research done on this area. This research studies the challenges occurring when adopting continuous software engineering methodology in the development of artificial intelligence.

The following chapters present the literature review that explains the concepts of continuous software engineering and artificial intelligence. After the literature review, the conceptual framework is formed to provide the foundation for analysis. Next, the chosen research method is presented. This is followed by examining the empirical findings and a discussion that connects the empirical findings to the theoretical background. In the final chapter, the study is concluded with an answer to the research question, a discussion on the study's limitations, and a proposition of future research opportunities.

1.1 Research questions

Continuous software engineering, and the agile models that influenced its emergence, have been extensively studied, but the concept is evolving as changing technology creates new-kind nuances and requirements for system development. The research also looks at other Agile models. This provides an understanding of the tasks different development models include, as well their opportunities and weaknesses. The goal is to understand what continuous means. Later this is incorporated into the artificial intelligence context. The goal of this study is to understand the challenges of adopting continuous software engineering method in the development of AI. This forms the research question:

- What are the challenges associated with the continuous software engineering of artificial intelligence?

The answer this question, an empirical research is conducted later in this study. To provide the scientific background for the research, two additional questions have been used. The first additional research question of the study is:

- What is agile and continuous system development?

The research question is answered by reviewing the scientific literature and research articles on the topic, considering the generalizability of the studies examined. The answer to the research question seeks to emphasize the introduction of selected design models and the related challenges.

As mentioned earlier, the purpose of the study is to investigate the applicability of continuous improvement in the development of artificial intelligence. The second additional research question is:

- What is artificial intelligence and how it is developed?

The research reviews concept of artificial intelligence, the development methods, and some of its practical implications. Ethical issues, as well as theoretical areas of artificial intelligence are excluded, due to the scope of the topic. Understanding the continuous engineering of artificial intelligence is especially important for those working with the development of artificial intelligence, machine learning, automation, or other similar technologies. The importance of automation has been growing in recent years, and more and more people are working with systems that utilize it. If the processes related to development are understood, making changes is flexible if the business environment requires it.

Literature part of the research is carried out as a systematic literature review. Literature sources are selected according to their relevance, expertise, reliability, and freshness. To assess the reliability of the sources, the aim is to check through the publication forum. Sources are searched for in the ACM Digital Library, AIS Electronic Library, Google Scholar, IEE Explore, and JykDok electronic data collections in data processing and information systems science. The search terms used are: “continuous software engineering”, “Agile principles”, “Artificial Intelligence”, and “continuous software engineering of artificial intelligence”.

1.2 Scope of the research

The literature review addresses various systems development methodologies, with a particular focus on continuous software engineering as well as agile models. The information obtained from selected methodologies are linked to the development of artificial intelligence. In addition, the literature review does not take a perspective on what kind of artificial intelligence is developed or what kind of industry it is applied to. Current approaches to the development of

artificial intelligence will be addressed at a basic level to understand the general challenges of applying continuous and agile development models. Therefore, this provides more opportunities for the application of the obtained information, as well as for further research. Little research has been done on the topic now, so there is a need for further research in the future.

The research examines applying continuous software engineering in the AI development, and the challenges occurring with the adoption. The research model created focuses on the application of the continuous practices, using essentializing toolbox in addition for the integration of the practices into the development of artificial intelligence. The research does not consider on ethical issues related to the development of artificial intelligence. The effects of the implementation of artificial intelligence, on people's work motivation or through organizational structures is not examined. As noted, the topic of the study has received little research. Continuous software engineering across organizational units is relatively new, although ideas related to continuous activities, such as continuous integration or continuous planning, have gained some attention. Due this, the literature review also seeks to highlight the ideas presented in other agile models, to create a holistic picture of the phenomenon.

2 SYSTEM DEVELOPMENT PROCESS

First chapter of the literature review goes through the different system development methodologies to better understand the development of digital products. Information systems refer to "a mechanism used for storing and retrieving an organized body of knowledge" (IEEE std 610 1991, 106). *Information system development* is a process in which the goal is to produce a technology that fulfills the user's needs. Furthermore, according to Welken (1983), information systems development is a change process taken focused object systems in an environmental setting by development to achieve or maintain objectives. The software is one element of an information system. Multiple development phases occur to produce a usable software product, as the idea has evolved to software on our computer or other devices. This chapter explores the information system development process and different methodologies used to achieve different goals attached to the development.

2.1 System development phases

Information systems development is not just coding software features or programming with Java or Python. It is complex progress that starts from the idea and usually ends with to finished product. Lyytinen (1987) describes information systems development as an organized collection of concepts, beliefs, values, and normative principles supported by material resources. System development and software development are sometimes used as a synonym, and some development practices do limit in which context its activities are used. Shortly, a system is a group of objects that interact, and software is technology in which a computer executes numeric patterns. Software engineering combines engineering techniques with software development practices and moves from the overall concepts to the more scientific approach. Software project means the development process of a software product for the customer. Each project has its specific goals, and the team carries them out. Even if each software project is unique, according

to Cotterell and Hughes (1995), software projects usually consist of the following phases:

- project evaluation
- planning
- requirements analysis
- specification
- design
- coding
- verification and validation
- implementation
- maintenance and support

The way the development process execution may vary, but most software projects include the phases listed above. However, some methods highlight some steps more, but others will have less attention. Pressman (1994) states that software engineering methods enclose different tasks. These include project planning and estimation, system and software requirements analysis, data structure design, program architecture and algorithm procedure, coding, testing, maintenance. Even if the executed activities may vary, the purpose is to achieve the unique objective within time, cost, and performance metrics.

There are many different approaches when it comes to systems development. Therefore, it can be challenging to define a unified journey for a development process. The system development life cycle describes the development, implementation, and retiring information systems through a multistep process in several phases: planning & selection, systems analysis, systems design, and systems implementation & operation (Valacich, George & Hoffer, 2004). After product implementation, the life cycle may end but can also be revisited several times. At some point, the life cycle end to product disposal. Figure 1 presents the typical system development life cycle, described by Valacich, George, and Hoffer (2004).

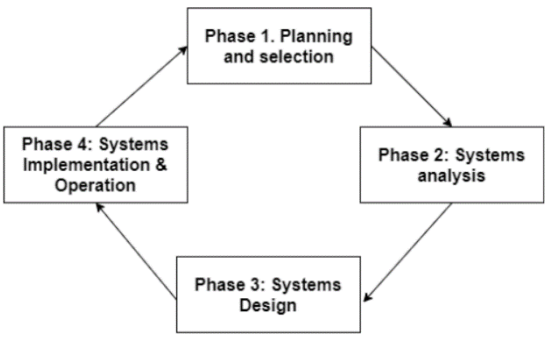

FIGURE 1 System development life cycle (Valacich, George & Hoffer, 2004)

2.2 Heavyweight and lightweight methodologies

The first information systems were created in the 1950s, and as the role of technology has grown over the years, so needs different information systems. Methodologies help to find the right ways to develop products that are as functional as possible. However, the business and customer needs and requirements of the systems have changed over the years. Thus, development practices have also changed and evolved, as they do nowadays as well. Therefore, the concept of methodology is defined in the information systems context in a mixed way. The word methodology means the science of method, a treatise, or dissertation (Blokdijk & Blokdijik, 1987). Furthermore, this materializes in one or more methods. A method is a systemic process, technique, or mode of inquiry employed by or proper to a particular discipline or a body of skills or techniques (Blokdijk & Blokdijik, 1987). Two or more practices to form a method. A framework means adapting a previous memory structure, a frame, to the new situation (Minksy, 2019).

So-called heavyweight methodologies are considered traditional for information system development (Akbar et al., 2018). These methodologies date back to the late 1960s. The models fulfilled the early development needs and requirements at the dawn of the development process. Furthermore, they served as a base for the latest methodologies and models that have evolved over the years (Petersen, Wohlin & Baca, 2009). Activities specified by the models include planning, analysis, design, and programming (implementation) (Fitzgerald & Stool, 2017). The heavyweight methodologies present a planned process in which sequential series of outlining requirements are deployed. The orderly process is predictable and, in that sense, effective (Akbar et al., 2018). An example of heavyweight methodologies is a waterfall process model, developed in the 1970s. The model is presented in Figure 2. below.

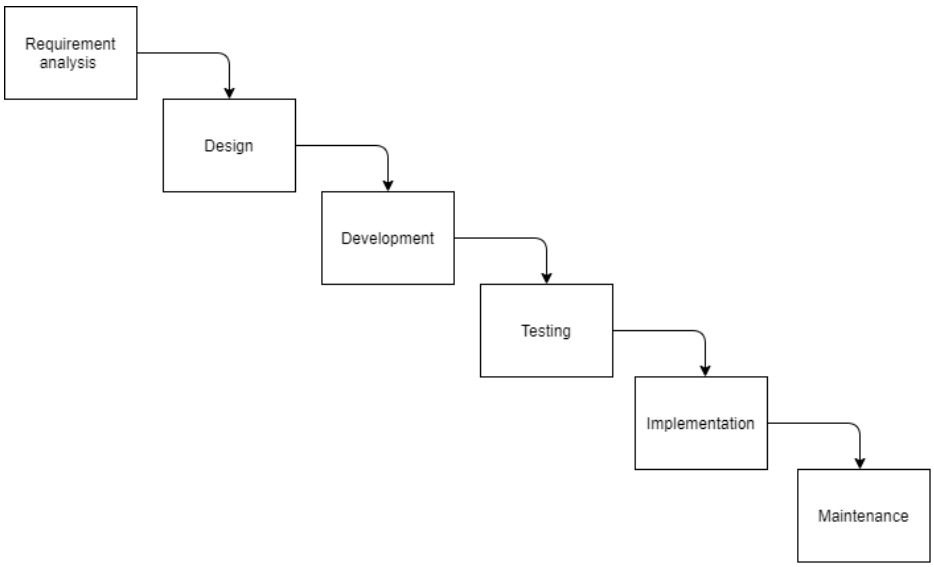

FIGURE 2 Waterfall process model life cycle (Balaji & Murugaiyan, 2012)

The heavyweight methodologies define a connected development sequence that details the plan for the development process. In addition, their characteristics are document orientation, process orientation, and tool orientation (Akbar et al., 2018). The heavyweight methodologies rely on large teams with a command-and-control culture that uniforms the organization. Due to the linear nature, the heavyweight methodologies predict the process and have a goal set in advance. As a result, the project becomes profitable when implemented at the end stages of the project sequence (Akbar et al., 2018). Thus, profit forecasting helps to plan project actions and predictability of return. Overall, heavyweight methodologies are most successful when the project environment is unstable, or the development project is large and complex (Akbar et al., 2018).

Even if heavy methodologies have many advantages, they do not exist without challenges. For example, customers do not necessarily have a clear product in mind when the development project starts, or there can be changes in the customer business environment. Also, in the field of technologies, groundbreaking innovations can emerge on short notice. Nevertheless, the changing requirements of the development project require rework and re-testing (Petersen, Wohlin & Baca, 2009). In this regard, heavyweight models are often slow and inflexible (Akbar et al., 2018), therefore, not suitable for the change. Thus, this might affect the quality and the budget of the work in process and cause delays. Furthermore, because the number of development cycles is limited, the change of the requirements is discouraged with heavyweight models. Due to this, the product might not correspond with the changed needs of the customer.

Due to the attributes such as rigidness, heavy documentation, and comprehensive upfront planning, heavyweight methodologies have become less attractive to software developers. New models, so-called lightweight methodologies, approach development as an iterative and incremental process. These ideas are also known as agile methodologies or iterative approaches. Agile software development methodologies rose after the publication of The Agile Manifesto in 2001 (Beck et al., 2001). The Agile Manifesto presented a new, customer-oriented, and effective way for development. Agile offers a dynamic approach and more adaptable ways, as building the system is done in small intervals. Those allow accommodating new business needs as they surface. The Agile Manifesto defines four center values for agile development, and Beck et al. (2001) describe them as:

- Individuals and interactions over processes and tools
- Working product over comprehensive documentation
- Customer collaboration over contract negation
- Responding to change over following the plan.

Four center values create a base for many lightweight models, and they shift the focus from planning, heavy documentation, and schedules to solution making. Lightweight models adapt to changes with less struggle and customer needs because they do not document the process or stiff protocols. Therefore, software development becomes more people-oriented, has small planning phases, lessens

the need for documentation, and makes accepting changes easier (Akbar et al., 2018). Akbar et al. (2018) have defined lightweight methodologies:

- People-orientation: Favor people or customer over process and technologies.
- Adaptive: Allow changes to requirements and the status of the project.
- Conformance to actual: Treat conformance as the actual value.
- Balancing flexibility and planning: Plan steps all to way for reaching the goal and therefore, planned period of the future may vary.
- Empirical process: Use empirical ways in the development.
- Decentralized approach: project-related decision-making takes place in teams, and management is not actively overseeing the development process.
- Simplicity: Simple tasks are easier to produce and change
- Collaboration: The developers are described as being agile, knowledgeable, collocated, and collaborative, and work with a goodwill.
- Small self-organized teams: Project aspects are communicated to the team and the team choses the best was to achieve their mutual goals.

Agile methodologies have an approach that favors people, and the success depends on the agile team members, and one success factors are developers, stakeholders, and end-users (Abrahamsson et al., 2001). The development process is more empirical and welcomes new, rising ideas and changes. Allowing changes on the requirements and status of the project helps to handle them effectively and efficiently. Also, agile methodologies do not plan all the steps for reaching the goal, and the approach is empirical. Therefore, the development can quickly adapt to emerging risks or opportunities (Abrahamsson et al., 2001). Compared to heavyweight ones with structured cycles and each phase has responsible teams, agile methodologies focus more on the collaborative approach and highlight the importance of communication (Balaji & Murugaiyan, 2012).

Even if agile methodologies focus on developing a user-friendly, understandable system and rapidly provide requirements in a changing environment, methodologies are evolving and changing. There is no clear definition for the concept of "agile" (Abrahamsson, Salo, Ronkkainen & Warsta, 2001). Moreover, it has been said that agile is not a process at all, but rather "a chaotic perspective, collaborative values and principles, and barely sufficient methodology" (Highsmith & Cockburn, 2001). However, one of the most accepted definitions of Agile is the so-called Agile Manifesto, which presents twelve supporting principles that guide the nature of all agile methodologies (Beck et al., 2001):

1. Customer satisfaction through early and continuous software delivery
2. Changing requirements are welcome.
3. Frequent delivery of working software

4. Collaboration of business and development teams
5. Motivated and trusted team works better and more efficiently.
6. Information should be shared face-to-face.
7. Working software is the main measure of progress.
8. Agile processes support sustainable development.
9. Attention to technical excellence and design enhances agility.
10. Simplicity, developing just enough for this stage.
11. Self-organizing teams emerge best architectures and designs.
12. Regular reflections on how to be more effective and process improvements.

Agile development happens in an evolutionary delivery, where the project planning is done during the process. Methodologies with this mindset apply an iterative development approach. In practice, product development occurs in a sequence in which the overall lifecycle is composed of several iterations. The iterative process is presented in Figure 3. Each iteration is a self-contained, small-scale project that includes requirements analysis, design, programming, and testing (Larman, 2004). Agile development embraces change, but not chaos, so after the requirements of the iteration on hand are chosen, they will not change before the next iteration. Between the iterations, feedback is given, which leads to refinement and adaptation of the process. Thus, the developed system grows increasingly. As mentioned before, agile lacks a precise definition, and agile methodologies are evolving due to new challenges and requirements. It is impossible to clearly define specific agile methods due to the number of variations. Therefore, managing agile projects may feel vague and full of uncertainty. On the other hand, this is an opportunity to adapt to the project needs.

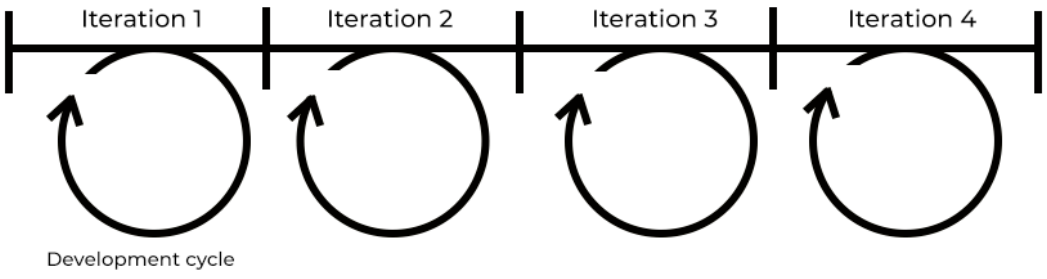

FIGURE 3 Simplifieid modeling of iteration cycles (Larman, 2004).

When reviewing literature about agile methodologies, the topic of lean development is involved. Lean thinking was born in Japan after the second world war when there was a need for more effective work methods (Ohno, 1988). The basic idea of lean thinking is to produce as much as possible with the least effort. The improvement in efficiency is achieved with five principles: defining

value, mapping the value stream, creating flow, using a pull system, and pursuing perfection (Womack et al., 1990). Lean and agile development have many similarities, such as tendencies to fast processes and customer prioritization. Lean encourages continuous improvement and a people-centric approach and creating a better workflow. Even if lean thinking has many similarities with agile development, lean focuses on effective workflow, whereas agile aims to deliver working software as fast as possible. However, even if the fundamental goals of lean and agile development differ, some agile methodologies borrow some ideas from lean thinking. The following chapters present selected agile development methodologies their core principles, defines their lifecycles, and explores the opportunities and challenges they have.

2.3 Extreme Programming

When a new software development project starts, the first task is to evaluate what the project is all about. The customer tries to communicate what they want from the new system, and the development team will analyze the information, and together, they set the requirements. However, the customer rarely has a clear understanding of their wants and needs at the start of the project. They can change their minds, or there could be changes in their needs, which causes vital problems in setting requirements and developing according to them in a long development process. Extreme Programming (XP) was invented due to possible changes that could cause issues or even a failure when using traditional development methods in a project (Beck, 1999). XP was created by Kent Beck, Ron Jeffries, and Ward Cunningham in the 1990s' (Paulk, 2001). They noticed that planning, analyzing, and designing for the far future caused the real-life development project to face difficulties in adaptability. Thus, this leads to the use of added resources and high costs of changes. XP is one of the earliest agile practices that gained a fair share of popularity. Rather than planning, analyzing, and designing in a sequence, XP uses the reduction in the cost of changing software to do activities in small portions at a time, blending development cycle activities throughout software development (Beck, 1999).

According to Beck (1999), XP approaches development from the iterative perspective, and development cycles are short and blend activities. At the start, the customer and programmers plan together the scope and time of the project releases. Minor releases are often released before the release of the whole system. The customer and the developers understand the process and functions on hand, and the design is simple. Therefore, the activities are easy to communicate, not only to the customer but also within the team. The customer's role is active, and they are engaged in the project. Furthermore, simplicity and communication make collective ownership possible, meaning project improvement is not dependent on place or time. The new release units get tested often, and further

development requirements are made based on the test result and the business costs of the change. New code is implemented into the system in a short time and tested. The customer is involved in the process the entire time and helps to write functional tests for iteration.

XP is founded on four core values that create the mindset for the development process: *communication, simplicity, feedback, and courage.* According to Fojtik (2011), XP considers active *communication* fundamental for problem detection. This happens between the team members and the customer, as they are full-time team members. Open communications help to solve development problems and ensures that all the participants are informed of future tasks. XP offers a practice known as pair programming to help with collaboration and problem solving (Muller and Tichy, 2001). *Simplicity* means that the program development is as straightforward as possible, focusing on the iteration on hand and planning the functionalities that are not currently imported into the future. Therefore, less time and energy are required for changes as the unnecessary functions can be detached. *Feedback* helps the decision-making and developing the correct software. Feedback is encouraged in different stages of development, not only after the implementation phase. *Courage* aims to remove the error and value the correct decisions at all costs. Therefore, even if a significant function or a part of the code is removed or re-done, this is not considered a failure but a necessary change. *Respect* means that the project participants are interested in each other's work. Working without mutual interest makes the process unstable and communication less fluid. Furthermore, XP encourages an open work environment, where people work closely together, both physically and mentally (Muller and Tichy, 2001).

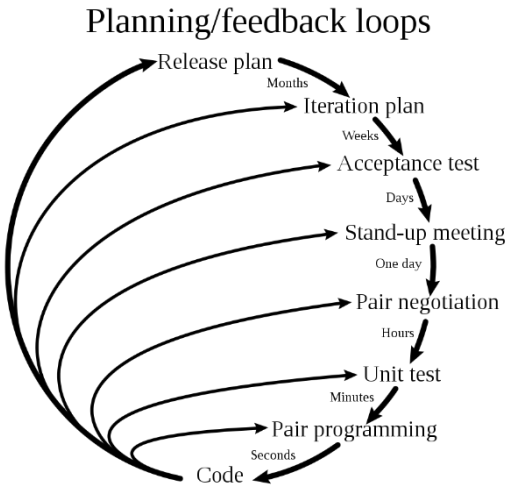

FIGURE 4 Extreme Programming planning and feedback loop (Wells, 2001)

XP presents many advantages especially comparing to traditional software development methodologies considering the changing requirements or needs.

However, XP does not come without problems that are caused by its characteristics:

1. The small iterations that able the flexible XP process can cause problems. Such a situation can happen if the development organization has some internal issues: if the organizational processes are strictly structured and well planned, teams distributed, and the organization has communication problems, collaborative and supportive environment and, flexible and iterative development approach are impossible to achieve (Mohammadi, Nikkhahan & Sohrabi, 2009).
2. XP requires constant testing; therefore, organizational resources, such as tools and skilled workers, and collision with quality control systems need to be well developed. In addition, lack of documentation or theory guidance can lead to uneven product and lack of direction in development. Therefore, the development organization needs well-developed technologies and skills that can be collaborative and supportive of development processes (Mohammadi, Nikkhahan & Sohrabi, 2009).
3. Clear communication between the customer and the organization is vital. The participants might lack a common understanding of the business environment or the complexity of the project. For example, this might occur if the customer lacks technical knowledge or does not share their business insights openly with the developers. Furthermore, a lack of standards regarding coding enhanced metaphors and practices might cause an unclear knowledge base, limit the experience, and cause issues (Mohammadi, Nikkhahan & Sohrabi, 2009).

XP offers practices for an iterative software development approach that pieces the large project into the more tangible pieces. Moreover, XP combines project tasks and activities and develops them actively based on the testing and received feedback. However, even if XP fits the changing business environment more than stiff heavyweight models, XP is still not perfect. If the organizational culture is not welcoming for the changes or processes are well defined and structured, XP practices are challenging to apply and utilize. Furthermore, XP requires a supportive and collaborative environment, and without these, communication between teams cannot be achieved, and the project loses flexibility. Moreover, the development process is more challenging to manage with large and complex projects and becomes stiffer and distributed. Also, the XP development process is people-oriented, and the customer's role is active. Without full-time availability and motivation, the customer's role becomes small. This might affect the product outcome.

.

2.4 SCRUM

Systems development takes place in a complicated environment: Complex and advanced, often unreliable technologies are produced to solve user's problems and achieve a competitive advantage for the companies. To create the compound technological systems, the ensemble becomes multidimensional. The previous chapter presented XP methodology that aims to improve cost control, delivery of systems, and the flexibility of the overall process (Schwaber, 1997). Still, XP expects that the iterations are well-defined and linear. However, the systems development projects are becoming increasingly more complex, which makes risk handling difficult. In response, the probability of success is more negligible. However, other agile methodologies are constantly evolving to tackle the challenges that the previous models have faced.

SCRUM is one of the most used agile methodologies that aims to maximize flexibility and appropriate control. SCRUM is an iterative, incremental, and general-purpose project management framework (Kumar & Bhatia, 2012). The first and the last phases of the Scrum project, called planning and closure, are well defined, but the process between them is purposely vaguer. The development happens in sprints, which are empirical processes. SCRUM offers a more flexible iterative development approach that initially plans the context and the broader deliverable and evolves deliverables during the project based on the environment. The difference between the defined (waterfall or XP) and empirical (SCRUM) approach is that the empirical approach assumes that the analysis, design, and development processes in the sprint phase are unidentified and unpredictable (Kumar & Bhatia, 2012). The controls are external and put on each sprint phase to avoid chaos, providing flexibility.

As mentioned, the SCRUM process defines only the planning and the closure phase, leaving the development phase unclear. The final product is set during the project, as are the project costs and the completion date. SCRUM methodology differs from previously developed iterative models by being responsive to the environment throughout the processes, unlimited team flexibility, and encouraged teamwork and knowledge transfer during the project. Releases are planned by using the following variables that form the intimal plan for the project (Schwaber, 1997):

- Customer requirements: what kind of enhancements the system needs.
- Time pressure: what is the time frame required for competitive advantage.
- Competition: what the competitors are doing and how to gain advantage to them.
- Quality: what is the required quality.
- Vision: what changes are required to fulfill the system vision.
- Resources: what staff and funding are available.

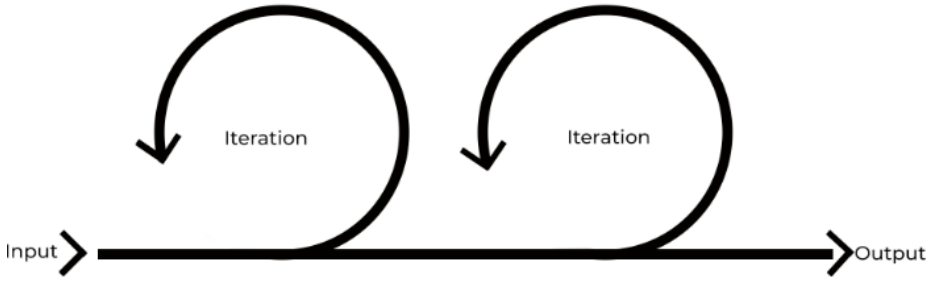

FIGURE 5 Simplified SCRUM cycle (Schwaber, 1997)

A so-called skeleton of the SCRUM describes the iterative nature of development activities that occur one after another. The output of each iteration is an insertion of the product. The number of iterations may vary, and they usually include inspection periods. During the inspection period, the individual team members inspect each other's activities and adapt if needed. The figure below presents a simplified SCRUM skeleton and iteration process, where input goes through two iterations to become the product output. The heart of the SCRUM process is iteration (Schwaber, 1997). During the iteration, the development team checks the requirements, chooses the technology, and evaluates their capabilities. Collective understanding of how to modify and build the functionalities helps with the development process's complexities. In addition, the team understands how to approach the change and required tasks. Thus, SCRUM's productivity lies ineffectiveness of the creative development process.

The SCRUM team members are committed and have more interest than other stakeholders. SCRUM teams usually contain three different types of roles: a product owner, a SCRUM master, and a team. The product owner is responsible for representing the interests in the project and the resulting system. They deliver the vision in a manner that maximizes the ROI of the project and prioritizes the requirements. The SCRUM master responds to the SCRUM process, teaching the practices, implementing SCRUM, and following the practices. The SCRUM teams are responsible for developing the functionalities. They are self-managing, self-organizing, and cross-functional, and highly motivated. Compared to traditional heavyweight development models, SCRUM projects follow a flow rather than a stiff structured and planned process. The development process is monitored throughout the sprints and redirects the development if there are new opportunities to take advantage of (Schwaber, 1999). This gives the starting point for the project, but the vision might change during the process. Changes in the process reflect changing business requirements and how the team can transform this into functionality (Schwaber, 1999).

SCRUM development processes are done in three phases: pregame phase, development, and postgame phase (Abrahamsson et al., 2002). At the start, the pregame phase consists of planning the project and the architecture. Then, the game phase includes the development sprints of the project. At last, the postgame

covers the project closure (Schwaber, 1997).). As mentioned before, only the planning and closure are clearly defined; otherwise, the process evolves. The SCRUM project starts with a vague vision of the product that becomes more precise during the process. In the pregame phase, the requirements are defined in the product backlog. The product backlog is an adaptable list of the requirements which helps to navigate the project (Schawaber, 2004). If the project enchases an existing product, the new release definition is based on the project backlog, and the critical analysis is limited. With a completely new system, the planning phase consists of conceptualization and in-depth analysis. In the pregame phase, the backlog item implementation is planned, and system architecture modification and high-level design of the product (Schwaber, 1997).

The game phase is all about development and product evolution. As mentioned, the SCRUM project consists of iterative sprints that are the backbone of the project. New release functionality is developed with constant respect to the SCRUM variables. Each sprint is an iteration of 7 to 30 days and starts with a sprint planning meeting. In this meeting, the product owner and the team collaborate about the plans on hand. They discuss content, purpose, meaning, intentions of the product backlog, and potential functionality. The goal of the sprint is shaped based on these thoughts. The daily meetings, called daily SCRUMs, help the team to stay intact. At the end of each sprint, a sprint review meeting takes place in which the team presents the development. Before the next sprint meeting, the SCRUM master holds a sprint retrospective meeting planning the next sprint. When the team feels that the variables of time, competition, requirements, cost, and quality have been reached with the new product, they declare the release "closed" (Schwaber, 1997). The last phase of the SCRUM project, the postgame phase, includes integration, system testing, and documentation of the project, and after these tasks, the product is released (Schwaber, 1995). This phase prepares the developed product for general release.

SCRUM is not a set of precise development tasks but tools to be used in development phases. It provides practices for controlled planning of a release and for managing the project variables as it progresses. This allows organizations to modify the project, therefore delivering the most appropriate release. Also, SCRUM project team members are learning during the project, and they can focus on developing innovative solutions rather than struggling with a learning curve. On the other hand, SCRUM does not become without weaknesses: one of them being that SCRUM's success of the project depends on customer involvement (Ionel, 2008). The customer must be available to test releases and plan the new functions. If the customer does not sense the product direction, it can affect the entire team and cause a sense of uncertainty and delays. Also, the customer cannot intervene in the project because the customer is not supposed to change the direction of the sprint. This can cause communication issues between the customer and the team, which can cause the customer to shift further from the project. Furthermore, the overall length of the project cannot be correctly estimated outside of the sprints. SCRUM requires an enterprise culture that embraces change, uncertainty, and complexity as part of all product development.

(Schawaber, 2007). Therefore, a long-term and detailed, predictive project plan is a waste of resources.

2.5 Scaled Agile Framework

While agile methods became more popular and gained wide acceptance in practice after the late 1990s, the problems regarding scalability and integration in large-scale development projects started to rise. The agile approach can be challenging to implement in larger development projects with many stakeholders, different components, layers, and complexity. Attributes like customer satisfaction, collaboration, commitment, decision time, organizational culture, team, social culture, and training have a significant relationship with the success of large-scale projects (Saeeda, Arif, Minhas & Humayn, 2015). When the process scales up and becomes more complicated, the success factors become harder to establish. In addition, the agile approaches are often criticized for applying primarily to small teams rather than large companies with hundreds of development teams (Stojanov, Turetken, & Trienekens, 2015). Abrahamsson et al. (2002) state that scalability is one of the significant issues with agile. Large development projects can benefit from a uniform model for assessing the progress and establishing a roadmap for the enterprise (Turetken, Stojanov, & Trienekens, 2017).

Scaled Agile Framework (SAFe) is a model created to tackle considerable agile projects experience (Turetken, Stojanov & Trienekens, 2017). SAFe implements agile practices at the enterprise level and aims to integrate existing agile models. The scaling does not happen only in "some" of the agile practices but also by introducing new practices and concepts that integrate with basic and scaled agile practices (Laanti, 2014). The purpose of this is to improve the process management, productivity, and quality problems that might occur with big agile projects. Like SCRUM, SAFe is not a group of strict practices but knowledge tools that guide the development process. SAFe is an open, free database and framework for lean-agile development and can be scaled for different organizations. The SAFe describes the best practices, roles, and artifacts of agile and lean principles but does not define any implementation strategy or method.

The foundation of the SAFe framework lies on two mindsets: The SAFe House of Lean and lean-agile principles. House of Lean is a tool created by Toyota but adopted and further developed SAFe's Lean thinking. As mentioned in chapter 2.2, lean thinking creates efficiency with the tasks: specify value by product, identify the value stream for each product, make value flow, let the customer pull value from the producer, and pursue perfection with the tasks (Womack & Jones, 2003). These ideas inspire the house of Lean in SAFe's thinking. The goal is to deliver maximum customer value in the shortest sustainable lead time possible. Thus, this single principle unifies the organization. The SAFe House of Lean is perpetrated as a roof, four pillars, and the ground floor. The foundation, as being described as a ground floor, is leadership. The top, or the roof, is the

customer value. There are four pillars between the top and the ground: respect for people and culture, flow, innovation, and relentless improvement. The SAFe House of Lean mindset aims to deliver maximum customer value in the shortest sustainable lead-time with the highest possible quality to customers. It is also said that high morale, safety, and customer delight are other goals and benefits of the SAFe House of Lean. (Leffingwell, 2018)

SAFe aims to integrate existing bodies of methodologies such as SCRUM and XP. As mentioned, SAFe not only scales up some of the agile practices but also introduces new practices and concepts as tools. Such concepts release train, business, and architecture epics, portfolio backlog, integrating primary and scaled agile practices (Turetken, Stojanov & Trienekens, 2017). When adopting agile practices on the large-scale organization, the process should start on lower levels because the higher-level agile practices are dependent on the practices introduced at the lower levels. SAFe framework's definition of three levels is introduced next.

The SAFe framework includes three levels: teams, program, and portfolio level for investments. The boundaries between these levels determine the scope and the scale between levels (Turetken, Stojanov & Trienekens, 2017). At the lowest level, the team level, the framework includes agile teams. They are responsible for defining, building, and testing a product in planned iterations and releases. The framework blends agile project management practices and technical practices and uses user stories from XP development and sprints from SCRUM. Different development teams operate on the same rhythm and with identical iteration lengths to provide integration among teams. (Turetken, Stojanov & Trienekens, 2017).

Furthermore, it has been said that SAFe promotes alignment, collaboration, and delivery (Knaster, Leffinggwell, 2018). The middle level or the program level organizes the agile teams at scale to optimize the value delivery as the primary goal. At this level, teams align with a strategic vision and roadmap for each investment concept. To plan the progress, business and architectural features are defined and prioritized in the program backlog. The agile release train is a fundamental concept on this level, which provides synchronization and integrated delivery. The agile release train produces releases or features at fixed periods. Furthermore, a system team is formed to establish an initial infrastructure and support continuous integration and end-to-end testing efforts (Turetken, Stojanov & Trienekens, 2017). The highest level of SAFe is the portfolio level, where the programs are aligned with the company's business strategy. This is done according to the value streamlines. Value streamlines are long-term series actions, including definition, development, and deployment phases used to build and deploy systems that provide a continuous flow of value to a company or customer. This level is needed for companies that require governance and management models.

SAFe is not the only agile framework used for large-scale projects. Frameworks, such as SoS (Scrum of Scrums), LeSS (Large Scale Scrum), and DAD (Disciplined Agile Delivery), are created to help with large-scale development

projects and to provide more control with the development. SAFe is used in development projects that include around 50-120 people in release trains. Also, SAFe projects have high diffusion and maturity level and are considered high to medium complexity (Vaidya, 2014). SAFe organizations are traditional enterprises that are not otherwise agile. These aspects provide required control for the complex and extensive development processes, while in many cases, other frameworks work better with low complexity or low diffusion projects.

However, companies experience several challenges when the goal is a scaled agile adaption. Challenges include resistance to change, distributed environment, quality assurance issues, integration non-agile, lack of commitment, pressure, lack of knowledge, requirements hierarchy, and progress measuring (Conboy & Carroll, 2019). Resistance the change is a common phenomenon that may happen at any level of the organization. Several researchers (Conboy & Carroll, 2019; Turetken, Stojanov & Trienekens, 2017) found that the cause of change resistance is how organizational processes become transparent. People feel observed with a high level of transparency and will not share their problems. However, at the same time, this transparency is vital to success with agile. Another problem related to the resistance to change is that people do not want to work with self-managed agile teams. This might be caused by confusion for the roles, lack of motivation, and lack of trust in the self-management by managers. In addition, the distributed environment makes the close relationship between team members and collaboration hard to maintain (Conboy & Carroll, 2019). Additionally, transparency between teams is difficult to achieve. Information sharing is easy to neglect if the teams work in a multisided environment. Therefore, transparency and ensuring that information reaches all the participants is essential when the agile processes scale.

Quality assurance issues occur due to added responsibilities of people, pressure on teams, and if the product end state is not defined (Conboy & Carroll, 2019). Also, it has been noted that the teams tend to rush with different development phases or do not use continuous integration in large development projects, thus causing technical issues. Quality problems also cause discouragement among the participants, which can affect the process even more negatively. It has been suggested that the waterfall parts of the organization should be included in the planning process, and non-agile teams are involved early on. Improvement of continuous integration and test automation systems helps, as it allows fluid integration. As mentioned, people might lack motivation due to difficulties with information sharing, and lack of commitment and teamwork is one of the problems regarding the adoption of scaled agile (Kalenda, Hyna, & Rossi, 2018). People in different teams have significant development projects and might not commit to a shared team plan. Also, people tend to start competing against each other, and teamwork hinders due to competitive atmosphere. With large projects, the pressure to fail is high: a new way of working, new responsibilities, and business pressure increase the tension within the team (Turetken, Stojanov & Trienekens, 2017). Standard plans, goals, and understanding of the process are vital that the teams work together in scaled agile projects.

With agile development methodologies, the development process is improved constantly. This might not be focused if the pressure is otherwise too high. The lack of knowledge due to the underestimated difficulty of the agile transformation, financial constraints, lack of management support, or rushed transition might cause that process development to be overlooked, and agile practices are challenging to implement. On the other hand, some agile practices will not work as they are when scaling. Scaling of requirements management cannot be avoided when scaling agile, or in other words, the Product Owner is not the single responsible person for the management. The biggest problem with the management is that the organizations did not know what to measure to get valuable data (Kalenda, Hyna, & Rossi, 2018).

2.6 DevOps

Even though the agile movement has changed the software industry, the methodologies had silos for different development tasks, separating them. Nowadays, automotive error detection and testing systems allow flexible development; the work can be done almost anytime and anywhere. Thus, the problems are solved quickly, and new development opportunities are considered. DevOps (a blend of the words Development and Operations) methodology is defined as practices intended to reduce the time between committing the change to a system and being placed into average production while ensuring high quality (Zhu, Bass & Champlin-Scharff, 2016). Therefore, when a new feature is added to the product, and the test automation has been done for the dived parts, the product is tested and released without delaying the project overall. Furthermore, DevOps includes practices that reduce and join repeated tasks with automation, integration, and deployment in development (Smeds, Nybom & Porres, 2015). Thus, the overall quality of the product is managed simultaneously as the development cycle and operation cycle go hand in hand.

DevOps is an evolution of Agile that combines development and operations into one process, therefore building a living bridge between the tasks. Thus, working and collaborating happen fluidity. DevOps extends collaboration of development towards operations, which is responsible for deploying, managing, and supporting systems' performance at the customer's site (Lwakatare, Kuvaja & Oivo, 2015). Traditional organizations divide their teams by type of work: For example, one team is responsible for the coding, and another tests the software (Hüttermann, 2012). Each department defines its goals based on its division of work in the overall project. The departments of operations succeed if the metrics of success are stable and unchanging. On the other hand, the development success metrics in agile projects strive to change. The conflict between development and operations causes the isolation of the departments and opposing goals can form between the teams. Moreover, the project development process tends to be stressful, containing manual, challenging activities and last-minute changes (Humble & Farley, 2010), and the crossing objectives might affect the overall goal

negatively. DevOps proposes a set of agile practices to the iterative delivery of software in short cycles effectively.

DevOps integrates development, delivery, and operations, to create a fluid connection of tasks that traditional development methods tend to separate (Ebert, Gallardo, Hernantes & Serrano, 2016). Furthermore, DevOps practices streamline the software delivery process, emphasizing the learning by direct feedback from production to development and improving the time from inception to delivery. The DevOps movement promotes collaboration between developers and operators: The operations department is responsible for managing modifications in production and service level, and the development department develops new features that meet the business requirements. In many development methodologies, development staff continuously push new product versions into production, while operations staff attempt to block these changes to maintain software stability. According to Leite, Rocha, Kon, Milojicic, and Meirelles (2019), conflicts between departments, long deployment times, and the need for frequent and reliable releases made the execution of agile processes inefficient. To solve this problem, developers and operators began collaborating within enterprises to close this gap. DevOps is based on agile methodologies and aims to improve the component delivery throughout the project deployment by collaboration between the departments.

There is no clear definition of the methodological background and nature of DevOps. DevOps can be considered an extension of agile development, but DevOps fulfills different needs of the stakeholders than the ones agile does (Lwakatare, Kuvaja, & Oivo, 2016). A distinctive diffracting aspect between DevOps and Agile is that DevOps provides a bridge of collaboration between development and operations, whereas agile does the same between business stakeholders and development. To integrate development and operation tasks, DevOps proposes the use of the deployment pipeline and increased automation. These provide the autonomy of publishing the code into production for developers. On the other hand, the operation department is responsible for the product and collaboration with developers to solve any problems. Therefore, the project process does not happen in isolated, self-managed silos (Lwakatare, Kuvaja, & Oivo, 2016). Thus, the DevOps project life cycles a continuous loop streaming from development to operations and back to development over again. The loop depicts how the tasks of the development department, which include planning, coding, building, and testing, lead to operating the software. First, new functionalities are brainstormed through the operation department's tasks, which are released, deployment, operating, and monitoring. Then, the planning starts again in the development department.

DevOps aims to achieve value through speed, agility, automation, and communication when it comes to business value creation. The four basic principles of DevOps are culture, automation, measurement, and sharing that establish the core of the development process (Virmani, 2015). In DevOps, the goal is to integrate development and operations to achieve a shared goal. As mentioned, DevOps gets rid of the isolated structure of silos; thus, the project and the

possible upcoming issues are shared between development and operations. Traditionally with many other methodologies, the departments work in a sequence, where the development process happens in one phase with the responsible teams. When the phase ends, the project is moved on to the next team responsible for the project tasks. DevOps shifts from this mindset as developers and operations agree upon their responsibilities and their mutual goals.

DevOps differs from other development methodologies since it focuses heavily on software release automation. The principle of automation is established through release pipelines, in which the build, testing, and deployment are automated and happen in a single path (Humble & Molesky 2011). Furthermore, the deployment pipeline enables quick software releases that can be done rapidly and used in daily releases. The pipeline practices also enable the previously mentioned culture principle: with automation, the developers have added independence and a sense of responsibility. Furthermore, with automation, the developers produce fast, pushing it instantly and achieving instant feedback from automated testing systems. This provides more freedom and makes it easier to adapt to changes in the development process.

Measuring the DevOps development process is an essential part of the project. Measuring and monitoring the process is easy with automated development systems that test, release, and deploy systems automatically. When the development processes are continuously measured, the participants can respond fast with any issues in the process or new opportunities. This also enables learning from the process and the opportunity to grow (Senapathi, Buchan, & Osman, 2018). First, however, the organization needs to decide on measured and monitored metrics. The chosen measured metrics should align with the project objectives and provide data about the current state of the DevOps adaption process. Measuring the process is not enough since giving data should also be implemented to the action. Moreover, using data in planning and decision-making enables growth and improvements.

The principle of sharing allows information to flow smoothly within the DevOps team. To improve the development and the process included, both success and challenges are shared with the teams. Thus, the whole organization shares the same knowledge and how to evolve. So, departments should work together, not in isolation, and this can only be achieved if the departments socialize with each other. With automation and measuring principles, sharing information improves, as does the response time for changing requirements. Moreover, sharing also affects the organizational culture: with the shared mindset, the staff has a clear goal and purpose in the DevOps development process. Ideas and concerns can be shared openly with other participants, which affects people's ability to collaborate and work together and the quality of the product (Senapathi, Buchan, & Osman, 2018).

As DevOps is a set of strict practices and a mindset, DevOps practices are used in the software or system development process. However, the ideas can be implemented in other areas of development as well, as the DevOps practices impact team processes, products, technologies, organizational structures, and

business practices and opportunities (Zhu, Bass & Champlin-Scharff, 2016). Therefore, using the DevOps mindset of bringing different organizational departments does not limit development and operations. As DevOps is to bridge the gap between development efforts and operation management, and the linking strategy and software development for constant assessing and improvement is called BizDev. The phenomenon complements the DevOps one of integrating more closely the software development and operations functions.

Even though DevOps tries to close the gaps that previous agile methodologies have not recognized, DevOps is not always successful. As mentioned, the foundation of DevOps is built upon the agile mindset, but as a concept, DevOps is vague and obscure. Thus, the uncertain nature of DevOps can cause challenges (Leite, Rocha, Kon & Milojicic, 2019). Moreover, to successfully adopting DevOps to the organization, the development, operational and cultural aspects need to open for the change. Finally, due to its holistic nature, the challenges and limitations of DevOps practices can happen in one or more organizational layers, and setbacks can indirectly affect other teams in addition to those that are directly affected.

DevOps can slow down the implementation or increase the risk of not fulfilling the goals. In many cases, when adopting a new way of work into the organization, change resistance may occur, and DevOps is not an exception. One of the key factors of the success of DevOps is communication (Riungu-Kalliosaari, Mäkinen, Lwakatare, Tiihonen & Männistö, 2016). The resistance to change and uncertainty can slow down the availability of skilled members and slow acceptance of the adoption of DevOps practices (Senapathi, Buchan & Osman, 2018). As mentioned, measuring the development process and DevOps adaption is one of the four principles of DevOps. If the performance metrics are not informed by the operations management or the measured data is not made available to use, which causes problems to developers. Another cause of communication problems is conflicting goals and metrics of different departments. The primary goal is to bridge the gap between development efforts and operation management. However, the differences in nature of these departments might cause frictions, especially if there is a lack of communication involved (Riungu-Kalliosaari et al., 2016).

Adopting DevOps practices requires the right cultural mindset that welcomes transparent communication and creative processes abled with automation. Problems in adapting to the new culture occur if there is stiff company culture in the organization, where merging roles, shifting responsibilities, and newly established actions for the people are seen as a threat. (Riungu-Kalliosaari et al., 2016). Senapathi, Buchan, and Osman (2018) say that one of the challenges of successfully adapting DevOps is putting the right people in the proper role, where their technical skills are used. The problem may occur both when recruiting new staff or when retraining current staff for the changing roles. Developers need to accept new tasks and responsibilities, and at the same time, the operations staff might feel threatened that developers take over their process. On the other hand, the steep learning curve for adopting new practices in daily life can cause stress and,

thus, resistance to the new practices. Behavioral changes can be challenging to manage, especially in a large, structured organization, without supportive management (Riungu-Kalliosaari et al., 2016).

There are circumstances and environments in which DevOps practices cannot be adapted. For example, some organizations have legally or contractually restricted production systems access, which means that the transparency of the process can be even impossible to achieve. In such a case, systems can be so complex that making replicants from the environment for verification and testing is difficult. Also, changing the technologies to more DevOps friendly ones, like cloud and micro-service architecture, can be incredibly difficult and slow down the organizational processes (Senapathi, Buchan & Osman, 2018), as heterogeneous software development environments are challenging to adapt. Also, the lack of standardized DevOps practices means that the organizations do not clearly understand which parts of DevOps are necessary to adapt and which practices are optional (Riungu-Kalliosaari, et al., 2016).

2.7 Continuous software engineering

Even if most lightweight methodologies can have a flexible and adaptive approach to software development, they construct silos between people with differentiating responsibilities and tasks. Some methodologies, such as previously introduced DevOps, try to break the structured nature of processes, characterized by disconnects between activities such as planning, analysis, design, and programming (O'Connor, Elger, & Clarke, 2017). A tight connection between development and execution has been the subject of improvement with the previous lightweight methodologies. Thus, errors detection is ensured, and problems are fixed as soon as possible. As a result, the quality and resilience of the system are improved and the delivery more rapid. These manifest in the increasing adoption of continuous integration practices. However, the improvement efforts have focused merely on the continuous integration of the software: The disconnected, isolated teams are still responsible for different parts of the project. Thus, the danger that the project can easily fall off track is still present. A more holistic approach that considers the development process as a continuous flow rather than a sequence of discrete activities has been suggested and introduced as the concept of continuous software engineering.

Not a single software development process is suited for every undertaking, and that all software development settings are changing nonstop (O'Connor, Elger, & Clarke, 2017). Due to constant process adaption and the required tailoring, a software process is a continuous phenomenon rather than a static process. Continuous software engineering is a rising area of research and practice that aims to bring different departments closer together, thus improving the agile processes. In the software engineering context, continuous means that the entire software life cycle is considered development (Fitzgerald & Stol, 2017). However, continuity is required in all levels of an organization and between these levels

(Suomalainen, 2015). Even if other agile practices, such as XP and DevOps, focused on continuous practices, continuous software engineering views the entire process of evolving experimentation and innovation as a continuous holistic journey. Continuous practices consist of three sub-phases: Business Strategy and Planning; Development, and Operations. In addition, various other continuous software engineering activities within these sub-phases are used in the development process (Fitzgerald & Stol, 2017). To understand these activities' holistic overlapping nature, they are next defined, and the various activities involved are explained.

Historically, business strategy and IT development have been two separate practices with competing goals (Fitzgerald & Stol, 2017). The often-occurring problem is that the development department looks for new, simplistic technological solutions without considering the complex reality of the business environment (Fitzgerald & Stol, 2017). As technological solutions have become a more critical part of the organization, developers have expressed a desire to get involved in the ongoing strategic business decision-making. The need to connect business management and the software development function has been identified. The close and continuous linkage between business and software development functions, also known as BizDev (blended from the words Business and Development), is a phenomenon that complements the DevOps, as the more business-oriented approach is needed. In continuous software engineering methodology, Business Strategy and Planning sub-phase consists of two continuous phases: continuous planning and continuous budgeting (Fitzgerald & Stol, 2017). Continuous planning means that plans are dynamic and open-ended, and they evolve in response to changes in the business environment. The process requires tight integration between planning and execution. Continuous budgeting tackles the problem that budgeting is traditionally an annual event. Due to traditional budgeting, the organization's investments, revenue, and expenses are prepared for the year to come. The Beyond Budgeting model proposes that budgeting becomes a continuous activity to facilitate changes during the year (Hope & Fraser, 2003). Therefore, budgeting is more flexible and does not rely on outdated figures, and thus, resources allocate timely. Performance measures based on the rolling forecasts will embrace the balanced scorecard linked to organizational strategy (Fitzgerald & Stol, 2017). Greater involvement of business-related stakeholders earlier in the development cycle and the setting of cooperative targets helps to prepare business activities simultaneously as the software development activities.

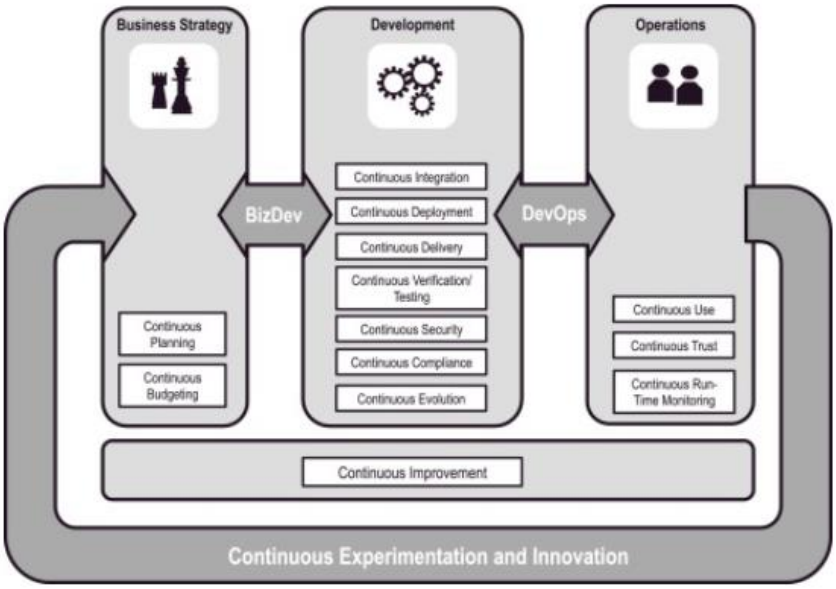

FIGURE 6 Continuous software engineering pipeline (Fitzgerald & Stol, 2017)

The more traditional software development approaches consist of activities such as analysis, design, coding, and testing. The continuous approach recognizes activities known as continuous integration, continuous delivery; continuous deployment; continuous verification; continuous testing; continuous compliance; continuous security, and continuous evolution (Fitzgerald & Stol, 2017). Continuous testing means executing automation as a part of the development pipeline. Continuous integration is the best-known one of these activities. It means a process that is automatically triggered and includes inner connected steps such as compiling code, tests, acceptance tests, validation, checking compliances, and building deployment packages. Continuous integration has several benefits: improved release frequency and predictability, increased developer productivity, and improved communication. Continuous integration requires a link between development and operations; thus, the concept is closely related to the DevOps phenomenon (Debois, 2009). Tasks such as continuous deployment, which refers to releasing good software build to users automatically, and continuous delivery that means deploying the software to some environment, making the development process flexible, but ensuring that products are delivered. Visibility of the development ensures that failures are prioritized for solutions as quickly as possible by whoever is deemed responsible. Other continuous software engineering tasks seek to improve product continuously, piece-by-piece, enabling delivery to customers as soon as it is completed and tested. Finally, continuous security seeks to prioritize security throughout all phases of the development lifecycle and even post-deployment. The system's maintainability depends on its architecture, which is defined through a set of initial design decisions during its creation. Some of the assumptions underpinning these decisions may no longer hold due to changes in the context or environment in which the system operates, or the architecture may not facilitate specific changes. When architecture is unsuited to facilitating new requirements, the projects need to

change. Continuous evolution is created to prevent technological or architectural decay due to a lack of changes (Fitzgerald & Stol, 2017).

Continuous operations mean processes that happen when the product is in active use (Fitzgerald & Stol, 2017) and the product's future development. Nowadays, most software products are made to be used in a long-term solution, so economic payoff forms from usage rather than a one-time purchase. However, so-called continuous use does not happen automatically: the intimal decision is made by the customer. Technology adaption models, such as the Technology Acceptance Model (TAM) (Davis, Bagozzi & Warshaw, 1989) or Unified Theory of Acceptance and Use of Technology (UTAUT) (Venkatesh, Morris, Davis & Davis, 2003) are not suitable for continuous software engineering. Since continuous development includes variables such as automation or unconscious product usage that does not exist in TAM or UTAUT, they cannot be implemented. Similarly, the previous technology adaption models do not consider trust as an essential aspect of development. Moreover, Fitzgerald and Stol (2017) define continuous trust as trust developed over time because of interactions based on the belief that an actor will act cooperatively to fulfill customer expectations. Continuous use is strongly dependent on continuous trust (Fitzgerald & Stol, 2017). Continuous trust evolves and is constantly being recalculated by users and can be deteriorating for the user experience. As mentioned earlier, modern software is rarely purchased once and does not involve further improvements from the producer. Continuous improvement activities are essential aspects of software quality and are based on the lean principles of using data in decision making and aiming to eliminate waste. Early activity in the continuous innovation space was beta testing, which became a widespread practice in the software industry. To keep the process flowing and growing, continuous experimenting and other operating tasks are required.

Even if continuous software engineering as a methodology is a new research agenda, the concept of continuous can be defined as an umbrella term. Many of the previously introduced agile development practices have similarities or the same practices with continuous software engineering. The evolution from heavyweight models to more connected development practices has created the necessary base for continuous development methods. In the agile world, the need to address the technical debt that accrues from not addressing potentially problematic issues when first encountered, but instead postponing these to be dealt with at a subsequent stage. Economic tradeoffs may prohibit the investments needed to remove technical debt.

2.8 The Essence of Software Engineering

As the previous chapters highlight, systems development is in constant change. The current way is an era of rapid technological change and new or evolving practice models. With dozens of different technologies, coding languages, and methods to guide the development and ever-changing business environments,

finding the most appropriate and efficient way to execute the project can be challenging. The development requirements differ with projects, as do the organizations and the teams with their people. Thus, the chosen elements should fit the motivations and goals. The teams face a problem when they need to choose from thousands of development practices. However, the research community has not gone unnoticed, as the underlying languages and software engineering methods are accepted as a standard to help develop and sustain high-quality products. Kernel means the common ground of different methodologies (Jacobsen et al., 2019). Using the common ground as a basis for practices will make teaching, learning, using, modifying, and comparing practices easier. The usage of kernels and chosen coding language combined create a standard structure named Essence. According to Jacobsen et al. (2019), Essence is common for all software engineering methods. This method relies on the following insight:

- methods are compositions of practices.
- some methods are more popular and have a large user base.
- only a few hundred practices out of thousands of methods are reusable.
- a common ground, or a kernel, is shared between all these methods and practices.
- focus on the essentials when providing guidelines for a method or practice.
- provide an engaging user experience when teaching and learning methods and practices.

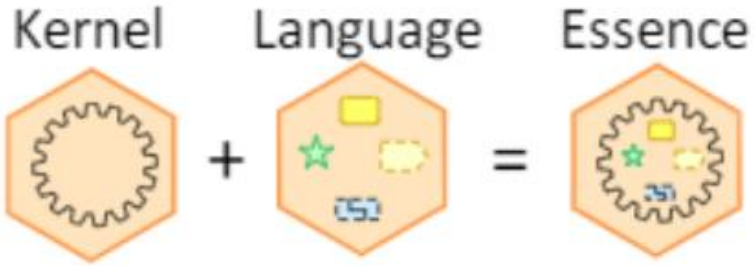

FIGURE 7 Essentializing process (Jacobsen et al., 2019)

The Essence language is simple, intuitive, and practical in describing the essence kernel with the elements that constitute a common ground. Essence method architecture is structured by essentialized methods and practices, the essence kernel, and language. Essentializing means that the method or practice is described using Essence. It focuses the description on what is essential by not changing the intent of the practice of the method. Thus, teams can combine practices to obtain a wanted mix, and new ideas can be essentialized and added to the practice library, where they can be selected later (Jacobsen et al., 2019).

Essentializing means that the practice of the method is described using Essence, and the focus is on what is essential. Jacobsen et al. (2019) explain that essentializing does not mean changing the intent: Teams can mix and match

practices to obtain a method they want and save the new ideas for later use. This liberates from uniform methods prison and lets developers freely select their desired ways of development. The essentials are usually a fraction about a subject that can participate without having all the details about the topic.

Essence approaches the development process from a user-friendly point of view that supports hands-on and tangible work. With the software that includes many different components, standards help to deliver the software representation. Kernel and the essentials can be touched and used using cards for visualization: The cards provide practical checklists and prompts, as opposed to conceptual processes, and therefore, the kernel becomes something the team uses daily. These actions are the primary difference from traditional approaches, which tend to over-emphasize method description and only consult people new to the team.

The Essence process starts with identifying the critical elements of software engineering. From a customer point of view, opportunities and stakeholder needs are considered. These impact the requirements and the product itself, as well as the workload and the team on the endeavor level. Some stakeholders benefit from the solution produced, and some will fund the endeavor. The solutions need to be delivered, and as the previous chapters explain, teams consist of motivated people. Like agile methodologies, Essence also highlights the importance of delivering value to the customer. The solution is the key to this.

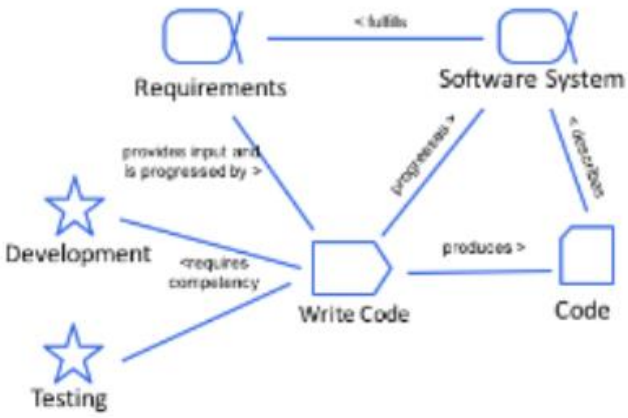

FIGURE 8 Simple Programming Practice Described Using Essence Language (Jacobsen, et al., 2019)

The Essence kernel and language provide an understanding and describing commonality and diversity of practices. Moreover, the Essence Language models things that software developers use as alphas, representing different dimensions of software engineering challenges. The essence kernel identifies a core set of alphas separated into different areas of concern. These alphas are Opportunity, Stakeholders, Requirements, Software System, Team, and Work and Way of Working (Jacobsen, et al., 2019). Thus, as the basics of the essence are like many other methodologies, current project practices can be used to essentialize the

process. In the next chapter, the concept of artificial intelligence is defined, and its development is explored.

3 ARTIFICIAL INTELLIGENCE

Learning a foreign language, calculating mathematical problems, or using commonsense are a few examples of actions that we understand as something that demands intelligence. Applying brainpower comes easily for humans and as something we consider natural to us. For instance, a child understands the concept of a light switch and turning the light on or off. However, creating the plan for completing such a task and making inanimate machine conduct is complex. Even if the child understands the concept of using a light switch easily, creating a machine that mimics the process includes many steps and complex combinations of small actions. Also, there is a question, can completing such a small task be considered truly intelligent.

This chapter covers topics relating to artificial intelligence (AI), the development of AI, and its areas of implementation. Intelligence itself is a multisided concept that has various definitions, depending on the perspective. When defining intelligence, AllWords Dictionary (2006) says it is "an ability to use memory, knowledge, experience, understanding, reasoning, imagination, and judgment to solve problems and adapt." Creating this has been a topic of human curiosity for centuries and has become an essential part of our everyday lives in the last few decades. Nowadays, intelligent machines have a significant role in many business environments. Dirican (2015) explains that the new trends like artificial intelligence, semantic studies, robotics and mechatronics developments, big data and mining, cloud computing, neural networks mean that business environments will look for new opportunities to lower costs and increase revenues. AI can transform industries, but the development and deployment of such technologies are not simple. The next chapter defines the history of AI further and briefly explains the critical concepts related to the topic, such as machine learning.

3.1 Definitions of artificial intelligence

As mentioned, people have tried to understand the concept of creating intelligence for centuries, ever since the ancient philosophers tried to explain and mimic the human mind. First attempts to build inanimate, humanlike objects trace back to ancient Chinese and Egyptian engineers a few thousand years back. However, modern artificial intelligence as we know it was not defined before the 1950s (Haenlein & Kaplan, 2019). Throughout the 1940s, the first ideas of intelligent robotics began to rise, and as the second world war raged through the world, there was a new need to automatize codebreaking. The processes allowed getting ahead of the enemy countries by breaking their coded messages and figuring out their strategies. After the revolutionary codebreaking machine Enigma invention, the first steps of modern AI were taken (Haenlein & Kaplan, 2019). The machine had significant importance when allied powers won the war. Writings of English mathematician Alan Turing defined AI by testing it rather than explaining it as a concept. Later, the ideas evolved to the so-called Turing Test, which described how humanity could not distinguish the machine from the human.

Even if the concept of AI is not a new one, and the modern concept was created in the 1950s, AI still lacks a clear definition. AI is complex with many different dimensions, and some may say that there is no definition altogether. Cambridge Dictionary defines artificial intelligence as follows:

> "The study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn."

> "Computer technology that allows something to be done in a way that is similar to the way a human would do it: To understand "natural language," computers must be equipped with artificial intelligence."

As can be seen in these descriptions above, Cambridge university describes AI as the development of machines with human qualities. It is widely accepted that the term means comprising machines that imitate human-like intelligent functions (O'Leary, 2013). However, Mackworth and Poole (2010) have a broader definition for AI:

> "Artificial intelligence, or AI, is the field that studies the synthesis and analysis of computational agent that act intelligently."

Mackworth and Poole (2010) mean something that acts in an environment and does something intentional by an agent. An agent can be, for example, a robot or a human. Mackworth and Poole are interested in the agent's actions and if those acts can be considered intelligent. By this, they mean that the acts are appropriate for the circumstances and have a goal. They are flexible for changing environments and goals. They learn from experience, and intelligent acts are

appropriate for choices given the perceptual and computational limitations. Mackworth and Poole (2010) also explain that a computational agent is an agent whose decisions can be explained in terms of computation. Their description gives a broader explanation for the AI, as the agent they describe can act on its own. As explained, Mackworth and Poole's explanation combines intelligence with technicality.

Mackworth and Poole (2010) describe AI as presented above and recognize its ability to act independently. Still, AI cannot be the same as the human mind, as it lacks certain qualities that people do (Kaplan, 2016.) AI is yet to the cable of being creative or having emotions, something that living things can. Therefore, some complex attributes of human Intelligence are yet to be reached with machines (Kaplan, 2016). On the other hand, AI has already surpassed humans in some actions. As described before, something is considered intelligent by creating memories and understanding, recognizing patterns, making choices, and learning. Also, Intelligence adapts from experience. As the computational power of machines has grown, AI can make machines act like humans, but they can also be faster and more even more humane. (Lehto, 2015.) However, AI as a humanly acting machine is not the way to use computational power to predict future outcomes.

Machine learning (ML), an attribute of AI where algorithms improve automatically through experience based on neural networks. Machine learning is commonly divided into three separate classes. First-class has supervised learning where we have access to the data and the "right answer," often called a label. In the seconds class, unsupervised learning has the set of data, and the ML systems try to find some standard structure in the data, for example, through categorizing. Finally, in the third class, reinforcement learning, the system learns a sequence of steps leading to a given goal (Saravanan & Sujatha, 2018).

The goal for AI systems is to learn independently, but this cannot be achieved without proper development. In the context of AI, development takes place over a longer time than learning and involves more changes in cognitive operations (Helm et al., 2020). Mikkonen, Nurminen, Raatikainen, Fronza, Mäkitalo and Männistö (2020) have explained the ML development process. As they start, used data must be available for training in the form of straight data sets. Data can be divided into various ways to training, testing, and cross-validation sets. Then, an ML model (data model) must be selected, together with the model's hyperparameters. The hyperparameters define how many and what kind of layers a neural network has and how many neurons there are in each layer. Next, the model is trained with the training data. During the training phase, the weights of the neurons in the network are iteratively adjusted so that the output of the neural network matches the "right answers" in the training material. The trained model can then be validated with different data.

3.2 Development of AI

AI deals with is about using computing power for learning from experience and predicting future outcomes. The fundamental differences between traditional programming and AI development are the goal of the product itself: traditional programs automate the task process of the user, whereas AI solves problems that are difficult to formulate, and the system needs to be adaptable. AI applications are beneficial for poorly understood problem domains, domains that contain valuable regularities in their databases to be discovered, and domains in changing environments (Zhang & Tsai, 2003). Still, traditional software engineering and AI development have similarities: Both try to solve problems, and processes start from understanding the task on hand. Both development processes start with discussions with the stakeholders and exploring ideas. However, AI developers, that are usually data scientists do not try to automate any tasks. AI development has historically been seen as a mathematical process: different algorithms are used to teach the system and create a learning software. Also, the process differences from the other software development, as the AI developers usually have personal definitions about AI, and no unified view has not been adopted (Sweeney, 2003). These habits can create silos between AI developers and other project members, as they cannot understandably explain their actions.

As explained in Chapter 3.1, AI has many different variations and implications that can fulfill different user needs. One of the main differences between software and AI development is that AI development takes longer due to the learning process. It also involves more changes in cognitive operations (Helm et al., 2020). Therefore, the project actions depend on the project on hand. Mikkonen et al. (2020) have explained the ML development process. As they start, used data must be available for training in the form of straight data sets. Data can be divided into various ways to training, testing, and cross-validation sets. Then, an ML model (data model) must be selected, together with the model's hyperparameters. The hyperparameters define how many layers a neural network has and how many neurons there are in each layer. Next, the model is trained with the training data. During the training phase, the weights of the neurons in the network are iteratively adjusted so that the output of the neural network matches the "right answers" in the training material. The trained model can then be validated with different data; thus, the product starts to learn and work "independently." However, integrating the developed function into the more extensive system is challenging, as is the maintenance of the whole product.

The development of artificial intelligence is not a simple process, mainly since AI does not have established, widely used frameworks or practices. The development of artificial intelligence requires specific knowledge of the area: System learning often requires attempts, mistakes, learning, and innovation. Even if AI is not a new concept, the process has many areas yet to be improved due to the technologies' challenges. The problem with AI development is its incompatibility with varying information environments. The development of AI is

driven both by research and the environment, with social goals in mind. As AI is opening a new field of business, practices, and opportunities, the technology itself is not always flexible enough for the nuances of its implementation environment. The goals of developers, users, and other parties do not necessarily align, as developers have pressured the work efficiently and still produce high-quality products (Mittelstadt, 2019). Because AI development processes are not formalized, there is a change for competing values, as the participants do not understand or care about their stakeholder's needs. AI development does not have a shared history, homogenous culture, identity, or ethics frameworks (Mittelstadt, 2019). In addition, AI systems are usually created by large, distributed teams that do not interact with the client and the users.

There is a lack of a particular model in the development of AI that would be significantly better than others. This makes choosing a model challenging, as organizations cannot directly deduce which model would suit them and whether another model is causing problems with processes, resources, or teams. Even if the AI development differs from the normal software development, both have a similar mindset that can be used. For example, modularity can be used both in software and in AI, in which separate parts, or modules, create together the product. Modules can be changed, but this does not affect the overall product (Mikkonen, 2020). However, practices such as agile models, are not used similarly than in software development.

3.3 Implications of artificial intelligence

AI-based technologies have become more and more common in recent years, and their importance is only growing. In some cases, people might know or recognize that they are in contact with an artificial being. For example, chatbots used in eCommerce websites start the conversation by introducing themselves as a robot, and their languages are usually inflexible and stiff compared to humans. However, there are products and processes in which AI is necessary, but not always a visible part of the product. For example, when using a calculator, the machine counts the user's operations, which is one way of using AI in our everyday lives. The implications can be a lot more complex than this and used together with versatile technological products.

Natural language processing means a computer system that can generate and "understand" natural language, such as English (Nilsson, 2014). This is a complex task from a development viewpoint: When communicating with each other, humans effortlessly use complex language and understand the user's process. People can understand even the most minor nuances of a sentence and produce answers in a blink of an eye. To generate the same action artificially with a machine is much more difficult. Languages evolve between intelligent beings to transmit personal "mental structure" to one other. Naturally occurring text means wrote or oral transmission of the mental structure (Liddy, 2001). This process

requires a similar, contextual structure for each participant to know that they also understand them. In addition, the other participants have the necessary skills that they can and will perform specific processes during communication efforts. To achieve a natural language process, the computer system should understand a message in natural language. Several frameworks and practices are used to achieve this, but the goal is to accomplish human-like language processing with the machine (Liddy, 2001). This process requires contextual knowledge and the processes for making the inferences assumed by the message generator. The foundation of developing such a system is about structures for representing contextual knowledge and specific techniques for making inferences from that knowledge (Nilsson, 2014). Liddy (2001) listed that the system should be able to paraphrase an input text, translate the text into another language, answer the questions about the content of the text and draw inferences from the text. Johnson and Valente (2009) worked with AI-based language and culture training systems and found that a critical issue with natural language processing is representing rich communicative acts. Hand-coded solutions have been found flexible changes but challenging to budget and develop fast, and so, the need for automotive processes is present. As mentioned, statistical and machine learning includes algorithms that allow a program to infer training data patterns and make predictions about new data. However, the challenge with this approach is that the machine makes poor predictions with new data, even if the training data is completed perfectly (Nadkarni, Ohno-Machado, & Chapman, 2011). The pipeline approach is being suggested, but the process might lack feedback on higher levels of development. In addition, the process lacks accuracy as one development error earlier in the process can affect the accuracy of language processing later (Nadkarni, Ohno-Machado, & Chapman, 2011).

Large bodies of information are stored in databases and can be used to fulfill user's needs. The design of adequate representation, storage, and retrieval is one of the implementations of AI. Automated, intelligent searching methods use the physical meaning as search criteria instead of the manual signal that has been used before. This can make information retrieval more effective and efficient (Vega et al., 2009) and is beneficial, especially within the fusion systems (Skulimowski, 2011). The classification system is used to retrieve information from the database of the signals that contain the most similar patterns. The process contains two steps: At first, the input pattern is classified by signals showing similar structural shapes. Secondly, the system computes the similarity measure between the input pattern and the signals of the corresponding group instead of navigating the whole database to calculate all similarities. This process reduces the number of needed computations drastically (Skulimowski, 2011). However, an intelligent information retrieval system requires a system that understands quires stated in a natural language and the problem with how to deduce answers from stored data (Nilsson, 2014). Also, understanding the quires and deducing an answer requires knowledge beyond that explicitly represented in the subject domain database (Nilsson, 2014).

Automatic consulting systems are used to help users to make conclusions about specialized subject areas or topics. For example, systems that help to detect and diagnose medical conditions and suggest treatment (Vaishya, Javaid, Khan, & Haleem, 2020) or how to build complex structures are some ways that can help to advise users with the problems on hand. However, representing and using the human expert's knowledge is difficult to produce with a technology. Human knowledge is often imprecise, uncertain, or anecdotal (Nilsson, 2014). Therefore, many consulting systems use rule-based deduction to implement AI into the system, meaning specific rules guide the dialogue between the user and the system, which deduces the conclusion.

When it comes to mathematics and other intellectual tasks, AI can prove or disprove a conjectured theorem. The theorem proving or disproving requires the ability to make deductions from hypotheses. To do this, intuitive tasks such as guessing and judging the problem and reflecting thoughts to previously proven theorems in an area can be helpful in the present case and help break the main problem into subproblems tasks (Nilsson, 2014). Automated theorem proving aims to solve more complex problems within the same resource limit than before. Many tasks can be formalized to theorem-proving problems, which helps develop AI that solves such tasks (Nilsson, 2014). Current automated theorem proving systems can solve non-trivial problems, and the technology can be used in many different implementations. For example, language, automation soundness, completeness, and solutions systems can be based on automated theorem systems (Sutcliffe, & Suttner, 2001).

AI systems use one of the implementations described above, but most technologies combine various practices for the most valuable product. Generally, AI products support three business needs: automating business processes, gaining insight through data analysis, and engaging people. Different systems can change industry processes and create entirely new products or services. For example, analyzing data, assisting uses, automating processes, and enhancing customer experience are a few currently available ways of using AI. In the future, the transformation of industries with AI can be even more drastic. However, AI development still faces challenges considering the lack of reality, societal challenges, and technological challenges.

3.4 Problems regarding AI development process and the developers

As Lee, Suh, Roy, and Baucus (2019) mention, AI development lacks commons aims and fiduciary duties, professional history and norms, proven methods to translate principles into practice, and legal and professional accountability mechanisms. Furthermore, as being mentioned, even if AI development has many similarities to software development, AI development lacks a variety of development methods compared to dozens of software engineering methodologies.

However, the challenges for both developers are not so different: both software and AI developers work in a high-pressure environment, with a constant need for cost reduction, increase profit, and deliver high-quality products Lee, Suh, Roy, & Baucus, 2019). Even more complicatedly, AI development can be done by various people with different backgrounds, such as data scientists, software engineers, and people with AI knowledge (Piorkowski et al., 2021). So, both the AI development processes and the demeanor of the developers are somewhat ambiguous. Due to this, understanding AI development can be complex, and communication issues may occur.

The development of AI functionalities is a complex process that involves people with different expertise collaborating. As mentioned in the previous chapter, AI development requires training the system by using data sets that are analyzed using algorithms. The Multidisciplinary AI development team includes data scientists and other AI-adjacent roles that do not have a knowledge mismatch (Piorkowski et al., 2021). However, the project teams consist of other stakeholders who do not share the same skillset as the AI experts, as they are not necessarily as skilled data scientists as the professionals. Communication gaps in code reading, code reuse, and code documentation are common, as well as communication problems due to diffracting motivation and communication behavior (Piorkowski et al., 2021). Translating complex business problems into a data science problem may also be difficult for data scientists. Therefore, the process needs a third party to bridge the broken communication between AI developers and other stakeholders. This can cause problems regarding project management and the openness of AI development.

Bostrom (2017) has studied the problems regarding the openness of AI development, such as openness about source code, science, data, safety techniques, capabilities, and goals. According to him, increased openness has both short- and long-term advantages, such as social benefits, openness around different product measures, and competitiveness. According to Bostrom (2014), the creation of AI faces two types of problems regarding the openness of the development: The control problem and the political problem. The control problem means that the designed AI works as developers intended and, the political problem means that the AI is recognized as something creating common good. As Bostrom (2017) suggests, one of the AI development problems is the product's openness. However, the control and the political problem are not the only problems regarding the missing bridge between AI developers and other stakeholders: lack of openness means slowing down the AI development and being less competitive.

As described previously, AI development lacks methodologies and practices regarding the development, and the role of AI developers can be challenging to understand by other project participants. Lack of understanding and communication can create silos between AI developers and the other project participants, and developing a fulfilling and successful product is difficult. Furthermore, as AI development does not necessarily fit the traditional software development process, it can be challenging to plan and predict. In the next chapter, the

suggested model for adapting continuous software engineering for AI development is presented, using the essence tools.

4 RESEARCH FRAMEWORK: CONTINOUS DEVELOPMENT OF AI

New technologies, business areas, rising opportunities, and occurring challenges require a flexible response from project participants. The development process of technologies and systems is rapid, and new needs might arise surprisingly quickly. AI development is considered similar to any other software (Mikkonen et al., 2019), thus benefiting from flexibility. However, when it comes to development methods used today, the literature points out that the development of AI has limitations regarding agility. Moreover, the current AI methods are simple, and the process is unpredictable due to a lack of standards or strict models. In addition, the current AI models are only practical for solving minor occurring problems (Bresina et al., 2012). These aspects of AI development can cause issues with the overall product if they cause issues to otherwise flexible project management. As mentioned in the chapters regarding agile, the development that welcomes change is uncertain but praises the unpredictable nature. This is not a common approach with AI development that depends much more on planning and a structured approach. Continuous software engineering methodology approaches the development process as a continuous lifecycle that bridges project lifecycle phases together, eliminating silos between project actions. This research aims to understand the challenges that may occur when adopting a continuous software engineering approach in the development of AI.

As told in Chapter 2.8, essentializing means taking the best and essential practices, tools, and actions and combining them for the best project development practices. For example, agile Essentials suggests the best development practice for agile is a continuous product cycle. Also, the ownership of the product is shared. Figure 7 presents the critical elements of the Essential Agile practices that

together create the basic toolbox that presents the starting point for team-based development.

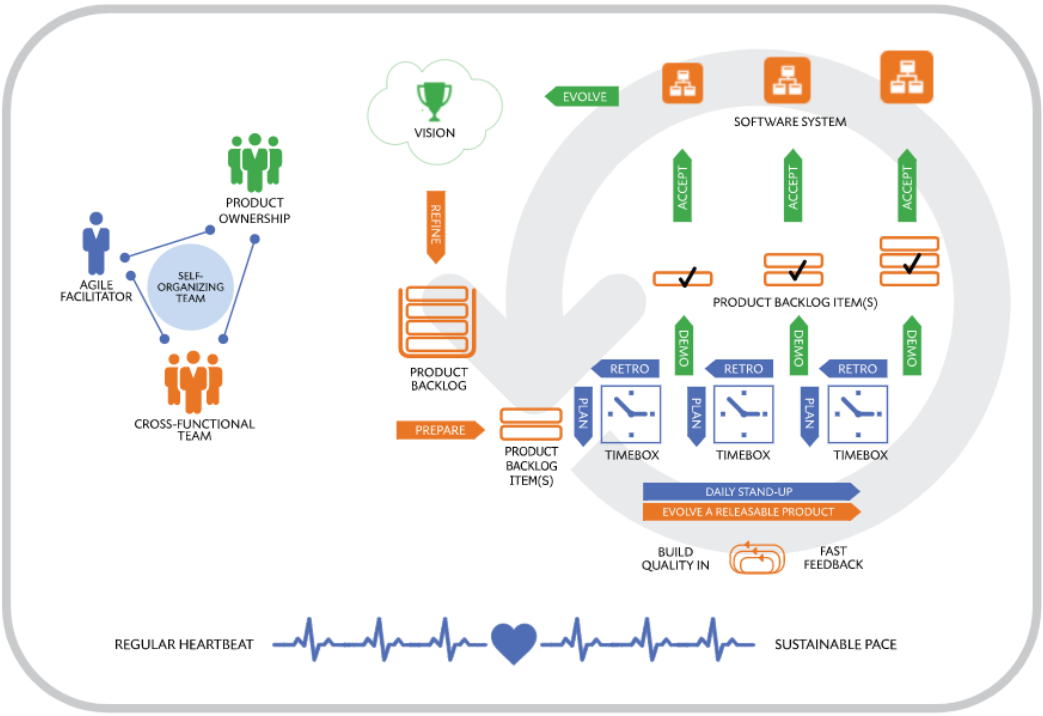

FIGURE 9 Agile Essentials - Overview of Practices (Ivar Jacobson International SA, ver. 2018.09) (practice library)

When it comes to fitting the continuous practises to the AI development environment, certain agile factors are essential for the successful implementation of agile practices in the new development environment. In this research, the Agile Essentials tools are a supportive factor for adopting continuous software engineering practices into the AI development environment. The following table presents the Agile Essential elements that the starter toolkit includes and their clarifications:

TABLE 1 Agile Essential elements (Jacobsen et al., 2019)

Agile Development Essentials	**Clarification**
Cross-Functional Team	The team contains the skills needed to get the whole job done. The team includes at least following competencies: stakeholder representation, analysis, development, and testing.
Product Backlog Item	Product backlog item consists of the following states: identified, ready for development and done
Test Case	Defines test inputs and expected results. Test ideas are captured, scripted and if possible, automated
Evolve a Releasable Product	Add value to a product and ensure it is usable and of production quality.
Software Change	A single change to the codebase that is made for a known purpose and is tested before it is integrated.
Manage Technical Debt	If value is built into a product one small value increment at a time, and just enough is done to make the product releasable each time, some desired changes to the codebase may be deferred, such as the fixing of non-critical defects. Each such item of technical debt should be logged, and its subsequent removal prioritized against the adding of more user-requested value into the product.
Shared Ownership	The team takes shared responsibility for the product, and no parts of the software system can be considered *"no go areas"* for any team members. This reduces the risks and delays associated with bottlenecks and single points-of-failure.
Minimal Design	Agile teams think hard about design but focus on adopting the simplest approach to achieving the known things that must be achieved next. This acts to minimize complexity, risk, and time-to-value, and maximize return-on-investment. The design strategy is then evolved continuously as more is learned.
Build Quality In	Quality is planned, designed, and built in the tests, and the item is not finished until the adequate quality has been achieved and enough design is done to ensure the right approach is taken
Automate as Much as Possible	If a software system is evolved one increment at a time, while ensuring its quality, many actions are repeated frequently. If these are not automated, they will be too slow and error prone.
Fast Feedback Loops	Key to agile development is getting as much feedback as possible, as early as possible, to converge on an accurate solution.

Thus, agile essentializing tools provide a base for reviewing the current environment readiness for adapting agile and continuous software engineering practices in the development of AI. The lack of agile development essentials illustrates the issues that may cause challenges in adopting continuous software engineering practices and what needs to change to make the adoption successful.

The following chapters explain the five parts of the research model. First part of the model aims to gather background information about current project development tools and mindsets. Four other parts (Business strategy and planning; Development; Operations; and Improvement and innovations) are based on the Fitzgerald and Stol's (2017) article about continuous actions in software engineering.

4.1 Current tools and usage of mindsets

Previously, agile practices have been successfully implemented into the software development process, thus making the heavyweight models less relevant in the current business environment. In many cases, agile is becoming a primary way for successful development. Agile practices ensure flexible and responsive project management that values interaction, working product, customer collaboration, and flowing planning over stiff and contact-oriented practices (Beck et al., 2001). However, different organizations prioritize various things when selecting project management practices. The first theme of the research model is to understand both physical and mental tools used in AI development and the current project development environments in which the interviewees work.

Even if AI functionalities are part of software and systems, their development differs from more traditional software products. Chapter 2 explained how the lightweight software and system development practices have changed over time and, thus, how agile practices were born. Continuous software engineering is one of the newest practices in the group of agile development methods. As the DevOps practices bring development and operations closer to each other, continuous product lifecycle involves other organizational activities to the process as well (Fitzgerald & Stol, 2017).

However, even if software development methodologies have evolved over the years, the same cannot be said about AI development. Just as the computing power and data available have increased and brought new possibilities for AI, there has been no significant evolution in development practices. As mentioned, AI development relies on static data sets and reviewing previous records and data. McMillan (2020) states that the ML development process starts from reviewed and verified data for training purposes. After that, the used ML model is tested with different training data. Therefore, data science has a vital role in AI development which involves less program development and more analyzing and getting insights from the data.

As mentioned, this part of the research model aims to gather information about current development tools and practices, especially continuous or agile frameworks used in AI development. Also, information about the project organizations is gathered to understand the interviewee's background and if similar backgrounds also have similarities in the practices used in the development.

4.2 Business strategy and planning

Continuous software engineering is a development methodology that bridges different organizational units that have traditionally been considered separate in the project life cycle. Business strategy and planning are a continuous software engineering phase consisting of business-related actions connected. These are connected to the development phase, and this linkage is known as BizDev. Continuous planning and budgeting are subphases of the business strategy and planning phase (Fitzgerald & Stol, 2017). This section of the interview aims to discover how AI experts describe their organization's business and strategic elements. Moreover, the goal is to find and highlight the challenges of adopting continuous planning and budgeting.

AI development differs from normal software development by planning the development process, as it requires creating an AI concept and testing it with data. However, the changing customer needs and evolving requirements are problematic when it comes to development. Moreover, the lack of adaptability can cause problems as the initially planned product need to be changed. Thus, the more flexible way of AI development has begun more relevant in recent years, as practices such as MLOps have emerged to tackle these problems (Karamitsos, Albarhami, and Apostolopoulus, 2020).

The research model aims to gather information about the current planning and budgeting actions in AI projects. Also, the goal is to understand the environment and possibilities for adapting agile and eventually continuous software engineering methodology. The Agile Essential is a basic toolbox covering all the standard and critical aspects of development in an agile team. Agile essentializing tools are also used to understand the possibilities of adapting the best methods to the development of AI. This part of the research also aims to highlight the decision-making process in the AI development team and if the AI experts took part in the business-related activities.

activities.

4.3 Development

AI and ML development aim to train the system to predict the future. Compared to more “normal” software that works in the present, AI needs to make predictions by using previously gathered data as a base for this. Data can be

divided into various ways to training, testing, and cross-validation sets from where the system learns to work "independently" by using algorithms to analyze data (Mikkonen et al., 2020). Combining this training-based development style with fluid and less plan-oriented agile development can be challenging. This part of the research model aims to understand the current development actions if they are an integrated part of the overall project development or more of an independent process.

Fitzgerald and Stol (2017) say that the continuous development phase consists of continuous integration, continuous delivery, continuous deployment, continuous verification, continuous testing, continuous compliance, continuous security, and continuous evolution. As mentioned, changing software development project requirements require rework and re-testing (Petersen, Wohlin & Baca, 2009). However, AI development is different from normal software development. The AI functionalities are tested by checking how an AI model works with test data sets; thus, the testing differs from the traditional software testing process. AI development is a mathematical process in which an algorithm is fed data, and the goal is that the model learns and can make predictions in future data. As explained in chapter 3.3, the trained model can then be validated with different data; thus, the product starts to learn and work independently (Mikkonen et al., 2020). Thus, the research model highlights the actions needed to produce a functioning product that works as intended.

Using essentializing in the development phase of AI could bring new possibilities by bringing a more agile and fluid approach to development. The goal is to get different participants involved and talk the common language when it comes to the project on hand. As the name suggests, the Agile Essentials describes how an agile process is used essentialized way. With essentializing, it is possible to combine continuous software engineering methods with different development needs, such as framework practices, coding languages, and other actions that can be used. Moreover, as only the best and most needed practices are considered, the development process can be used in various situations. The necessary changes can be easily made, and the participants can learn from past projects.

4.4 Operations

The operational actions of continuous software engineering bridge development and operational actions together, as in DevOps. The goal is to bring these traditionally separate actions together and ensure the usage of the software. Therefore, the operational phase of continuous software engineering is like DevOps methodology. According to Fitzgerald and Stol (2017), continuous operations include three actions in this phase: continuous use, continuous trust, and continuous run-time monitoring. These tasks require recognizing the difference between intimal adoption and continuous software usage and trust-building

between developers and users. Moreover, the project participants need to feel that their mutual goal is to fulfill user expectations (Fitzgerald & Stol, 2017).

Karamitsos, Albarhami, and Apostolopoulus (2020) have conducted a study about DevOps practices for the ML application. However, the current development practices are complex and time-consuming. Furthermore, the regular ML models require significant and costly maintenance, improvement, and monitoring efforts in large-scale projects. Karamitsos, Albarhami, and Apostolopoulus (2020) suggest applying continuous integration and delivery principles, practices, and tools to minimize waste, support rapid feedback, explore the hidden technical debt, improve value delivery and maintenance, and improve operational functions. Due to similarities with the development of AI and ML, some of the ML practices would use with AI as well. However, continuous AI development has not been widely researched, so the conclusion cannot be made directly. Furthermore, the implementation of other continuous practices in AI development is yet to be explored in research. Karamitos, Albarhami, and Apostolopoulus (2020) concluded that especially continuous improvement is problematic due to the tedious data collection, data extraction, and data cleansing that AI development requires. These steps are time-consuming and challenging to apply to continuous, agile development. Thus, the AI development.

It has been discovered that agile transformation is essential to improve the efficiency of the companies to optimize the lifecycle delivery, break the gaps, and create a continuous feedback loop between the business users and development teams (Karamitsos, Albarhami & Apostolopoulos, 2020). The so-called MLOps approach aims to combine machine learning with DevOps principles. Practicing MLOps means using automation and monitoring ML system construction. These steps include integration, testing, releasing, deployment, and infrastructure management (Karamitsos, Albarhami & Apostolopoulos, 2020). In addition, the model aims to understand what practices are used after the project goes into production and how the product is monitored.

4.5 Innovation

According to Fitzgerald and Stol (2017), improvement and innovations are the basis of the continuous software engineering life cycle. Agile methodologies embrace unexpected and experimental development and are a fundamental aspect of agile methodologies (Petersen, Wohlin & Baca, 2009). Moreover, the product life cycle is continuous, and a new planning phase starts when new opportunities are recognized. This part of the research model aimed to gather how the AI experts feel about new opportunities, innovation, and technologies. Also, the ending of a project was discussed, so in other words, when does the AI expert move on to the next project.

As mentioned, purchasing a software product is rarely a one-time buying but rather a flexible tool that needs to evolve to the customer's changing needs

and business environment. An innovative mindset means transforming new ideas to create business value (Fitzgerald & Stol, 2017). The model aims to explain the mindset that the AI experts have towards the innovations: if they are motivated to suggest a new way to create AI, both the technologies and the mindsets used for project management. Also, the relationship with the customer is a topic of interest, as the customer is part of the decision-making process.

5 RESEARCH DESIGN

This chapter explains the collection of empirical material for the research analysis. Furthermore, the chapter explains how the empirical material is collected using interviews and its different approaches to finding the research questions. Finally, the data gathered is analyzed and used to find out the answers to the research questions. The interviews were conducted as a semi-structured research method, and the chapter highlights the reasons leading to this method. The research is done by interviewing people working with AI-related projects to produce information about current development practices used in the development. The goal is to produce information about current continuous methods used in AI development and the challenges that adopting them wider might face.

5.1 Goals of the empirical research

As the literature review suggests, agile practices are desirable in a changing environment, yet unpredictable to adapt. When new technologies rapidly evolve and become more meaningful for the people and the organizations, the development methods have changed to make the process more fluid. Furthermore, the ability to harness technology can provide an advantage to competitors makes successful development practices essential to achieve. Still, it is crucial to ensure that the flexibility does not affect the delivery or the quality of the product produced.

Continuous software engineering actions are Business & Strategy, Development, Operations, and Improvement and innovation. The subphases contain smaller tasks that need to be considered depending on the organization and its products. Historically, business and development departments have had a separation between them and had competing goals (Fitzgerald & Stol, 2017). The often-occurring problem is that the development department looks for new, simplistic technological solutions without considering the complex reality of the business environment.

Continuous software engineering is a relatively new practice and is not widely researched. The topic lacks studies, especially when applying the practices into complex technology development, such as AI or ML development. The capabilities of continuous software engineering practices seem exciting to adapt to the development of AI: AI technologies are about predicting and learning. Therefore, developing AI systems continuously to new fit requirements would be practical action. DevOps principles have sparked curiosity in AI implication developers, as methods such as MLOps (combination of machine learning and operations) has been adopted in recent years (Fursin, Guillou, & Essayan, 2020; Mäkinen, Skogström, Laaksonen & Mikkonen, 2021). Primarily continuous

integration has been used in ML and AI development, but wider usage of DevOps and continuous software engineering methods have yet to be adapted.

This research aims to understand the challenging aspects of adapting continuous software engineering into the development of AI. If these elements are well known, this might help the participants prepare themselves for any negative outtakes that the development might face. Darke, Shanks, and Broadbent (1998) say that empirical research aims to provide a deeper understanding of phenomena with some aspects that are not understood. Also, the goals are to create a description or theory and test a theory about the phenomena on hand. Thus, the approach fits this research due to little understanding of the challenges that adopting continuous practices for the development of AI may bring. A qualitative research method was selected and used to analyze the development phenomenon and give a more defined description of the topic.

5.2 Data collection

Data were collected using semi-structured and thematic interviews using a pre-created question structure. The other two interview styles that could have had been used are structured and unstructured interviews. A complete interview structure is pre-done with the structured interview, and any parts cannot be improvised. The unstructured interview is closer to the everyday conversation that flows naturally but might easily get sidetracked. The semi-structured interview was selected since the answers can form freely, but the interview topic remains the same. Thematic, semi-structured interviews able the rich data obtention from the interviewers and leaves room for individual opinions and thought processes. Also, the interview style allows asking additional questions and if the interviewed person has anything to add. Still, this method does not come without any possible problems: As in any interview situation, the thematic interview is also an artificial situation, not a flowing conversation between two people with shared factors. Thus, this can cause a lack of trust between the participants and therefore change the answers, negatively affecting the quality of the data gathered (Myers & Newman, 2007).

As the study aims to explain phenomena regarding AI development processes, the people being interviewed were working with AI development, or they worked closely with AI-related projects. The goal was that people interviewed worked in different organizations and a wide range of AI projects. Thus, the people were able to explain the development practices used in everyday work and their experiences. The people interviewed were intentionally chosen using two ways: firstly, a snowball effect, in which the first participants were asked to propose any. Secondly, asking directly from people known to be working closely with AI development projects was used to branch out the people interviewed. Eventually, the interviewees' backgrounds varied from research assistant to a service manager, and the experience from months to decades work with AI technologies.

The interviews were conducted individually using remote meeting platforms Zoom and Skype. Due to the ongoing COVID-19 pandemic in spring 2021 and the fact that interviews were done nationally internationally, the remote meeting platforms provided safe choices for conducting interviews. Interviews were conducted in English and Finnish, but the basic structure and questions for the interviews were the same. The planned time scope for the interviews was 30 to 45 minutes, and the average time that the interviews took was 35 minutes, which did not include an introduction, instructions, or the end world. Every interview was recorded and transcribed (and translated in English if needed) for the analysis.

As mentioned, the interviews included the introduction, the recorded interview part, and the ending segment. Only the interview part was recorded and included several discussion topics. Moreover, the interview had five parts: current job and challenges; Business Strategy; Development; Operations, and Improvement and innovation. The first part aimed to introduce the interviewed person, and they were able to explain their background with AI. The other interview topics are constructed based on the continuous software engineering subphases, and the questions were based on the phases that these subphases include. Due to the topic of the thesis and the intention to gather relevant information, the questions about product life cycle phases were selected to be conversational nature for the interview and yet gather data for the study. Fitzgerald (2017) has defined more minor phases for each continuous software engineering subphases, and the questions are created in mind. The goal was to gain insight into the practices used in organizations today and compare them to practices that continuous software engineering development methodology proposes.

The first theme contained questions about the interviewees' current work role, how the work is dived in their private project groups, the tools they use in their work, and if they use any framework mindset, method, or theory to guide their work. As mentioned in the previous paragraph, the first theme was constructed to understand the background of the interviewed people and what aspects of their everyday work life include. The intention was to gather information on how different AI developers work and how their projects are conducted.

The second theme was about Business strategy. As mentioned in chapter 2.7, business strategy and IT, development has been seen as competing departments (Fitzgerald & Stol, 2017). They are especially prominent in the development of AI, which could be seen as new, innovative technology. However, fitting the newest technological solutions to the complex business environment, sometimes without any previous experience with results, can be seen as risky from the business side of the process. The purpose of this theme was to understand what kind of challenges there is in AI development from the business side of the process. Thus, the questions were about interactions with other project participants, requirements, and resources. Moreover, the theme highlights the gap between business and development and how the clash can be seen in current projects using AI.

The third theme is Development. Some research has highlighted that AI development has some aspects of continuous development activities already in use: for example, integration and delivery are used when developing AI products (MLOps lähde). Furthermore, as mentioned in chapter 2.7, continuous integration can enhance release frequency and predictability, increase developer productivity, and improve communication (Fitzgerald & Stol, 2017). However, continuous development requires a bridge between development and operations, and the questions in this theme are created to craft information about continuous practices used in AI development. Moreover, one of the things that previous research highlights are that AI developers usually work separately from the other project participants. Thus, the theme aimed to determine if there are silos between developers and operational workers that make continuous development actions challenging or even impossible to adapt.

The fourth theme is about Operations. The interview questions touch on maintenance and if the AI producers interact with the users after the product release. As said above, the bridge between development and operations makes continuous development possible, and is these two phases are fundamental parts of the DevOps practice. DevOps practices have been studied before and used primarily in the ML development context. MLOps means using automation and monitoring on ML systems (Karamitsos, Albarhami & Apostolopoulos, 2020). The steps to do this include integration, testing, releasing, deployment and infrastructure. As the thesis topic is continuously development, the questions are formed to get information about active usage. Continuous software engineering considers that software is not just a one-time purchase. However, the customer decides to if they keep using the product. Therefore, the questions were more aimed at the continuous usage and the user-developer interaction.

The last theme of this interview includes improvement and innovation. The question is about improving the product, even after the release and adding innovations. The questions include product improvements and the relationship between the customer and the developers, the end of the relationship, and when the project participants move on to other work. Moreover, the theme aims to highlight continuity or lack thereof in the customer-product developer relationship. As mentioned before, nowadays, the software is not a one-time purchase, but moreover, a product is in continuous use. Therefore, the need for improvement is constant, as the products need to adapt to the changing business environment and customer needs.

5.3 Data analysis

To create structured information, the data is analyzed further. The analysis process means structuring and transforming data so that results could be understood and conclusions to be drawn. As the thesis aims to understand the challenges of adopting continuous development to the development of AI as a phenomenon, the data was collected through interviews. Thus, the collected data was

qualitative, and the analysis method was selected depending on this. The data were analyzed with qualitative thematic analysis to identify key concepts from interview data. The purpose of the thematic analysis is to understand current development practices and if continuous development can be used in this context due to the nature of AI development.

There are three different approaches to coding the data. These are deductive, inductive, and integrated approaches (Cruzes and Dybå, 2011). The deductive approach means that the coding has some previous themes expected to be found or reflected from the data. The inductive approach means that data determines the themes. The integrated approach combines both above, as the codes are created from emerging themes and data and the pre-constructed codes (Cruzes & Dybå, 2011.). However, coding data does not come without problems that may occur during the analysis process. For example, it has been said that coding is too general. Identifying the desired themes from the data can blind the information because coding does not have a clear definition. Also, if the codes are vague and do not have a clear definition, they lack their unique semantic, and the results do not present truthful information about the phenomenon.

The data of this master's thesis is coded using an integrated approach. As continuous software engineering practices are the framework that is used, there is a need for some previously defined concepts. Continuous software engineering phases (Fitzgerald & Stol, 2017) create a basic starting listing for the coding. On the other hand, the combination of AI development and continuous development lacks research. Therefore, the inductive approach is also considered, so possible new emerging themes can be analyzed.

6 EMPIRICAL FINDINGS

This chapter goes through the empirical findings gathered through semi-structured thematic interviews. Altogether, eight people were interviewed with the same interview pattern, but the discussion could flow on the topic. The data from these were analyzed. The goal was to understand rising themes and concepts that could challenge adapting continuous software engineering methods to AI development. All the interviewed people work or have recently (in under three months) worked with the projects, including AI development. As the background and work titles of the interviewee varied, the umbrella term "AI expert" refers to the interviewee, as all the work consists of a form of AI function. The term "AI developer" refers to people working primarily in the development part of the project. However, roles varied from project management to research to coding.

TABLE 2 Interviewees and their work titles

Interviewee	**Work title**
Interviewee 1	Research assistant
Interviewee 2	Data scientist
Interviewee 3	Research assistant
Interviewee 4	Service Manager
Interviewee 5	Professor in department of software engineering
Interviewee 6	Senior lecture in software engineering and applied AI
Interviewee 7	Regulation specialist
Interviewee 8	Software developer

6.1 Overview

The analysis phase aimed to identify the factors that may cause challenges when adopting continuous software engineering practices into the development of AI. As mentioned in chapter 4, the research model aims to adapt the methodology elements and find the essential themes regarding continuous practices in the AI context. The data previously gathered was assigned both deductive and inductive code. The deductive codes were assigned using the continuous software engineering phases and actions as codes. Inductive codes are under the deductive codes, and the data that did not fit into either group were gathered and assigned as the code "Observation." As the topic of the research was to understand the challenges regarding adoption of continuous software engineering, the further analysis concentrates the lack of certain codes.

TABLE 3 Assigned codes and their occurrences within the data

Deductive code	Inductive code	Occur-rence
Continuous planning	Shared ownership	4
Continuous budgeting	Shared ownership	2
Continuous integration	Software change	1
Continuous delivery	Evolve a releasable product	2
Continuous deployment	Build quality in	3
Continuous verification	Test case	2
Continuous testing	Automate as much as possible	2
Continuous compliance	Observation	0
Continuous security	Observation	0
Continuous evolution	Product backlog item	5
Continuous use	Fast feedback loops	1
Continuous trust	Fast feedback loops	1
Continuous run-time monitoring	Manage technical debt	2
Continuous improvement	Minimal design	1
Continuous innovation	Shared ownership	2
Continuous experimentation	Shared ownership	3

The thematic analysis revealed that some continuous software engineering concepts did not appear. Fitzgerald and Stol (2017) says that continuous trust is one of the continuous project development's operations activities. *Continuous trust* is the trust developed over time based on the belief that customer expectations are fulfilled without exploiting their vulnerabilities. However, the trusting relationship defined as such was not mentioned. Instead, a good relationship with the client was discussed. This finding forms the first said to be valuable. However, none of the interviewees defined this relationship as a continuous element. These findings form the first empirical conclusion.

EC1: Continuous compliance and continuous security were not present within the data.

Also, continuous trust and continuous security were not operationalized in the interview outline, thus forming the first primary empirical conclusion PEC1.

PEC1: Continuous compliance and continuous security were not present within the data.

6.2 Oversight of used tools and mindsets

The first theme of the interview included questions about the current work role of the interviewee and the tools and mindsets used in AI development projects. The goal was to oversee the coding languages, programs, and systems used in AI

development. In the data, seven people mentioned using Python as a primary coding language in AI development, and two people mentioned Java. However, when it came to software used in the development, the answers were more distributed. Especially information regarding database systems and possible cloud infrastructure varied greatly as all the interviewees gave different answers. For example, Interviewee 2 mentioned that almost all the coding was done with Python and the data preparation with Dataprix. However, Dataprix was not mentioned by any other interviewee.

> [Development was] 99% or like fully Python based, but then some of the data preparation was done in a Dataprix, you know the kinda spark service, so that was used. … I would not say that I followed any framework with intend but rather trying to have like the mindset within many of the frameworks. – Interviewee 2

Altogether, interviewees mentioned five coding languages, eight systems, and four database systems. As explained, other than the primary coding language (Python), the tools varied greatly. In some cases, the customer offered the data used in the training of AI. However, the AI developers used internal training data in training, as the product in development would later be implemented into customer's systems and using customer's data. Interviewees mentioned a few reasons why they used specific tools in the development: For instance, some people had a background in developing AI as a hobby or during their studies. Thus, the developers' current tools were the same that they were most familiar with in the past. Usually, the AI developers worked either alone or in a small group with other AI experts, so they were the only ones that needed to understand the code.

> The data scientist experiments have been done with whatever tools that they that data scientists have been conveniently. – Interviewee 5

Also, some interviewees worked in a project organization that offered practical tools for the developers; in these cases, the organization usually had prethought processes that the developers used and given tools. They used practical tools that depended on what the developers were most used to and what the organization offered. These findings form the second empirical conclusion:

> EC2: Other than using Python as a main coding language, the practical tools of development varied greatly.

Frameworks help pinpoint the essential points of the process and define the common ground for the project participants (Greenfield & Short, 2003). However, there were only a few points in data in which any framework or mindset was well defined. The only frameworks mentioned were SCIKTLearn (machine learning in Python), SAFe, DevOps, and MLOps. More important than knowing and strictly using any framework was to use the most reasonable practices in development. For example, Interviewee 4 explained that their organization had adopted an Agile mindset, but no specific lightweight model was used in AI development.

> [Organization name] is adopting and has adopted agile. – Interviewee 4

Other interviewees explained that they did not use any methodologies in the development but rather a mindset built upon many different methods. This seemed to be the general approach using the frameworks, as most of the interviewees seemed familiar with several of them but did not use any of them strictly. Scrum was present in the data, but only briefly. Rather than using the entire framework, and they used some practices such as sprints. The lack of generality in data was expected when asking about tools and methodologies, as AI development lacks clear definition and variety (Sweeney, 2003). This may cause unpredictable processes, primarily as the developers worked individually or in a small group. In three cases of eight, the interviewee said that the teamwork mainly happened after the development at the point in which the participants had presented the development efforts.

The frameworks provided more of a mindset than a practical toolbox for the development, but some elements were used. Two people out of eight interviewed mentioned that their project groups used agile practices in the AI development projects. Interviewee 2 mentioned agile frameworks by name, which were Scrum and SAFe. However, they explained that those were not used in the AI context but software development projects in general. Interviewee 4 described agile work practices, such as sprints and iterations, used in the development but said that there was no framework they used.

> We go into general development cycle used in whatever all other aspects of the development of the product meaning there are two-week sprints and before the sprint starts there are a prioritization and we decided what we want to accomplish. – Interviewee 2

> All people are caught in all cases, work needs to be prioritized. The work is planned in sprints. We have 4 planning periods per year. If we speak about doing AI then you can think that sprints always produce something that can be put into production, with the next sprint it will be improved and expanded. – Interviewee 4

These findings form the third empirical conclusion and moreover, the second empirical conclusion:

> EC3: Frameworks are known but mostly used partly or as a unidentified mental guideline.

PEC2: Frameworks offer support for AI project development, but they are not used systematically or accurately in the process.

The tools and mindsets used by interviewees varied when it comes to AI development. Essentializing aims to take the best practices and combines them with languages to create the essence kernel, a combination of best practices. Jacobsen et al. (2019) offer Agile essentializing processes where the most reasonable project development practices can be selected and used. Moreover,

when aiming to use continuous software engineering methodology in the development of AI, essentializing can be used to take only the most applicable continuous practices for AI projects. There seemed to be a good starting point for using agile essentializing, as some interviewees already had picked the best practices from various projects. However, the lack of named frameworks could indicate that the AI developers do not have deep knowledge about the variety of development frameworks available or the practices.

6.3 Business strategy and planning

In the interviews, people were asked questions about teamwork, requirements, and resources. When asked about teamwork, collaboration with others was mentioned by six interviewees out of eight. Collaboration of business and development teams is one of the principles mentioned in the Agile manifesto (Beck et al., 2001) and certainly an essential element when essentializing agile. Furthermore, continuous planning approaches the future with a dynamic and holistic planning style. Thus, a collaboration between stakeholders is vital to ensure planning and execution happen accordingly. However, in the interview data, collaboration, and the importance of communication between the interviewees were mentioned.

> I think collaboration is very important when developing software. I think independent work can be sometimes it is dangerous because it leaves like human made errors to product. So, there can be lots of participants you have to collaborate with. Depending on the project. At least you must collaborate with the customer site from the project. – Interviewee 8

> There is an exchange of news every day, so if there are anyone throws ideas then, that is mainly the interaction. – Interviewee 3

Four interviewees said that AI experts mainly worked independently, as there is no significant development group for AI functions in their organization. However, the work was not wholly independent as these interviewees said that research, iterations, and review of the results were done in teams. Two of the eight interviewees worked in big organizations, in which the projects were highly regulated. They said that collaboration and communication were necessary to produce AI functions that worked accordingly and were helpful. One of these two worked in a government-led company, in which the developed AI projects were internal. Therefore, the project team and stakeholders worked in the same organization, and the communication between the participants was active throughout the project.

> We work closely with other units: our job is to take care that everything that is made, works. – Interviewee 4

However, other interviewees said that in their organizations the independent work was more prevalent. Even if collaboration and communication occurred, they were not considered vital or necessarily encouraged. Interviewee 3 said that daily, informal discussions with the teammates were critical when new ideas and possible problems were examined. However, the collaboration was otherwise minimal. Two interviewees of eight mentioned that due to the COVID-19 pandemic and the remote work requirement, the work environment had recently changed: The work was primarily individual, and collaboration was more challenging to establish with remote work tools such as Microsoft Teams. Moreover, there were also cases in which individual work had caused challenges in the project development: three out of eight say that they did not clearly understand communication or other people's work efforts. These people worked with small, unregulated projects and two of them had just recently started working on the organization. One interviewee even said that they did not understand the project that they worked with:

> Personally, the point of the project is unclear to me. – Interviewee 3

It seems that the collaboration level varies significantly in AI projects. Two interviewees worked with the projects that were regulated, and the customer was involved with the project. They also mentioned the importance of the inner relationships of the team. These things were mentioned to be success factors and make error detection and fulfill the project needs easy. On the other hand, the other interviewees worked on a smaller project, and the project environment was informal. Even if outer customers ordered the AI projects and provided the resources, customer involvement was not constant. Sometimes, the customer was seen as a necessary evil controlling the project and sometimes made the development work more difficult. These findings form the following empirical conclusions:

> EC4: The communication between AI developers and project participants is mostly unformal.

> EC5: Lack of clear communication between AI developers and project participants causes problems with understanding the work efforts or the project as whole.

Jacobsen et al. (2019) say that one of the critical elements essentializing Agile is a cross-functional team and shared ownership. The cross-functional agile team includes competencies such as stakeholder representation, analysis, development, and testing. Also, the team shares ownership of the product, and there is no area considered a "no-go" for the team member. Cross-functional teams are critical elements of lightweight models, and team members are highly motivated about the project (Schawaber, 1996). This reduces risks for failure and helps to detect errors. Continuous development consists of many acts simultaneously, as the development process is a continuous flow of events. However, AI development differs from normal software development, as data scientists and

mathematicians traditionally do it. The development consists of creating AI concepts by using algorithms to train the system. The process is usually done by data scientists (Schawaber, 2003), that might face difficulties when explaining their development efforts to other team members. When analyzing the interview data, people gave different answers regarding collaboration and interaction with other team members. Six interviewees out of eight said that collaboration is an essential part of the project consisting of AI because it was part of the more prominent software or system. However, collaboration might have had been merely integrating one person's work efforts into a more extensive system.

> So, for example, it could be that in some projects, you decide," I'm going to develop this feature or this subcomponent". And then you can do and then you can integrate if the works from others. – Interviewee 6

Combining a cross-functional team shared ownership and AI development seemed to be that AI development requires different expertise than "regular" software development. Interviewee 4 explained that the AI development unit was small in their organization and that people with competencies regarding AI were challenging to find. Also, this meant that allocating the human resources was challenging as the AI developers worked thoroughly with projects.

> Practically everyone is doing multiple projects. All people are caught in all cases, work needs to be prioritized. The work is planned in sprints. Resourcing is challenging. Nowadays, there are more roles, the number of staff has increased. .. Plus, there is less and less expertise. AI needs to have analysts and people who know algorithms, it is not that simple to have them on every branch. – Interviewee 4

Interviewee 5 explained that the human resources depend on the project:

> Some of them have a very closely related ways of working and the researchers work ideally, in the same projects, although this year [due to COVID-19] has been not the ideal. And then in some other projects, people work independently of each other. – Interviewee 5

As Fitzgerald and Stol (2017) explain, budgeting has traditionally been an annual event, but continuous budgeting facilities change during the year. Four out of eight said that the project and the resource were usually planned with the customer. Three people of these mentioned that a person in a role such as product owner, senior developer, or project manager was the one that had more responsibility when it comes to resource planning and decision making. Thus, planning was done mainly by trying to fulfil the customer needs as accordingly as possible. A project team member had a bit more responsibility in this regard. However, two people out of eight said they did not know how the project was planned or planned before getting involved with the project. These two people had been included in an ongoing project and felt that they needed to do the actions to understand the project and the requirements. One of these two was frustrated about the situation and had difficulties managing their workload and tasks.

> I do not know how they [resources] are decided. It feels like my time is being spent on everything else that is not related to my work. – Interviewee 3

EC6: AI developers did not usually take a part in the allocating the resources.

In addition to human resources, the budgeting of the project was usually determined by the customer. For example, only one interviewee worked with internal AI projects, in which the AI development unit and the business unit of the organization decided the budget together. Other interviewees worked with the projects regulated by a contract that also determined the budget. Therefore, continuous and fluid budgeting was rare, as the customer had control of how much money was used in the development.

> If we get green light from a company, the project price already includes a lump sum of money either from the company or in the form of some collaboration. And we try to do our best with that or go as far we can go with that sum of money. – Interviewee 5

Even if the developers usually were able to make suggestions when adding new aspects to the project, the customer made the ultimate decision. Some projects had additional resources planned and used if there were changes in the scope of the project or any added functionalities.

EC7: AI developers can make suggestions if there is a need for budget changes, but with the contract-based projects, the customer makes the ultimate decision.

As mentioned, shared ownership is one of Jacobsen et al.'s (2019) agile development essentials. However, a close relationship with the customer was difficult to achieve, as in many cases, the teams had a person in a customer relationship role. Therefore, the other team members did not usually stay in touch with the customer daily but rather when a new product or process was presented. Data shows that the developers felt unsatisfied since the customer was the ultimate decision-maker and gave the blessing for the product. Three interviews of eight said that the customers changing their minds cause unpredictability and difficulties in the development.

> For example, the clients might change their mind in every two weeks as it has happened in some projects or the approach to gain some insight to something has changed. – Interviewee 1

It seemed that AI expertise found the uncertain mindset of customers frustrating, mainly because the customer had the last word about the product. One interviewee said that the customer did not always understand AI functions and even less of the development of AI.

EC8: Customers did not have a clear understanding of AI functionalities.

EC9: AI developers and customers did not have a clear dialogue when it came to product development.

The biggest problem regarding essentializing agile development and continuous practices in Business & Strategy seemed to be the role of the developers. As mentioned, six interviewees out of eight found collaboration necessary. Still, in many cases, the AI development process was done independently. Understanding the role of AI development can be different, as the development differs so much about software development. For example, AI developers and data scientists have different development actions compared to system developers. However, the lack of proper understanding in both ways seems to go another way around: Three people out of eight said they were uncertain about other people's work and the team members' work effort had on one another.

I think this is a rather difficult question. At least knowing or having any kind of idea how much my work influences other … I do not really know but I do think that my work is influenced by others. – Interviewee 1

EC10: Developers did not have a clear understanding of their role as a project team member.

As mentioned, one of the agile development essentials (Jacobsen et al., 2019) is a cross-functional team and shared ownership. Even if a project team has cross-functional members, not knowing their exact work roles and competencies can cause uncertainty. AI development lacks shared aims, fiduciary duties, professional history, and norms (Sweeney, 2003). Thus, understanding the professional roles can be complicated as the roles and practices are so dependent on the participants and the projects. Not understanding the project participants may cause uncertainty and frustration in both the developers and stakeholders who try to define the common ground.

I would say it's difficult. Every team member knows the kind of impact they can have on the larger project. But it's difficult to gauge how much impact there would be. - Interviewee 6

The most significant challenges regarding AI experts and business and strategy actions of continuous software engineering are the lack of common ground caused by the nature of the AI. Thus, the relationship with the customer and other project participants was vague, and the AI experts did not take an active role in the project management. These attributes may cause challenges when adopting continuous software and essentializing any agile methodology to the project management process. These findings for the third primary empirical conclusion.

PEC3: Due to lack of active communication between AI experts and other project participants, the AI experts often work in a silo. Thus, they do not participate business and strategy related activities as actively as other project participants.

6.4 Development

The interview questions about the development of AI projects included how functionalities are added, how the product is tested, and when the product is ready for production. As mentioned before, AI development lacks various frameworks as software development (Sweeney, 2003). Therefore, when asked about the development practices, there were differences between the interviewees' answers as the project tools and development methods vary greatly. Moreover, any framework did not guide the development processes, and the AI experts seemed to be uninterested in them. The development actions with iterations seemed to resemble lightweight agile practices, however, only partly.

> I guess I only remember the name of a one, which is the crisp, but I wouldn't say that I followed any one with intend but rather trying to have like the mindset within many of the frameworks that you expect each part of the whole development process to be iterative. – Interviewee 2

Moreover, as Interviewee 3 explained, the lack of project management models and the lack of teamwork caused uncertainty with developers and the project overall. The challenges occurred in the development phase and other project life cycle phases:

> I feel that it is up to you to decide [when to implement your work] because there is no teamwork, therefore others have nothing to say. I don't know if it's because of my own experience, but it's hard to trust that that product will work. - Interviewee 3

> EC11: AI experts seemed to have no understanding of the frameworks and the lack of them caused uncertainty for the development.

Frameworks were not wholly abandoned in every development case. Three interviewees out of eight worked in the university environment, two as a professor and one as a research assistant. They had a more experimental approach, as there was research done simultaneously with the development process. Moreover, the development was an opportunity to experiment both with the product and with the development practices.

> We have tested different auto ML systems or systems where you don't have to do anything yourself, you just feed your data to a system and decides itself what kind of things should be done, then we use various different deployment approaches. .. The data scientists experiment with whatever tools that the data scientists have been finding convenient. – Interviewee 6

It seemed that due to the lack of a guiding framework, many of the development actions were separated from each other. The fluidity between

development actions seemed to be challenging to achieve. This became more and more prevalent when asked about the testing process. Jacobsen et al. (2019) explain that part of the Agile development process is test cases that define the test inputs and expected results. Test ideas are captured and scripted to give an accurate description, what is tested. Furthermore, this process able the testing process to be automated after the test cases are defined. Test cases were necessary to practice in AI development, as the system was trained with test data. All of the interviewees mentioned some training phase in which test cases were used.

> You work with simple test cases at first and apply it into bigger junk of data then there's some pair review done by the other data scientist. – Interviewee 2

Continuous software engineering aims to make testing more effective by using automation and test cases (Fitzgerald, 2017). However, the automated testing process for AI is a less studied subject and not widely used. Only one interviewee said that they used the MLOps pipeline to automate the process.

> We use agile wherever possible, artificial intelligence and machine learning MLOps. ..The aim is to keep the level of test automation high, as it accelerates development. – Interviewee 7

Other interviewees said that they tested AI functions manually. The reasons for this varied. For instance, one interviewee said that their organization did not use automated testing tools, such as robotics, since tools like those were not available for AI development.

> It was not automated for sure. It was all manual. There was unit testing, then there was regression testing, also integration testing and all the things that you see. – Interviewee 5

> Testing is not done with testing robotics; those cannot be used in AI. The product is taught to manually work with the material in desired way. – Interviewee 4

> EC12: In most of the AI projects, testing is done manually.

Another interviewee said that since the developed concepts were so exotic, only those responsible for the development could understand them. In those cases, the person responsible for creating the AI concept was also responsible for testing.

> You work with simple test cases at first and apply it into bigger junk of data then there's some pair review done by the other data scientist in the product team but kinda depends how well that can work because if you are working with something that is pretty exotic then not even other data scientist don't know that much about it. – Interviewee 2

> EC13: Testing was often made by the same person that developed the functionality.

As the testing was mostly done manually and usually conducted by the same people responsible for the development, fourth primary empirical conclusion is:

PEC4: Automated testing is rarely used in the development of AI, due to lack of automatic testing tools for AI and exotic nature of the products.

Fitzgerald and Stol (2017) explain that continuous verification means formal methods and inspections when verifying the AI function throughout the development process. Three people out of eight being interviewed said they worked in projects or organizations in which projects were regulated. Interviewee 4 worked in a public organization that developed systems for the internal use of their organization. As the projects had many inner stakeholders and users, the group presented their process to others. Also, the interviewee explained that as the AI functionalities were used in tasks dealing with secure information, having formalities were necessary to secure quality and make the process presentable.

> We have described the process of how it goes, checkpoints and testing. Production testing and production use that are more closely monitored before can be accepted for actual use. The customer gives the ultimate blessing that now it works as desired. – Interviewee 4

> EC14: When developing an AI product for inner use, the development process is usually more seamless.

According to Jacobsen et al., (2019), one of the goals of Essentializing Agile is building a releasable product, and this seemed to be the goal of Interviewee 4's projects. However, some other interviewees mentioned that they did not constantly develop a complete AI product but rather an AI concept that the customer company would adapt to their systems and data. Therefore, the implementation into practice was not as important as aiming to produce a suitable concept. For example, Interviewee 5 worked with the projects that were mainly used for testing new AI-related ideas.

> I guess that the requirements mainly come from the companies themselves … projects have been much more complex because or complicated, because they have also had to try to figure out what the interesting problems are to solve. Whereas we kind of have the real-life problem that where we are going to where we are trying to help the best we can. – Interviewee 5

> EC15: When the AI experts were not responsible for the implementation, the development process was incomplete.

Whenever the AI experts were not responsible for the whole project lifecycle, there was less continuity in the process. For example, some interviewees explained that they only developed functionality or a concept, and the responsibility of implementation and monitoring was on a customer. Moreover, the interviewees seemed careless about what happened to the project after the

development role ended. Also, the continuous quality of the product seemed irrelevant to them after the customer took over. Thus, actions such as continuous verification and continuous compliance seemed irrelevant.

> We could say that there are separate projects: one with coming up with the good model or the AI thingy and then a separate one with bringing it to the production. - Interviewee 3

PEC5: In AI development projects, project participants did not have fluid roles, but they their own are of responsibility from which they rarely deviated.

6.5 Operations

The operations part of the interview discussed topis about what happens after the product deployment. Also, the usage and monitoring of the product are discussed and the relationship with the customer. The theme consisted of questions about user interaction, user expectations, monitoring, and eventual departure from the project.

Continuous use differs from an initial adoption versus, and the already gathered customers can be seen as more effective than trying to attract ones (Fitzgerald, 2017). To have continuously used software products, the one producing the software need to understand if the user expectations are fulfilled. However, it seemed that the understanding of the users was not well established. Four interviewees out of eight said their relationship with the users was usually indirect: The development projects were done for the customer company that implemented the AI concept or the product to their systems. Therefore, the users were stakeholders of the customer company. Thus, the customer company representatives were the ones with the AI experts and interacting with the users.

> We get the feedback from the company. But since the AI model is usually trained with the company own data, we don't know to what extent our dataset was biased or faulty or something else. – Interviewee 6

In most cases, the completed product was delivered to the customer organization, which implemented it to their processes without help from the development organization. However, receiving feedback from the users is possible. Most interviewees explained that the customer organization gathered feedback from the users and redirected it to the AI developers. In other words, the developers did not have direct channels for achieving feedback. Also, one interviewee explained that such feedback was only directed to them if there was something "great" or "terribly wrong" with the product. Also, the relationship was depended on the fact that specific customers had a contract with the developers that ensured direct and active feedback channels and closer monitoring.

> So first of all, it depends on who is who's your actual end user. Second of all, it depends upon what was your contract about. If your contract just said that you have to deliver it, and maintain it only sometime, then you probably don't get to hear too much. – Interviewee 6

> To say that [interaction with the users] it is rare, a privilege. I have made a specific use for the product; it has been really great to get to see it in context. In general, we are dealing with that in a product owner role.

As Fitzgerald & Stol (2017) said, understanding user expectations and customer awareness is essential in creating fulfilling products. As mentioned previously, four interviewees said that the relationship was indirect, and they received indirect feedback on the product. However, people out of the 8 said that there is no such interactive relationship. Significantly, the people responsible solely for developing the AI functions and concepts did not have management work tasks and did not have much interaction with the customer or the users:

> These goes beyond what I do [that is] the concept and model systems, so no. And when it [the product] is integrated in the customer's system, they are the ones that may or may not interact with their customers. – Interviewee 1

> EC16: AI developers have rarely a direct relationship with the product users.

> EC17: AI developers' role rarely included interaction with the customer or with the users.

Some people even said that having a relationship with the users was something they did not want to have. This seemed to be especially prevalent with the hands-on AI developers.

> I guess some people like it and it can give some valuable feedback for the developers, but I did not appreciate it in the past. - Interviewee 2

> EC18: AI experts found receiving feedback bothersome.

There was only one interviewee who worked with in-house AI projects, in which the product would be used in the same organization as the developers worked in. The product was used both by inner users that worked in the same organization and outer users that were outer stakeholders. Inner users were in regular interaction with the users, and feedback was regularly gathered from the outer users to ensure the quality of the AI product. Also, in this organization, the roles of the development team were more divided, as there were certain people in management roles, developers, testers, and own unit for innovations. Thus, the feedback was directed for the right people quickly, and the changes accordingly were easy to make.

> Production testing and production use that are more closely monitored before can be accepted for actual use. The customer give the ultimate blessing that now it works as

> desired. All the required elements are implemented, and the business makes the decision. The decision is made together with the client, that is usually the business unit, and they have the biggest word. IT unit is also responsible for making its own decision. We work closely together, the product is monitored together and also further developed together. – Interviewee 4

As Fitzgerald and Stol (2017) explain, the separations between software design time and the run-time have blurred. Thus, continuous run-time monitoring means detecting problems early by using technologies such as the cloud. Most of the interviewees said that monitoring of the released product varied. Interviewee 1 said there was usually a short crisis fixing period after the AI product implementation when the developers were responsible for problem detection. After the period, the customer took care of the monitoring.

> Well, there is a short price/crisis period during which I can still provide help if still needed but it is the customers job to deal with the rest. – Interviewee 1

> EC19: Monitoring was not automatically done by the AI developers, and was usually responsible of the customer.

As mentioned, the relationship between AI developers and customers was primarily contract-based, and the active communication would end after the product was delivered. In addition, none of the interviewees mentioned trust as something that they tried actively to establish. However, the developers seemed to understand the advantages that a good relationship with the customer would bring. For example, interviewee 7 said that even if the relationship were not necessarily active between projects, a well-established relationship during the project development meant that future projects could have had been suggested openly.

> So, I there are less opportunities to do something completely innovative once you've released the project or the product. But if new opportunities arise in the sense, if it is a long-term relationship with the client, and the new if you're continuously working on something bigger, then of course you have the possibility of improving or actually innovating or completely replacing something that you've all delivered a couple of years ago. – Interviewee 7

The biggest challenges regarding operations seemed to be the lack of interaction between AI experts and the product users. Moreover, it seemed that many AI experts that worked in the development did not interact with the customer as well, as the was usually a person in the team responsible for customer relationships. In addition, the contracts determined what kind of operational activities the AI experts had. In many cases, the AI experts seemed inactive and happy if they did not need to interact with the outer stakeholders after project deployment. This forms the sixth primary empirical conclusion of the study:

> PEC6: The lack of user and customer interaction causes the difficulty for AI experts to ensure that the product can be continuously used.

6.6 Improvement and innovation

One of the interviewees worked in an organization where the AI projects were developed for internal usage. Therefore, the stakeholders and product users worked in the same organization as developers, and the participants could easily communicate. Moreover, getting direct feedback was an essential and valued part of the development. As a result, the AI development unit worked closely with other units in the organization, and the product was created together:

> We work closely together; the product is monitored together and further developed together. – Interviewee 4

Interviewee 4 explained that the products had two types of users: inner and outer users. Inner users worked in the same organization, and the outer users were customers. The relationship with inner stakeholders and users was close, as their feedback was necessary for the development and error detection. Furthermore, Interviewee 4 explained that feedback was also gathered and prioritized through ticketing systems and other error monitoring systems. Outer users were also able to give feedback by giving customer feedback directed to AI developers. Interviewee 4 was the only person out of the eight interviewed that had a close and continuous relationship with the stakeholders and inner product users. In interviewee 4, feedback was the catalyst for improving the product, and the project participants recognized its value for the overall quality.

> Through ticketing, the process starts: CM card, where the message comes to the developers. An additional feature or change can be easily added, we also have an innovation department... Today, it is a must to be on the crest of the wave. – Interviewee 4

According to Jacobsen et al. (2015), fast feedback loops are an essential part of essentializing agile processes. Feedback is used to guide the development process towards the most fitting solution and is gathered as early as possible, as much as possible. However, in other interview cases, AI products, functionalities, and concepts were produced in a customer relationship, which would eventually end. The customer was not part of the development organization but came from the outside. Thus, the relationship started when the product life cycle started. Four people out of the eight said that after the ordered AI product or concept was delivered, the relationship with the customer would end, and thus, the product was in the customer's hands. Therefore, further improvement of the AI was not made, as the product would learn from the customer's data. However, suggestions for improvement were not wholly unwelcome. Four interviewees out of eight said that the contract determined the possibility for improvement: If the

customer was willing to provide more funding and recognized the need, improvements were made.

> I would say that if they are ready to provide the funding for the next project then of course I would be continuing if that happens. – Interviewee 1

Two interviewees out of eight said that the contract might have had included a fixed monitoring period, in which bug fixes and some minor improvements were made. However, this fixed period did not include implementing new functionalities or crafting new ideas. If the resources were provided and the customer wanted to develop the product further, the improvements started as a new project.

> I would say that if they are ready to provide the funding for the next project then of course I would be continuing if that happens. – Interviewee 1

> There are no plans for future improvements unless someone offers money to develop. – Interviewee 3

Three interviewees out of eight said that the improvements were not automatically thought of or welcomed to the project. Interviewee 7 said that due to the contract-based nature of the project, there was no chance to start to innovate or improve spontaneously. As the contracts usually determined the project's scope, the changes were something that needed to be discussed with the customer.

> In the project house where every working hour must be recorded somewhere. In your own product development projects, the thing is quite different – Interviewee 7

Emerging AI-based technologies and their business strategies are not well-studied subject matter. The uncertainty might be why the innovations are not always welcomed, as the value they bring can be unpredictable. Furthermore, as noted in previous data, the communication between customers and AI developers can be difficult and sometimes cause misunderstandings. AI innovations also include many inner risks since AI developers work independently, and many interviewees said that communication with other project participants was not active. Furthermore, as some parts of the AI development were made by the same person, there is a high risk for problems or failure.

> It is up to the customers to decide whether they want to keep using the product. – Interviewee 1

> EC20: In contract based projects, the possible ideas of improvement needed to be negotiated with the client, who made the decision.

Three other interviewees explained that the improvements were suggested, but the customer ultimately decided to develop them further. Interviewee 8 said that if the AI experts noticed minor changes, they were proposed quickly. However, if innovations required more changes to the project scope, the changes required more planning and were riskier. As explained earlier, adding more considerable innovations can be seen as riskier, and therefore, they were not as actively suggested.

> If it is like, smaller thing that brings lots of value, then probably yes. So, there is also that, like, amount of work needed to bring that innovation into the product. Yeah. But of course, if it does not affect the project scope much anyway, then it is possible to do. – Interviewee 8

> EC21: Smaller improvements were more easily added than bigger ones.

Even if there seemed to be some prejudice regarding implementing new ideas, the four interviewees out of eight said that an innovative mindset was an essential part of their work. As the changes in the technological environment happen constantly, the openness for learning new and bringing new ideas to the production was thought to be necessary. It seemed that the interviewees were interested in new ideas and innovation in the field of AI and were willing to learn something new, even if the innovation were not straight up used in customer projects.

> An additional feature or change can be easily added. Today, it is a must to be on the crest of the wave. – Interviewee 4

> Yes, of course this [innovation] is the bread and butter because we are a research facility. In the end, we are supposed to deliver new kinds of solutions and try out the new technologies. – Interviewee 5

Especially the interviewees that worked in the research-focused environment or as AI consultants were the ones that thought that an innovative mindset was vital for understanding the business environment and staying ahead of the competitors. Also, they were the ones that suggested innovations for their clients, as the experimental approach was part of the problem-solving. However, experimenting with something new was still the customer's decision, and a more traditional innovative approach was taken when needed.

> There are less opportunities to do something completely innovative once you have released the project or the product. But if new opportunities arise in that sense, but I already told you if it is a long-term relationship with the client, and the new if you are continuously working on something bigger, then of course you have the possibility of improving or innovating or completely replacing something that you have all delivered a couple of years ago. Yep, you always do you always try to suggest something new. You always propose new things. Now, if it is accepted or not, that is a completely different story. So that depends upon the budget and the relationships. – Interviewee 6

It seems that an innovative attitude was welcomed if the development organization had a mindset to be at the wave's crest. However, some interviewees explained that most of the time, the contracts were such that they were no room for further evolution. In some cases, the customer was solely responsible for the product after the implementation, and thus, the producer-client relationship ended for the moment. The biggest challenge for adapting a continuous innovation mindset seemed to be the lack of communication between AI developers and their customers. As mentioned earlier, Interviewee 4 said that their organization valued feedback and used it in development. However, as mentioned in previous Chapter 6.5, some interviewees did not appreciate feedback from the customer. In addition, it was stated that there was usually a person in charge of the communication with the customer. If this person lacked the understanding of AI possibilities and what could be improved, there could have had been some changes for the unused development. This forms the last primary empirical conclusion:

> PEC7: New ideas or innovations were not automatically added to the AI project, as this depended on the customer's wishes and the contract.

6.7 Summary

Chapter 6 included the analysis of empirical data and the seven primary empirical conclusions formed from it. Altogether, 21 empirical conclusions and seven primary empirical conclusions were drawn from the data. The challenging elements of the adoption of continuous software engineering were identified within the data as the empirical conclusion. To fit continuous software engineering practices into the development of AI, the research used the agile essentializing toolbox to understand the basic requirements for using the agile framework in the development context. These are the remarks that form the primary empirical conclusions.

TABLE 4 Empirical conclusions formed from the data

Identifier	Empirical conclusion
EC1	Continuous compliance and continuous security were not present within the data
EC2	Other than using Python as a main coding language, the practical tools of development varied greatly
EC3	Frameworks are known but mostly used partly or as an unidentified mental guideline.
EC4	The communication between AI developers and project participants is mostly unformal.
EC5	Lack of clear communication between AI developers and project participants causes problems with understanding the work efforts or the project as whole

EC6	AI developers did not usually take a part in the allocating the resources.
EC7	AI developers can make suggestions if there is a need for budget changes, but with the contract-based projects, the customer makes the ultimate decision
EC8	Customers did not have a clear understanding of AI functionalities.
EC9	AI developers and customers did not have a clear dialogue when it came to product development.
EC10	Developers did not have a clear understanding of their role as a project team member.
EC11	AI experts seemed to have no understanding of the frameworks and the lack of them caused uncertainty for the development.
EC12	In most of the AI projects, testing is done manually.
EC13	Testing was often made by the same person that developed the functionality
EC14	When developing an AI product for inner use, the development process is usually more seamless.
EC15	When the AI experts were not responsible for the implementation, the development process was incomplete.
EC16	AI developers have rarely a direct relationship with the product users.
EC17	AI developers' role rarely included interaction with the customer or with users.
EC18	AI experts found receiving feedback bothersome.
EC19	Monitoring was not automatically done by the AI developers and was usually responsible of the customer.
EC20	In contract-based projects, the possible ideas of improvement needed to be negotiated with the client, who made the decision.
EC21	Smaller improvements were more easily added than bigger ones.

The seven primary empirical conclusions are based on the empirical evidence presented above. They form the foundation for discussion in the following chapters.

TABLE 5 Primary empirical conclusions formed from the data

Identifier	**Primary empirical conclusion**
PEC1	Continuous compliance and continuous security were not present within the data.
PEC2	Frameworks offer support for AI project development, but they are not used systematically or accurately in the process.

PEC3	Due to lack of active communication between AI experts and other project participants, the AI experts often work in a silo. Thus, they do not participate business and strategy related activities as actively as other project participants.
PEC4	Automated testing is rarely used in the development of AI, due to lack of automatic testing tools for AI and exotic nature of the products.
PEC5	In AI development projects, project participants did not have fluid roles, but they their own are of responsibility from which they rarely divided.
PEC6	The lack of user and customer interaction causes the difficulty for AI experts to ensure that the product can be continuously used.
PEC7	New ideas or innovations were not automatically added to the AI project, as this depended on the customer's wishes and the contract.

For clarification, context enriched PECs are presented in Table 6.

TABLE 6 Context-enriched conclusions

Identifier	**Context-enriched conclusion**
PEC1	Regulatory compliance standards or security regulations were not brought up by the interviewees developing AI.
PEC2	AI experts have a basic understanding of different software engineering frameworks, but the usage of frameworks is not common.
PEC3	AI experts do not participate business and strategy related activities as they usually work independently with the AI function and are not keenly seeking a collaboration.
PEC4	AI is mostly tested manually by its developer, due to nature and the lack of automated testing technologies available.
PEC5	AI experts rarely divide from their work role or actively seek new responsibilities.
PEC6	AI experts rarely interact with the product users directly, and do not get information about if the product fulfils the user expectations.
PEC7	The customer provided the resources for the improvements and thus determined if new ideas or innovations were welcomed in the AI project.

7 DISCUSSION

In this chapter, the concepts analyzed in the previous chapter are connected to the theoretical background of this study, and the practical and theoretical implications are discussed.

7.1 Practical implications

The study aimed to highlight the challenges that AI development may face if they adopt continuous software engineering methodology. Empirical evidence suggests that each continuous phase of the project life cycle includes aspects that make their adoption challenging in the AI development environment. Therefore, continuous software engineering provides a set of practices for the continuous delivery pipeline (Fitzgerald & Stol, 2017).

As stated in PEC1, continuous compliance and continuous security were not identified in the research data. The lack of security mentions could be since many of the AI experts interviewed were responsible for developing the AI product and the concept. As AI function was usually part of the more extensive software product, the development organization may have a dedicated department for the security actions. Also, some interviewees explained that they usually developed only the basic idea of the AI function, and the customer was responsible for the implementation and thus security. Both cases make continuous security challenging to achieve. Continuous compliance was also not identified in the research data, and this might be because compliances differ in each project, and the discussion was about AI project development overall.

As PEC2 suggests, frameworks were rarely used in AI projects, even if AI experts understood knew their content. As mentioned, AI development lacks various frameworks as software development (Sweeney, 2003). Moreover, fitting a software engineering framework to an AI environment does not come without challenges, as the development processes and the requirements differ significantly. It seemed that lack of guiding frameworks caused communication problems, as it created gaps between project life-cycle phases and people responsible for different actions. This also caused problems with understanding the role of the AI expert. The following figure presents the AI development process explained by several interviewees. The processes resemble a waterfall model, as it lacks iterations and progresses systematically.

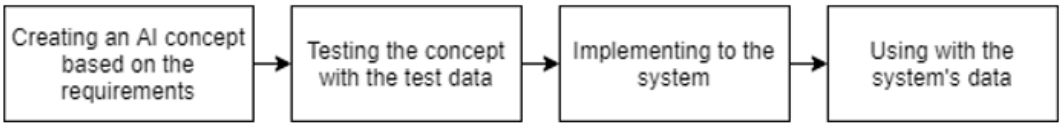

FIGURE 10 Simplifyed AI development process based on the description of the interviewees

PEC3 implied that the AI experts rarely participated in business and strategy-related actions. Interviewees explained that there was usually a senior team member responsible for the management and customer relationship actions. Also, interviewees lacked interest in such actions, especially communication with the customers. PEC2 seemed to be closely connected to PEC5 and PEC6, as the AI experts' roles, communication issues, and lack of interaction with other project participants were typical. As the AI experts did not have a good understanding of the overall project, were not able to successfully communicate with stakeholders, and did not interact with the users, the project outcome was unpredictable, at least for them. Also, the fulfillment of user expectations was unknown for them, as they received feedback only in exceptional cases.

From a practical standpoint, the challenging factors for adopting continuous development were caused by the exotic nature of AI, which further caused communication and collaboration issues and problems with the continuity and flexibility with the development. Also, it seemed that as only a few AI experts were working in development teams, they were usually responsible for several development actions, making the product prone to human errors.

TABLE 7 Practical implications of primary conclusions

Identifier	**Implication for practice**
PEC1	Regulatory compliance standards or security regulations were not brought up by the interviewees developing AI.
PEC2 & PEC3	AI experts have a basic understanding of different software engineering frameworks, but the usage of frameworks is not common.
PEC3	AI experts do not participate business and strategy related activities as they usually work independently with the AI function and are not keenly seeking a collaboration.
PEC4	AI is mostly tested manually by its developer, due to nature and the lack of automated testing technologies available.
PEC5	AI experts rarely dived from their work role or actively seek new responsibilities.

PEC6	AI experts rarely interact with the product users directly, and do not get information about if the product fulfils the user expectations.
PEC7	The budget, and the relationship with the customer were the main aspects that determined if new ideas or innovations are welcomed in AI projects.

7.2 Theoretical implications

The study aimed to locate and understand the challenges associated with the development of artificial intelligence using continuous methods. Empirical evidence strongly suggests that little research has been done on the challenges of developing artificial intelligence and coordinating continuous methods, and some empirical evidence is strongly interlinked. The following table presents a primary empirical conclusion and its relation to existing research.

TABLE 8 Primary empirical conclusions and their relation to existing research

Identifier	**Primary empirical conclusion**	**Relation to existing research**
PEC1	Continuous compliance and continuous security were not present within the data.	Contradicting, continuous compliance and continuous security are part of the operation actions of continuous software engineering (Fitzgerald & Stol, 2017). Continuous software engineering pipeline is not completed without these practises.
PEC2	Frameworks offer support for AI project development, but they are not used systematically or accurately in the process.	Corresponding with the previous research, as AI development lacks common aims and duties (Lee, et. al., 2019)
PEC3	Due to lack of active communication between AI experts and other project participants, the AI experts often work in a silo. Thus, they do not participate business and strategy related activities as actively as other project participants.	Corresponding with the previous research (Piorkowski, et al. 2021)

PEC4	Automated testing is rarely used in the development of AI, due to lack of automatic testing tools for AI and exotic nature of the products.	Contradicting, as the AI-Ops and MLOps pipeline has recently emerged to tackle problems regarding ML test pipeline (Fursin, Guillou, & Essayan, 2020; Karamitsos, Albarhami, & Apostolopoulus, 2020; Mäkinen, Skogström, Laaksonen & Mikkonen, 2021).
PEC5	In AI development projects, project participants did not have fluid roles, but they their own are of responsibility from which they rarely divided.	Novel, previous research about flexibility of the role of AI experts was not identified.
PEC6	The lack of user and customer interaction causes the difficulty for AI experts to ensure that the product can be continuously used.	Novel, previous research about lack of user and customer interaction with AI projects was not identified.
PEC7	New ideas or innovations were not automatically added to the AI project, as this depended on the customer relationship and the contract.	Novel, previous research about the impact of the customer relationship in AI project was not identified.

With continuous software engineering practice, the product's entire life cycle is a continuous process, considered a simultaneous event between different development levels (Suomalainen, 2015). However, continuous software engineering methodology is a new development practice used primarily to develop traditional software and information systems. The adoption of continuous software engineering practices in the development of AI is not a well-studied topic. AI development differs from normal software development, as the goal is to build a learning system that predicts outcomes from the input data (Saravanan & Sujatha, 2018) since AI development has not previously been studied as a whole pipeline, the lack of research regarding continuous security and compliance in understandable.

AI development lacks similar evolved development methodologies and practices as normal software development (Sweeney, 2003). Even if continuous software engineering does not necessarily define any specific tools to be used in the development process, the continuous development loop can be challenging to achieve if the developers are free to select any tool they prefer. As PEC2 states, AI developers did not rely on pre-set development practices and had various tools for practical development; their development process can be difficult to translate for the stakeholders (Piorkowski et al., 2021). Also, the AI experts were often uncertain about the project management itself, making the silo between

them and the other project participants significant. This also linked PEC2 to PEC3, as the lack of frameworks and selected practices was one of the causes of communication issues that prevented the usage of agile and continuous methods in the development. Especially the interviewees, that worked in the development of AI concepts or coding did not feel connected to the business and planning side of the project. Thus, using a business-oriented attitude to development, in which the business actions are close to development ones, is impossible (Fitzgerald & Stol, 2017).

Continuous software engineering also suggests that the testing be highly automated to ensure a continuous development life cycle (Fitzgerald & Stol, 2017). However, the test automation tools were not widely available with AI development, thus forcing the testing to be done manually. Even if there is research done with the MLOps and AIOps (Fursin, Guillou, & Essayan, 2020; Karamitsos, Albarhami, & Apostolopoulus, 2020; Mäkinen, Skogström, Laaksonen & Mikkonen, 2021), most of the AI experts did not mention them in the interviewees. The interviewees seemed to have two opinions about why continuous testing was not a possibility in AI development: Firstly, there was no such automated system to do the testing. Secondly, the concepts were so complex that they were only tested by the same person that developed them. Fitzgerald (2017) says that continuous testing is possible in two ways: automating testing with automated technologies or prioritizing test cases. However, according to interviewees, neither of these is available with current technologies. Only one interviewee explained that they used automated systems in testing; they were not in everyday use. The developers were solely responsible for testing the AI functionalities and rarely did discuss the testing with other project participants. This may cause a human error in the product and make the product unnecessarily complicated. It seemed that the interviewees did not have a complete understanding of how testing pipelines work, and thus, the testing was primarily manual. Therefore, PEC4 contradicts the current research about ML and AI testing pipelines.

Continuous software engineering aims to build a life cycle-long continuous pipeline for product development. PEC5 states that AI experts usually do not divide their roles. For example, an interviewee stated that customer relationship actions went beyond their work description. Thus, they did not participate in such. However, the continuity of the project life cycle could be challenging to achieve, as the AI experts were not interested in other project development phases than the one that they were responsible for. Piorkowski et al. (2021) have studied the communication issues between AI developers and other project participants. They explain that the gaps are caused due to not sharing the same status in knowledge, no trust, stakeholder expectations are not managed, and the communication participants do not share the mental model lens. In the interviews, some explained that they found customers annoying and discovering common ground with others difficult, so they did not feel motivated to participate in other development actions. In addition, the customers did not always understand the product or what they wanted. Three interviewees explained that their teams consisted of people responsible for customer relationships and

understanding their mindset. Therefore, the AI developers did not contact the customer or the end users, as the team's contact person gave the feedback. However, the agile mindset states that there should be no area in the project that is a "no-go" zone, as teams should be cross-functional. (Schawaber, 1996). The existence of such areas and the communication issues meant that the AI experts were not motivated to switch their responsibilities in the project fluidly.

The most prevalent problems seemed to occur with the customer relationship and the user-customer and the customer-developer interaction. Some interviewees explained that the product responsibility of the developers ended during implementation to the customer's system. Thus, the customer was the one that interacted with the product users. However, the customer was not necessarily the actual user. As AI was implemented to the customer's systems, they were in contact with the end-users. Also, as the PEC6 informs, AI experts' role did not usually include customer relationship actions, which was the responsibility of the project manager or product owner. The distant relationship between the AI developers, the customers, and the users meant that AI experts received feedback only on special occasions. There was no research identified about the user and the customer relationship problems with the AI experts. However, the lack of outcome feedback has been studied in general software development as part of agile development. It may cause a rise in costs, a longer decision-making process, and seeing development results (Highsmith & Cockburn, 2001).

Software products are rarely a one-time purchase, as the products need to work in the rapidly changing business environment (O'Connor, Elger, & Clarke, 2017). Therefore, agile methods suggest that the product lifecycle does not end with the implementation, but the product evolution continues after the initial project ends. In the research data, many of the interviewees explained that the contract determined improvements and evolution. It seemed that the most experimental mindsets were with the AI experts, that worked in an organization that did active research. As part of their work was to try new technologies and innovations, their attitude towards new things was welcoming. On the other hand, AI developers working with contract-based projects in which the improvement and evolution of the development were customer decisions. PEC7 states that improvement, innovations, and new ideas depended on customer relationships. Previous research did not identify this phenomenon in the AI context. However, agile methodologies aim to bring the project development team and the customer closer together (Muller & Tichy, 2001). As previously stated, there was usually a person responsible for the project management with other continuous software engineering actions. Also, they were the ones with a close relationship with the customer. As the AI expert's role did not usually include contact with the customer, they did not suggest improvements easily. Thus, suggesting new ideas was not expected.

8 THANK YOU AND GOODBYE

This chapter goes through the final conclusions for the study. These include the answer to research questions, limitations of the study and future research opportunities.

8.1 Answers to the research questions

The study's goal was to understand the challenges associated with the adoption of the continuous software engineering methodology in the development of artificial intelligence. Therefore, two additional research questions were introduced to clarify the topic to answer the main research question. The additional first research question of the study was:

- What is agile and continuous system development?

The research question was answered by reviewing the scientific literature and research articles on the topic. The answer to the research question aimed to emphasize the introduction of selected design models and the challenges they may cause. Fitzgerald and Stol (2017) had researched continuous software engineering, which was used to map out the continuous software engineering phases.

The second research question aimed to explain the nature of AI and its important development:

- What is artificial intelligence and how it is developed?

The research question was answered by reviewing the scientific literature and research articles on the topic. The answer to the research question aimed to give a general understanding of AI and its development. Also, the challenges that AI development may face are also included in the literature review.

As the goal of the study was to understand the challenges of continuous software engineering in AI context, The main research question of the study was:

- What are the challenges associated with the continuous development of artificial intelligence?

To combine continuous software engineering with AI development, the research model using agile essentializing tools was created. This formed the research model for the empirical research of the study. The empirical findings suggest that the adoption of continuous software engineering in the development of AI has many challenges caused by the nature of AI development. AI development is done more stiffly compared to agile software engineering, and AI experts

mainly worked independently. Communication issues caused by lack of shared knowledge, lack of guiding frameworks, and issues in the role of AI experts meant that the project life cycle did not resemble a continuous cycle but a step-by-step heavyweight development model. Furthermore, the AI experts rarely interacted with the customer or the product users, as they felt that their work role did not include such actions. Also, the customer was usually responsible for the AI product after it was implemented in their systems, thus interacting with the system users.

8.2 Limitations

The thematic interview method was used to gather research data. This gave room for improvising and talking about topics that they found the most important. However, as the interviewee's background varied greatly, the interviewees concentrated on those questions the most that they were the most familiar with. On the opposite, their unfamiliar topics were skipped mainly by them, as the interviewees' work roles did not include all the product life cycle actions. For example, only one person worked in a management role and thus had the most insight into the business actions but did not have deep knowledge about the development tasks. This made the answers diverse, and some variables were more emphasized. However, the deep knowledge of understanding one phase of the life cycle, but not the others, was seen as a piece of evidence for the silos in which the AI team members worked.

Artificial intelligence as a topic was studied on high-level. The AI technologies were not strictly divided into smaller groups, and all the projects were viewed through the same lens. The more precise separation of technologies would have required a more precise selection of the interviewees and the scope of the study to be different. This high-level approach was adequate for the study.

While the study gave suggestions of the possible challenges in adopting continuous software engineering in the development of AI, the methodology is yet to be tested. Furthermore, the lack of real-life scenarios using continuous software engineering in AI means that the continuous process may include other challenges not discussed in this study.

8.3 Further research

As mentioned, the research did not divide the interviewees into groups based on the AI technologies they were working with. In addition, the size of the development organization or the projects was not considered. However, as not all AI functionalities are not produced similarly, separating different projects from each other is suggested. Also, scaling continuous software engineering practices up or

down can cause additional challenges that should be considered with further research.

Even if the study provides information about some challenges regarding the continuous engineering of AI, the possible links between the challenges and the disadvantages they cause to different project lifecycle phases are yet to be studied. Giardino et al. (2015) have presented the Greenfield Startup Model, which explains the priority of start-ups to release the product as quickly as possible. However, the need to shorten time-to-market by speeding up the development through low-precision activities is counterbalanced by the need to restructure the product before targeting further growth.

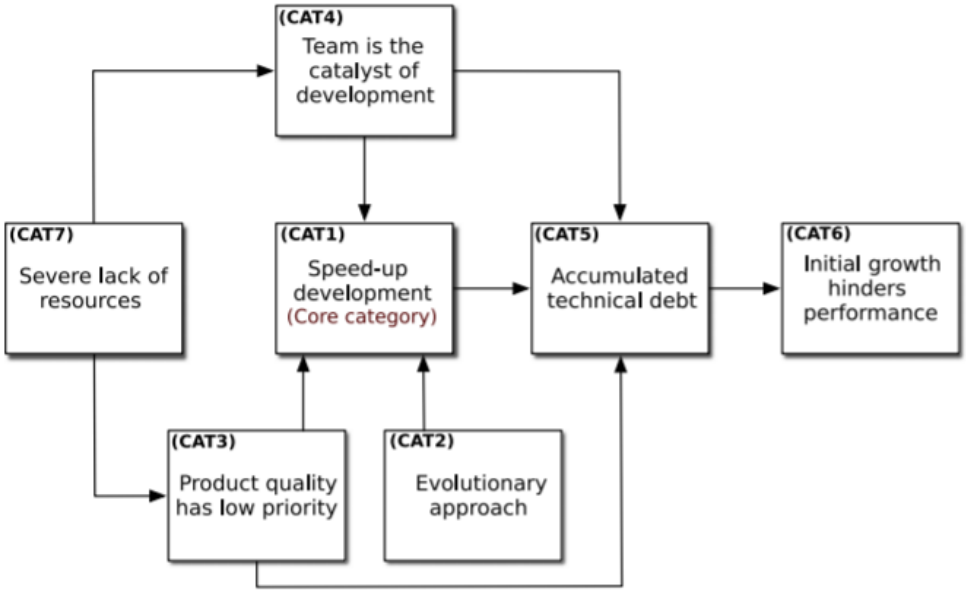

FIGURE 11 Main categories and causal relationships in the Greenfield Startup Model (Giardino, et al., 2015)

A similar phenomenon was noticed in this study, as the AI experts seemed to speed up the development to push the product forward. As they rarely collaborated with other project participants or received feedback, the end quality of the product and the customer fulfilment did not seem to be a high priority. In further research, the challenging factors and continuous software development and the linkages could be studied using a model based on the Greenfield Startup Model to ensure the quality of AI products.

REFERENCES

Akbar, M. A., Sang, J., Khan, A. A., Amin, F. E., Hussain, S., Sohail, M. K., ... & Cai, B. (2018). Statistical analysis of the effects of heavyweight and lightweight methodologies on the six-pointed star model. IEEE Access, 6, 8066-8079.

Beck, K. (1999). Embracing change with extreme programming. Computer, 32(10), 70-77.

Blokdijk A, Blokdijk R (1987) Planning and Design of Information Systems. London: Academic Press.

Bresina, J., Dearden, R., Meuleau, N., Ramkrishnan, S., Smith, D., & Washington, R. (2012). Planning under continuous time and resource uncertainty: A challenge for AI. arXiv preprint arXiv:1301.0559.

Bostrom, N. (2017). Strategic implications of openness in AI development. Global policy, 8(2), 135-148.

Bostrom, N. (2014). Superintelligence: Paths, Dangers, Strategies. Oxford University Press.

Cotterell M., & Hughes, B., (1995) Software Project Management, An international Thomson Publishing Company, London

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. Management science, 35(8), 982-1003.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. Ieee Software, 33(3), 94-100.

Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176-189.

Fojtik, R. (2011). Extreme Programming in development of specific software. Procedia Computer Science, 3, 1464-1468.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. Software Development, 9(8), 28-35.

ft n il Ionel, N. (2008). Critical analysys of the Scrum project management methodology.

Fursin, G., Guillou, H., & Essayan, N. (2020). CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking. arXiv preprint arXiv:2001.07935.

Greenfield, J., & Short, K. (2003, October). Software factories: assembling applications with patterns, models, frameworks and tools. In Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (pp. 16-27).

Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., & Abrahamsson, P. (2015). Software development in startup companies: the greenfield startup model. IEEE Transactions on Software Engineering, 42(6), 585-604.

Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. California management review, 61(4), 5-14.

Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., ... & Ramkumar, P. N. (2020). Machine learning and artificial intelligence: Definitions, applications, and future directions. Current reviews in musculoskeletal medicine, 13(1), 69-76.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. Computer, 34(9), 120-127.

Hope, J., & Fraser, R. (2003). Beyond budgeting: how managers can break free from the annual performance trap. Harvard Business Press.

Hüttermann, M. (2012). DevOps for developers. Apress.

Jacobson, I., Ng, P. W., McMahon, P. E., & Goedicke, M. (2019). The essentials of modern software engineering: free the practices from the method prisons!. Morgan & Claypool.

Laanti, M. (2014, May). Characteristics and principles of scaled agile. In International Conference on Agile Software Development (pp. 9-20). Springer, Cham.

Larman, C. (2004). Agile and iterative development: a manager's guide. Addison-Wesley Professional.

Lee, J., Suh, T., Roy, D., & Baucus, M. (2019). Emerging technology and business model innovation: the case of artificial intelligence. Journal of Open Innovation: Technology, Market, and Complexity, 5(3), 44.

Leffingwell, D. (2018). SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises. Addison-Wesley Professional.

Legg, S., & Hutter, M. (2007). A collection of definitions of intelligence. Frontiers in Artificial Intelligence and applications, 157, 17.

Liddy, E. D. (2001). Natural language processing.

Lyytinen, K. (1987). Different perspectives on information systems: problems and solutions. ACM Computing Surveys (CSUR), 19(1), 5-46.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2015, May). Dimensions of devops. In International conference on agile software development (pp. 212-217). Springer, Cham.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016, November). Relationship of DevOps to agile, lean and continuous deployment. In International

conference on product-focused software process improvement (pp. 399-415). Springer, Cham

McMillan, A. B. (2020). Making Your AI Smarter: Continuous Learning Artificial Intelligence for Radiology.

Minsky, M. (2019). A framework for representing knowledge (pp. 1-25). de Gruyter.

Mohammadi, S., Nikkhahan, B., & Sohrabi, S. (2009). Challenges of user Involvement in Extreme Programming projects. International Journal of Software Engineering and Its Applications, 3(1), 19-32.

Mäkinen, S., Skogström, H., Laaksonen, E., & Mikkonen, T. (2021). Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?. arXiv preprint arXiv:2103.08942.

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. Journal of the American Medical Informatics Association, 18(5), 544-551.

O'Connor, R. V., Elger, P., & Clarke, P. M. (2017). Continuous software engineering—A microservices architecture perspective. Journal of Software: Evolution and Process, 29(11), e1866.

Ohno, T. (1988). Toyota production system beyond large-scale production. Diamond Inc

Petersen, K., Wohlin, C., & Baca, D. (2009, June). The waterfall model in large-scale development. In International Conference on Product-Focused Software Process Improvement (pp. 386-400). Springer, Berlin, Heidelberg.

Radack, S. (2009). The system development life cycle (sdlc) (No. ITL Bulletin April 2009 (Withdrawn)). National Institute of Standards and Technology.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016, November). DevOps adoption benefits and challenges in practice: a case study. In International conference on product-focused software process improvement (pp. 590-597). Springer, Cham.

Piorkowski, D., Park, S., Wang, A. Y., Wang, D., Muller, M., & Portnoy, F. (2021). How ai developers overcome communication challenges in a multidisciplinary team: A case study. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1), 1-25.

Poole, D. L., & Mackworth, A. K. (2010). Artificial Intelligence: foundations of computational agents. Cambridge University Press.

Pressman R.S. (1994). Software Engineering A practitionerís Approach, McGraw-Hill International, UK

Sánchez-Gordón, M., & Colomo-Palacios, R. (2018, October). Characterizing DevOps culture: a systematic literature review. In International

Conference on Software Process Improvement and Capability Determination (pp. 3-15). Springer, Cham.

Senapathi, M., Buchan, J., & Osman, H. (2018, June). DevOps capabilities, practices, and challenges: insights from a case study. In Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018 (pp. 57-67).

Schwaber K. 1995. SCRUM Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation.

Schwaber K. 2004. Agile Project Management With Scrum. Washington: Microsoft Press.

Schwaber K. & Beedle M. 2002. Agile Software Development with Scrum. New Jersey: Prentice-Hall.

Suomalainen, T. (2015, December). Defining continuous planning through a multiple-case study. In International Conference on Product-Focused Software Process Improvement (pp. 288-294). Springer, Cham.

Smeds, J., Nybom, K., & Porres, I. (2015, May). DevOps: a definition and perceived adoption impediments. In International conference on agile software development (pp. 166-177). Springer, Cham.

Srinivasan, K., & Fisher, D. (1995). Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering, 21(2), 126-137.

Sweeney, L. (2003). That's AI?: a history and critique of the field.

Turetken, O., Stojanov, I., & Trienekens, J. J. (2017). Assessing the adoption level of scaled agile development: a maturity model for Scaled Agile Framework. Journal of Software: Evolution and process, 29(6), e1796.

Vaishya, R., Javaid, M., Khan, I. H., & Haleem, A. (2020). Artificial Intelligence (AI) applications for COVID-19 pandemic. Diabetes & Metabolic Syndrome: Clinical Research & Reviews, 14(4), 337-339.

Valacich, J. S., George, J. F., & Hoffer, J. A. (2004). Essentials of systems analysis and design. Prentice Hall.

Vega, J., Murari, A., Pereira, A., Portas, A., Rattá, G. A., Castro, R., & JET-EFDA Contributors. (2009). Overview of intelligent data retrieval methods for waveforms and images in massive fusion databases. Fusion Engineering and Design, 84(7-11), 1916-1919.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. MIS quarterly, 425-478.

Welke, R. J. (1983). IS/DSS: DBMS support for information systems development. In Data Base Management: Theory and Applications (pp. 195-250). Springer, Dordrecht.

Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. IEEE computer, 36(6), 39-43.

Womack, J.P. & Jones, D.T., (1996). Lean Thinking: Banish waste and create weath in your corporation. New York: Free Press..

Womack, J. P., & Jones, D. T. (1997). Lean thinking—banish waste and create wealth in your corporation. Journal of the Operational Research Society, 48(11), 1148-1148.

Zhang, D., & Tsai, J. J. (2003). Machine learning and software engineering. Software Quality Journal, 11(2), 87-119.

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and its practices. IEEE Software, 33(3), 32-34.

APPENDIX 1

Themes and interview questions:

1. Theme: Current job and challenges
    a. What is your current work role and what does it includes?
    b. How is the work divided in your project group? How much collaborate with others?
    c. What kind of tools did you use when developing AI?
    d. Can you name any framework, model, or mindset, that you use as a development guideline?
2. Theme: Business Strategy
    a. Can you work independently in the project, or does your work require collaboration with other project participants?
    b. How are the requirements of the project decided? You can use a previous or current project as an example.
    c. How are the resources planned at the beginning of the project?
3. Theme: Development
    a. When is a new functionality or part of the code applied to larger project on hand?
    b. Can you describe the testing process in some of your projects?
    c. How you decide that the project is ready to be released?
    d. Do you know how does your part of the work affect the overall quality of the bigger project? For example, the quality of a software project.
4. Theme: Operations
    a. Do you interact with the product users after the release?
    b. After the product release, do you know if the user expectations are fulfilled?
    c. Is the product monitored after the release? If it is, how?
5. Theme: Improvement and innovation
    a. Is the product quality improved after the release by you?
    b. Are new innovations added to the product if new opportunities rise?
    c. When does your involvement with the project end?

Do you have anything that you like to add?