TU WIEN Informatics

Extending Scrum with DevOps Practices: Concept and Validation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Lorenz Graiff BSc
Matrikelnummer 01326363

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Wien, 13. August 2025

Unterschrift	Unterschrift Betreuung

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Extending Scrum with DevOps Practices: Concept and Validation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Lorenz Graiff BSc
Registration Number 01326363

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Vienna, 13th August, 2025

Signature | Signature Advisor

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Extending Scrum with DevOps Practices: Concept and Validation

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Lorenz Graiff BSc
Matrikelnummer 01326363

ausgeführt am
Institut für Information Systems Engineering
Forschungsbereich Business Informatics
Forschungsgruppe Industrielle Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Wien, 28. März 2025

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Lorenz Graiff BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. März 2025

BSc

Danksagung

Ich möchte meinen Betreuern Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig und DDI Dr. Raoul Vallon danken, dass sie sich dieser Arbeit angenommen haben und mir während des gesamten Forschungsprozesses kontinuierlich mit Unterstützung und fachlicher Anleitung zur Seite gestanden sind. Außerdem danke ich meinen Freund*innen und Kolleg*innen, die mir geholfen haben, mit DevOps-Expert*innen in Kontakt zu treten, um den Validierungsprozess dieser Arbeit zu unterstützen, als auch den Expert*innen selbst für ihre Zeit und Bereitschaft, an den Interviews teilzunehmen und ihre wertvollen Erkenntnisse und Erfahrungen zu teilen. Abschließend gilt mein besonderer Dank meiner Familie für ihre andauernde Unterstützung während meines gesamten Studiums.

Acknowledgements

I want to thank my advisors Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig and DDI Dr. Raoul Vallon for taking on this thesis and the continuous support and guidance throughout the research process. Furthermore, I want to thank my friends and colleagues for helping me get in touch with DevOps experts to help with the validation process of this thesis, as well as the DevOps experts for their time and willingness to participate in the interviews and share their valuable insights and expertise. Finally, I want to thank my family for their continuous support throughout my studies.

Kurzfassung

Seit seiner Einführung im Jahr 2008 wird der Begriff DevOps kontinuierlich weiterentwickelt und bisher wurde kein gemeinsames Verständnis darüber erreicht, was er genau beinhaltet. Existierende Forschung zeigt, dass trotz der häufigen Erwähnung von DevOps-Praktiken nur wenige Details zu deren konkreter Anwendung bereitgestellt werden. Zudem wird DevOps oft ohne ein geeignetes Framework für das Softwareprozessmanagement praktiziert. Diese Diplomarbeit untersucht diese Forschungslücke mit Hilfe einer systematischen Literaturrecherche, identifiziert die am häufigsten verwendeten DevOps-Praktiken im Kontext agiler Softwareentwicklung und liefert, auf Basis des Konsens aus existierender Forschung, Definitionen für diese Praktiken. Diese Praktiken werden anschließend genutzt, um einen üblichen Scrum-Prozess zu erweitern, wodurch DevOps mit einem geeigneten Framework für Softwareentwicklung kombiniert werden. Dabei werden sowohl für die Entwicklung wichtige Aspekte, wie Agilität und Offenheit gegenüber sich jederzeit ändernder Anforderungen, berücksichtigt, als auch für den Betrieb wesentliche Bereiche, wie die Stabilität der Anforderungen und die Integrität der ausgelieferten Anwendung. Dadurch wird das Risiko einer Kluft zwischen Entwicklung und Betrieb im Softwareentwicklungsprozess reduziert. Der entwickelte Prozess wird validiert, indem semi-strukturierte Interviews mit 8 Expert*innen aus verschiedenen Unternehmen und Branchen geführt werden, die unterschiedliche Rollen und Positionen innerhalb ihrer jeweiligen Teams einnehmen. Es wurde gezeigt, dass DevOps und Scrum sich sehr gut komplementieren. Es war nicht nur möglich, alle ausgewählten DevOps-Praktiken mit einem üblichen Scrum-Prozess zu verknüpfen, sondern es konnten auch viele Synergien zwischen den Methodologien beobachtet werden.

Keywords: *DevOps; Agile; Scrum; DevOps Praktiken; Softwareentwicklung; Qualitative Forschung*

Abstract

Ever since its introduction in 2008, the term DevOps is subject to continuous evolution and no common understanding of what it entails has been achieved yet. Existing research shows that despite DevOps practices being mentioned, few details of how to use them are presented and DevOps is often practiced without a proper framework for software process management. This thesis examines this gap by performing a systematic literature review of existing research, identifying the most frequently used DevOps practices for agile software development, and providing practice definitions based on the consensus of existing research. These practices are then used to extend a common Scrum process, thereby combining DevOps with a proper framework for software development. This process takes into account areas important for development, such as being agile and open to changes in requirements at any time, as well as areas being important for operations, such as stability of the requirements and the integrity of the deployed application. Doing so the risk of a gap between development and operations within the software engineering process is reduced. The developed process is validated, conducting semi-structured interviews with 8 experts from different companies and sectors, occupying different roles and positions within their respective teams. It was shown that DevOps and Agile complement each other. Not only was it possible to link all selected DevOps practices with a common Scrum process, but also could many synergies between the methodologies be observed.

Keywords: *DevOps; Agile; Scrum; DevOps Practices; Software Development; Qualitative Research*

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

In order to stay competitive, companies need to develop, deploy, and change their products fast and on a continuous basis.

Currently, there is a wide portfolio of frameworks and approaches that guide the development process of a software product, these solutions can be traditional or agile approaches. Traditional approaches seek to impose discipline on the software development process and thereby make it predictable and efficient. To achieve this, they are supported by a detailed process with an emphasis on planning. On the other hand, agile approaches are adaptive - not predictive - and oriented to people and not processes and generation of value from early stages, which makes a great difference compared to traditional frameworks [MD21].

However, those frameworks define their process elements from one of the two areas involved in the software development process, i.e. from the development area or the infrastructure area, which has created a kind of dividing wall between these two areas significantly affecting the development process of a software product from its conception to its final delivery [MHKB15]. The proposed frameworks define workflows, roles, and artifacts only from a particular perspective or area, which is often complementary to the objectives and / or needs of the other area involved, e.g. development areas implement practices to be more agile and be open to changes in requirements at any time, while the infrastructure area implements practices to control the stability of the requirements, since these directly affect the processes related to the enlistment of the technological infrastructure. This means that the main priority of the infrastructure area is to take care of the integrity of the deployed applications and based on this, the policies and processes are defined [GZCP20].

To tackle this problem, we need to change the way software development is frequently implemented, and DevOps shows us a way of doing so [KHD+16]. Within this thesis

the term DevOps is understood as defined by [Hü12]: "The term DevOps is a blend of development (representing software developers, including programmers, testers, and quality assurance personnel) and operations (representing the experts who put software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians)."

Applying DevOps practices, everyone within a team shares a common goal [KHD+16]. Developers care not only about implementing features, but also actively ensure their work flows smoothly and frequently through the entire value stream without causing chaos and disruption [KHD+16]. Simultaneously, quality assurance (QA), IT Operations, and information security (Infosec) are always working on ways to reduce friction for the team, creating the work systems that enable developers to be more productive and get better outcomes [KHD+16]. This enables organizations to create a safe system of work, where small teams are able to quickly and independently develop, test, and deploy code and add value quickly, safely, securely, and reliably to customers [KHD+16].

Given that DevOps is subject to continuous evolution ever since its introduction based on a discussion between Andrew Clay and Patrick Debois at the Agile Conference in Toronto in 2008, no common understanding of what it entails has been achieved yet [Hü12] [JbAPT16]. Despite it being often compared to agile software development, it differs as DevOps adds to agile software development by having an emphasis on automating the analysis of applications, i.e. continuously monitoring the performance of the system, presenting the results in automated dashboards, etc. and providing specific approaches and tools [JbAPT16]. However, existing research shows that despite practices being mentioned, few details of how to use them are presented [JbAPT16] and DevOps is often practiced without a proper framework for software process management [SP17]. This complicates the adoption of DevOps for companies working with widespread agile software development processes like Scrum. Most references in existing literature to such processes are due to them being agile processes, however, information on how to practically combine them with DevOps are limited.

1.2 Expected Results

Combining literature research of existing DevOps definitions, existing case studies of agile DevOps applications and conducted interviews with experts in adoption and usage of DevOps in companies, a concept will be developed on how to extend a Scrum process with DevOps practices.

Firstly, a systematic literature review of existing research (DevOps practices in agile software development) is performed in order to determine coverage of the research field and find a set of most frequently used DevOps practices for agile software development. Secondly, a common Scrum process is extended by said practices, thereby combining DevOps with a proper framework for software development, taking into account areas important for development, such as being agile and open to changes in requirements at any time, as well as areas being important for operations, such as stability of the

requirements and the integrity of the deployed application [GZCP20], reducing the risk of a gap between development and operations within the software engineering process [SP17]. Following the development of this new process, expert interviews are conducted, adding the practitioner's perspective to the validation process, following the suggestion made by Jabbari et al. [JbAPT16] and Toh et al. [TSM19b] and the process is further improved based on their feedback.

The process to be created within this thesis strives to help companies to adapt their working process towards enabling agile DevOps and achieving a competitive advantage in the marketplace.

To this end the following research questions are deducted:

RQ 1: Which DevOps practices are most frequently used in state-of-the-art research in the context of agile software development?

RQ 2: Which DevOps practices can be used to extend agile software development, by example of Scrum, in order to reduce the risk of development and operations working towards unaligned goals?

A detailed description on which research methods are used answering the respective question can be found in Chapter 3 (Research Methodology).

1.3 Structure

This thesis is structured as follows. Following the introduction in Chapter 1, Chapter 2 elaborates on the theoretical background by defining agile values, Scrum, as well as DevOps. Chapter 3 then gives an overview of the used methodology. Afterwards Chapter 4 describes the preparation and execution of the systematic literature review, answering the raised question of most frequently used DevOps practices in state-of-the-art research in the context of agile software development (RQ1). Chapter 5 (Process Definition) then uses these found practices in order to extend a common Scrum process to reduce the risk of development and operations working towards unaligned goals. This newly defined process is then evaluated in Chapter 6 (Process Validation) using expert interviews and adjusted according to their feedback. Finally, the results are discussed in Chapter 7 (Results) and the thesis is concluded in Chapter 8 (Conclusion).

1.4 Relevance for Business Informatics

DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops), emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices [JbAPT16]. In order for DevOps to be properly used within a company, either new processes need to be developed, or existing processes need to

be adapted. Such processes are perfect representations of information systems (IS) as they are socio-technical systems that include human and machine components [Wie21]. As business informatics is an engineering discipline, which mainly dedicates itself to analyzing, designing, implementing, and evaluating IS in business, public administration, and the private sphere [Wie21], designing and deploying DevOps within a company towards the implementation of corporate goals falls within the area of responsibility of someone working in Business Informatics.

1.5 Motivation

My motivation for this topic is a long-term interest in managing and supporting project and development teams. This is also the reason why I started working as an IT-consultant alongside my studies in order to gather experience in this field. Doing this I witnessed firsthand how inefficient processes can negatively impact projects and companies. I believe many of these problems could have been prevented using DevOps practices within the existing agile development setting. Therefore, I would like to help companies to avoid such problems in the future by guiding them in their transition towards DevOps.

CHAPTER 2

Theoretical Background

This chapter gives an introduction in the covert topics within this thesis. The introduced topics are agile value and their most popular representative Scrum, DevOps including a state-of-the-art analysis of existing DevOps research, as well as agile DevOps, a closer look at how agile and DevOps work together.

2.1 Agile Values

The manifesto for agile software development was created by seventeen practitioners in 2001 due to the need for an alternative to documentation driven, heavyweight software development processes [BBvB+01]. It identifies four key values and thereby reveals which items are considered valuable by agile software development methods (ASDMs). Table 2.1 shows that ASDMs prioritize 1) individuals and interactions over processes and tools, 2) working software over comprehensive documentation, 3) customer collaboration over contract negotiation and 4) responding to challenges over following a plan [BBvB+01] [Cho08]. "That is, while there is value in the items on the right, we value the items on the left more" [BBvB+01].

More Valuable Items	Less Valuable Items
Individuals and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to challenges	Following a plan

Table 2.1: Manifesto for agile software development [BBvB+01]

Furthermore, it identifies twelve principles that its authors believe software developers should use to guide their work. These principles are shown in Table 2.2.

#	**Principle**
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4	Business people and developers must work together daily throughout the project.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity–the art of maximizing the amount of work not done–is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Table 2.2: Principles behind the Agile Manifesto [Cho08]

Following these values and principles ASDMs have the potential to provide higher customer satisfaction, lower bug rates, shorter development cycles and quicker adaptation to rapidly changing business requirements [BT03] [ML05] [PSHH04].

2.1.1 Scrum

The most popular agile methodology, representative agile values, is Scrum. It is used by 63% of team-level agile users and has been the most popular methodology ever since the first yearly survey regarding agile in 2006, which results in the annual "State of Agile Report" [Dig24].

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems. In a nutshell, Scrum requires a

Scrum Master to foster an environment where [SS20]:

1. A Product Owner orders the work for a complex problem into a product backlog.
2. The Scrum team turns a selection of the work into an increment of value during a sprint.
3. The Scrum team and its stakeholders inspect the results and adjust for the next sprint.
4. Repeat

The Scrum process was developed by Schwaber and Sutherland [SS20]. The former developed and formalized the Scrum process for system development while he was at his company, Advanced Development methods (ADM), in the early 1990s [Cho08]. By a joint effort of both Schwaber and Sutherland, the Scrum process was first introduced to public at the conference of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) in 1996 [PSHH04]. In 2001 Schwaber and Beedle published the first book on Scrum, Agile Software Development with Scrum [SB01]. The same year Schwaber, Derby and Cohn founded the Scrum Alliance, a non-profit organization dedicated to nurturing the agile movement by providing education, advocacy, research, community and connection [Scr24a]. In 2006 Sutherland founded his own organization Scrum Inc to offer training and consulting services [Scr24b] and in 2009 Schwaber left the Scrum Alliance and founded Scrum.org to provide a variety of Scrum training[Scr24c]. In 2010 Schwaber and Sutherland wrote the first version of the Scrum Guide to help people worldwide understand Scrum. Ever since the guide has evolved through small, functional updates [SS20].

2.1.1.1 Theory and Values of Scrum

Scrum is founded on empiricism and lean thinking. Empiricism asserts that knowledge comes from experience and making decisions based on what is observed. Lean thinking reduces waste and focuses on the essentials [SS20].

Scrum employs an iterative, incremental approach to optimize predictability, control risk [SS20], manage and control complex software and product development [Cho08] and is an enhancement of preexisting iterative and incremental approaches to delivering objected-oriented software [PSHH04]. Scrum engages groups of people who collectively have all the skills and expertise to do the work and share or acquire such skills as needed [SS20].

Scrum combines four formal events for inspection and adaptation within a containing event, the sprint. These events work because they implement the empirical Scrum pillars of [SS20]:

- Transparency
  The emergent process and work must be visible to those performing the work as well as those receiving the work. With Scrum, important decisions are based on the perceived state of its three formal artifacts. Artifacts that have low transparency can lead to decisions that diminish value and increase risk.

  Transparency enables inspection. Inspection without transparency is misleading and wasteful.

- Inspection
  The Scrum artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems.

  Inspection enables adaptation. Inspection without adaptation is considered pointless. Scrum events are designed to provoke change.

- Adaptation
  If any aspects of a process deviate outside acceptable limits or if the resulting product is unacceptable, the process being applied or the materials being produced must be adjusted. The adjustment must be made as soon as possible to minimize further deviation.

  Adaptation becomes more difficult when the people involved are not empowered or self-managing. A Scrum team is expected to adapt the moment it learns anything new through inspection.

These pillars come to life by the Scrum team and the people they work with becoming more proficient in living and embodying the five key values of Scrum (Figure 2.1) [SS20]:

- Commitment
  The Scrum team commits to achieving its goals and to supporting each other.

- Focus
  Their primary focus is on the work of the sprint to make the best possible progress toward these goals.

- Openness
  The Scrum team and its stakeholders are open about the work and the challenges.

- Respect
  Scrum team members respect each other to be capable, independent people, and are respected as such by the people with whom they work.

- Courage
  The Scrum team members have the courage to do the right thing, to work on tough problems.

Figure 2.1: Scrum values [Scr16]

These values give direction to the Scrum team with regard to their work, actions and behavior. The decisions that are made, the steps taken and the way Scrum is used should reinforce these values, not diminish or undermine them [SS20].

2.1.1.2 Framework of Scrum

The framework of Scrum consists of three components including roles, events and artifacts [Sch04].

Roles
The fundamental unit of Scrum is a small team of people, a Scrum team. The Scrum team consists of [SS20]:

1. One Product Owner (PO)
2. One Scrum Master (SM)
3. Developers

Within a Scrum team, there are no sub-teams or hierarchies. It is a cohesive unit of professionals focused on one objective at a time, the product goal. The Scrum team is

self-managing, self-organizing and cross-functional to maximize team performance. All of the team members are responsible for both the success and the failure of sub-systems and of entire systems [Sch04] [SS20].

The PO is responsible for getting initial and on-going funding for the project by creating the project's overall requirements, Return on Investment (ROI) objectives and release plan [Sch04] as well as accountable for maximizing the value of the product resulting from the work of the Scrum team [SS20]. The PO is also accountable for effective product backlog management, which includes [SS20]:

- Developing and explicitly communicating the product goal
- Creating and clearly communicating product backlog items
- Ordering product backlog items
- Ensuring that the product backlog is transparent, visible and understood

The developers are committed to creating any aspect of a usable increment each sprint. They are accountable for [SS20]:

- Creating a plan for the sprint, the sprint backlog
- Instilling quality by adhering to a Definition of Done (DoD)
- Adapting their plan each day toward the sprint goal
- Holding each other accountable as professionals

The SM is responsible for ensuring that Scrum values, practices and rules are enacted and enforced [Sch04] by helping everyone understand Scrum theory and practice, both within the Scrum team and the organization [SS20]. The SM is accountable for the Scrum team's effectiveness [SS20] and tries to remove any impediments imposed on the developers [Sch04] Furthermore, the SM represents management and the team to each other [SB02].

The SM serves the Scrum team by [SS20]:

- Coaching the team members in self-management and cross-functionality
- Helping the Scrum team focus on creating high-value increments that meet the DoD
- Causing the removal of impediments to the Scrum team's progress

- Ensuring that all Scrum events take place and are positive, productive, and kept within the timebox

The SM serves the PO by [SS20]:

- Helping find techniques for effective product goal definition and product backlog management
- Helping the Scrum team understand the need for clear and concise product backlog items
- Helping establish empirical product planning for a complex environment
- Facilitating stakeholder collaboration as requested or needed

The SM serves the organization by [SS20]:

- Leading, training and coaching the organization in its Scrum adoption
- Planning and advising Scrum implementations within the organization
- Helping employees and stakeholders understand and enact an empirical approach for complex work
- Removing barriers between stakeholders and Scrum teams

Events

There are several events in the Scrum process including the sprint planning, the daily Scrum, the sprint review and the sprint retrospective. They all happen within sprints, which are fixed length events of one month or less, where ideas are turned into value. A new sprint starts immediately after the conclusion of the previous sprint and during the sprint 1) no changes are made that would endanger the sprint goal, 2) quality does not decrease, 3) the product backlog is refined as needed and 4) the scope may be clarified and renegotiated with the PO as more is learned [SS20].

The sprint planning initiates the sprint by laying out the work to be performed for the sprint. The project is broken down into a set of small and manageable tasks so that all the tasks can be completed in one sprint [Cho08]. The PO ensures that attendees are prepared to discuss the most important product backlog items and how they map to the product goal [SS20].

The daily Scrum is a 15-minute status meeting for developers of the Scrum team to talk about what has been accomplished since the last meeting, what items will be done before the next meeting and what obstacles developers have. It facilitates communications,

identifies and removes impediments to development, highlights and promotes quick decision-making and improves transparency (visibility) [SS20] [Cho08].

During the sprint review the outcome of the sprint is inspected and future adaptations are determined. The Scrum team presents the results of their work to key stakeholders and progress toward the product goal is discussed. Based on this information, attendees collaborate on what to do next. The product backlog may also be adjusted to meet new opportunities [SS20].

The sprint retrospective concludes the sprint. It is held to plan ways to increase quality and effectiveness. The Scrum team inspects how the last sprint went with regards to individuals, interactions, processes, tools and their DoD. It is discussed what went well during the sprint, what problems were encountered and how those problems were (or were not) solved. Subsequently, the Scrum team identifies the most helpful changes to improve its effectiveness in the future [SS20].

Artifacts

The Scrum process provides three artifacts namely the product backlog, the sprint backlog and the increment. They all represent work or value and are designed to maximize transparency of key information. Each artifact contains a commitment to ensure it provides information that enhances transparency and focus against which progress can be measured [SS20]:

- For the product backlog it is the product goal
- For the sprint backlog it is the sprint goal
- For the increment it is the DoD

The product backlog is a collection of functional and non-functional requirements, which are prioritized in order of importance to the business. The items in the product backlog are created and maintained by the PO [Cho08]. Furthermore, the product backlog contains the product goal, which describes a future state of the product and can serve as a target for the Scrum team to plan against [SS20].

The sprint backlog is a highly visible, real-time picture of the work that the developers plan to accomplish during the sprint in order to achieve the sprint goal, which is the single objective for the sprint. It is composed of the sprint goal (why), the set of product backlog items selected for the sprint (what), as well as an actionable plan for delivering the increment (how) [SS20]. The sprint backlog is created by team members from the product backlog in a way that the high priority items in the product backlog are first selected and broken into a set of smaller tasks [Cho08].

An increment is a concrete stepping stone toward the product goal. Each increment is additive to all prior increments and thoroughly verified, ensuring that all increments work

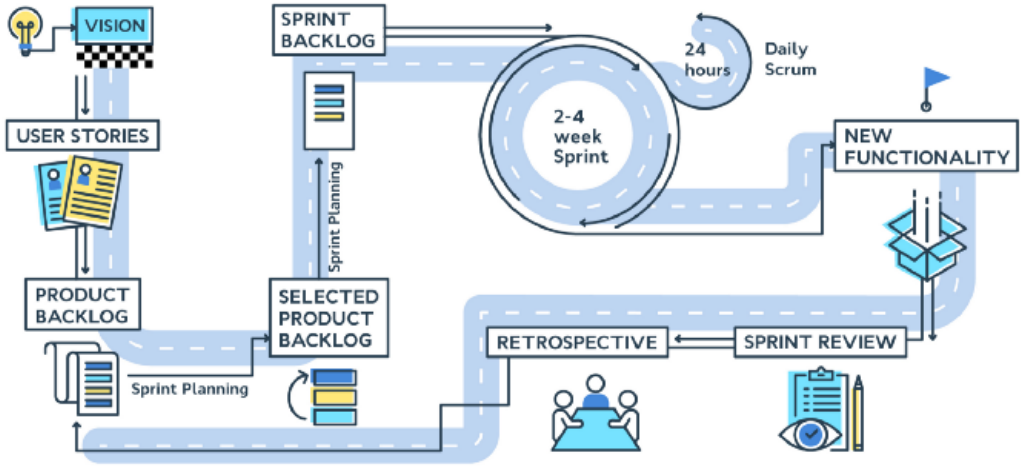

Figure 2.2: Scrum process [Scr17]

together. In order to provide value, the increment must be usable. The DoD is a formal description of the state of the increment when it meets the quality measures required for the product. The moment a product backlog item meets the DoD, an increment is born [SS20].

2.1.1.3 Flow of Scrum

The Scrum process begins with a vision of the system and a simple plan on ROI and release milestones. The vision is described in business terms rather than technical terms. The vision may be unclear at first but will become more precise as the project moves forward. As mentioned earlier, the PO is responsible for getting initial funding, delivering the vision while maximizing ROI and creating the product backlog. In the sprint planning, the PO explains the content, purpose, meaning and intentions of each item in the product backlog. Subsequently, the items are divided into smaller tasks, the sprint goal is defined and the associated tasks to accomplishing this goal are placed in the sprint backlog. All the tasks in the sprint backlog are processed through the iteration of the sprint which consists of the daily Scrum. At the end of the sprint, all tasks that meet their DoD (newly born increments) are presented in the sprint review and progress toward the product goal is discussed. All other tasks are placed in the product backlog for future consideration. Finally, the Scrum team concludes the sprint during the sprint retrospective by reflecting on the last sprint and identifying helpful changes to improve effectiveness in the future. Figure 2.2 illustrates the flow of the Scrum process.

2.2 DevOps

The term DevOps was not established overnight. Instead, many movements and people have influenced the development of DevOps and the term is still subject to continuous evolution [Hü12]. As a result, no common understanding of what it entails has been achieved yet [JbAPT16].

Therefore, one part of existing research focuses on further characterizing and clarifying DevOps. Jabbari et al. [JbAPT16] characterize DevOps by exploring central components of DevOps definitions reported in the literature, concluding that "DevOps is a development methodology aimed at bridging the gap between Dev and Ops, emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices". Ghantous & Gill [GG17] provide clarity and understanding of DevOps by developing catalogs of underpinning concepts, practices, tools, benefits and challenges which provide a collective knowledge base of DevOps that can be used by researchers and practitioners.

The main part of existing research focuses on the implementation of DevOps. Books like "The DevOps Handbook" [KHD+16] and "DevOps for Developers" [Hü12] provide general guidance transitioning towards a DevOps process by introducing approaches, processes and tools. Furthermore, research is conducted using both literature review and case study approaches identifying practices and principles, which organizations choose to include in their DevOps adoption journeys [MG20], identifying factors influencing the implementation of DevOps [SBO18a], as well as exploring advantages and problems adopting DevOps [TSM19b]. It is concluded that, in order to fully understand the DevOps adoption advantages and problems, more case studies need to be conducted in the future interviewing some organizations directly [TSM19b].

As mentioned in section 1.1, within this thesis the term DevOps is understood as a blend of development (representing software developers, including programmers, testers, and quality assurance personnel) and operations (representing the experts who put software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians) [Hü12].

DevOps extends the boundaries of ownership, accountability, responsibility and transparency across to production. It breaks down silos between development and operations (and security), and makes developers responsible for ensuring that the software they build always does what it is supposed to do [Bir17].

DevOps extends feedback loops from design and development into production and back, to shape design and improve engineering practices based on how a service is actually used and how it runs in the real world. It deals with the complete picture [Bir17].

DevOps emphasizes small and continuous releases and compresses the timeframes in which teams work [JbAPT16]. Changes are pushed out directly to users in production every day, or many times each day instead for every month or every couple of weeks.

This changes how organizations design software and think about software development, how teams are structured and how they work, and how they make decisions [Bir17].

DevOps is using a combination of cultural philosophies, practices and tools to increase an organization's ability to deliver applications and services at high velocity. Product teams adopt DevOps when they intend to look out for integrating processes across the product lifecycle, which is possible by leveraging processes, tools and automation. The principles of DevOps bridge the gaps between siloed teams and leverage automation for optimizing the processes that connect the entire lifecycle for the product. Each phase in the lifecycle is driven with its own set of tools, processes and teams. A DevOps model integrates each phase and each tool, streamlines the processes and focuses on a common vision, following the belief of a continuous journey of improvements (Figure 2.3) [SRA22].

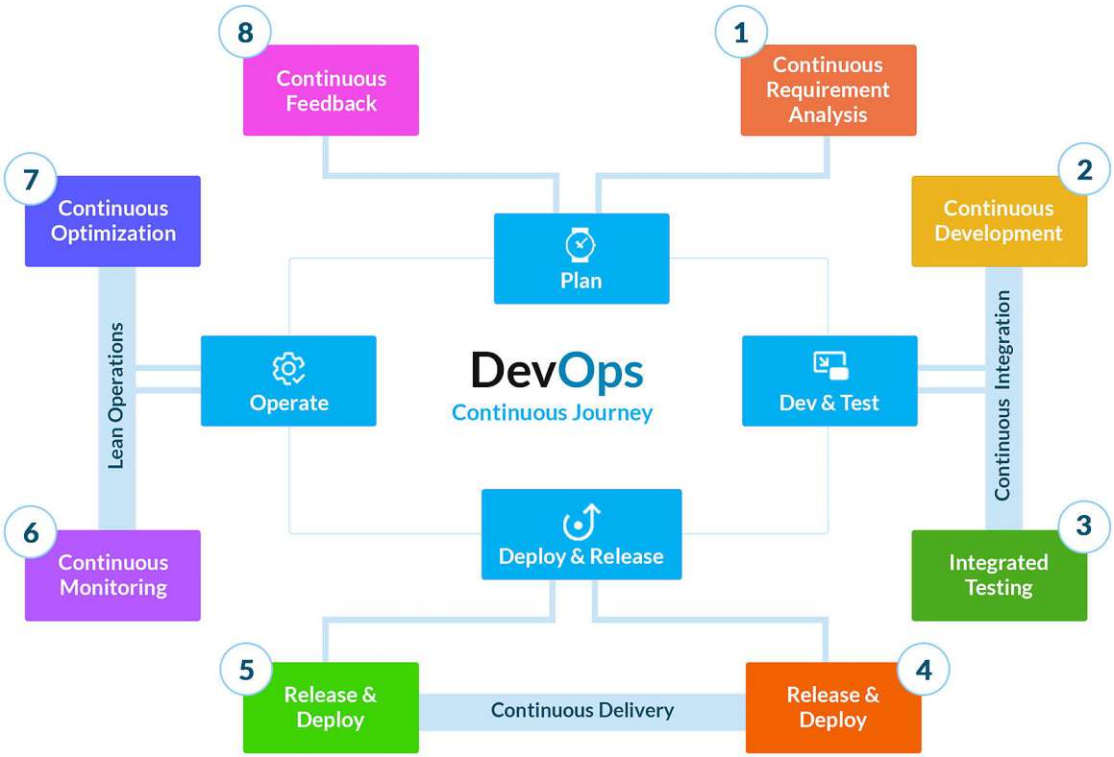

Figure 2.3: DevOps continuous journey [SRA22]

2.3 Agile DevOps

Agile DevOps is about extending agile ideas and values from product management and software development to the rest of the service delivery organization. It is about following agile and lean approaches across the full life cycle of a system, from planning and design to release, and to ongoing operations, maintenance and support. It is about breaking work down so that it can be delivered incrementally and iteratively by small, collaborative and

cross-functional teams; moving decision-making down so that these teams can respond more effectively to new information; continuously learning, adapting and improving [Bir17].

DevOps applies agile ideas, values, and approaches to packaging and deployment, change control and release management, service delivery, production support and maintenance, security and compliance. It also takes agile practices like continuous integration, test automation and test-driven-development, refactoring, pairing and peer reviews, and applies them to infrastructure provisioning and configuration management in code [Bir17]. Furthermore it adds the emphasis on automating the analysis of applications, i.e. continuously monitoring the performance of the system, presenting the results in automated dashboards, etc. and provides specific approaches and tools, such as infrastructure as code, or the DevOps maturity evaluation model [JbAPT16].

Overall, Agile and DevOps complement each other, but they should not be considered as replacements for each other [SRA22]. DevOps — the approach, the practices, even the tools — all build on and extend Agile. DevOps and Agile require the same kind of commitment and energy, the same approach to cultural and organizational change, the same focus on iterative and incremental change and on feedback and learning. [Bir17].

IT teams that practice Agile and DevOps reap long-term benefits such as the following [SRA22]:

- Collaborative and self-organized teams
- Embracing change through trust and transparency
- Faster time to market with automation
- Continuously improving with feedback loops
- Simplified processes and integrated workflows
- Lower costs
- Higher customer satisfaction scores
- Better business alignment

As DevOps and Agile complement each other, this should also hold for Scrum, as it is the most prominent agile methodology. However, existing research in the context of Scrum is limited. As DevOps is often put in the context of agile software development, most references to Scrum are due to it being an agile process. Exceptions to this are studies like [EAD17], where experiences of companies adapting DevOps in their existing (Scrum) processes are analyzed. Schaefer & Söllner [SS17] aim to implement DevOps using a combination of Scrum and Kanban. Furthermore, Samarawickrama & Perera [SP17] introduce a framework to enhance Scrum with DevOps. However, their introduced

framework focuses only on continuous integration and rapid delivery, neglecting other DevOps practices.

This state-of-the-art analysis shows perfectly that there are still many uncertainties regarding DevOps. Resolving these uncertainties is a lot of work and leaves room for further research. Firstly, existing DevOps research should be classified in order to get a better understanding of what has been done already. Secondly, there should be a clear overview of which practices DevOps entails and what these practices actually mean, as most existing research seems to name practices without any context or explanation. Finally, as DevOps can be seen as an extension of Agile and research in the context of Scrum, the most common agile methodology, is limited, there is still potential for further research regarding the interplay between those two methodologies.

CHAPTER 3

Research Methodology

This chapter gives an overview of the used research methodology within this thesis. It highlights the deducted research questions and gives an overview of the methodologies used to answer them. Finally, this chapter outlines the research design, listing the research phases, their content and their outcome.

3.1 Research Questions

In order to help companies to adapt their working process towards enabling agile DevOps, by extending a common Scrum process using DevOps practices, taking into account areas important for development, such as being agile and open to changes in requirements at any time, as well as areas being important for operations, such as stability of the requirements and the integrity of the deployed application [GZCP20], and reducing the risk of development and operations working towards unaligned goals [SP17], the following research questions (RQs) are deducted:

RQ 1: Which DevOps practices are most frequently used in state-of-the-art research in the context of agile software development?

RQ 2: Which DevOps practices can be used to extend agile software development, by example of Scrum, in order to reduce the risk of development and operations working towards unaligned goals?

3.1.1 Methodology RQ1

RQ1 is answered conducting a systematic literature review, following the process created by Petersen et al. [PFMM08]. Firstly, a systematic map is built, conducting thematic analysis [DWAJ+05], identifying relevant publications in the focus area and classifying

the research field [PFMM08]. Secondly, the relevant publications are analyzed in detail by conducting a full-text analysis in order to identify and map all mentioned DevOps practices [PFMM08].

A detailed description of the systematic review process can be found in Chapter 4 (Systematic Literature Review).

3.1.2 Methodology RQ2

RQ2 is answered using a conceptual-deductive analysis, a combination of deduction, which is a general analytical method based on modeling and semi-formal models [WH06]. This method uses different theories to develop a new model and gain new insights [PMSS03] and is therefore part of the construction science paradigm (Design Science) [Sim98]. Based on the results of RQ1, the found DevOps practices are further examined and practices which can be used in combination with Scrum, in order to reduce the risk of development and operations working towards unaligned goals, are chosen.

A detailed description of this process can be found in Chapter 5 (Process Definition).

Additionally, evaluation of the proposed model is done using semi-structured interviews as a method of qualitative data collection [WH06] and summarizing qualitative content analysis according to Mayring [May15], which summarizes and structures the feedback using a matrix.

A detailed description of this process can be found in Chapter 6 (Process Validation).

Finally, the summarized and structured feedback is used in order to update the Scrum process with a refined set of suitable DevOps practices in Chapter 7 (Results).

3.2 Research Design

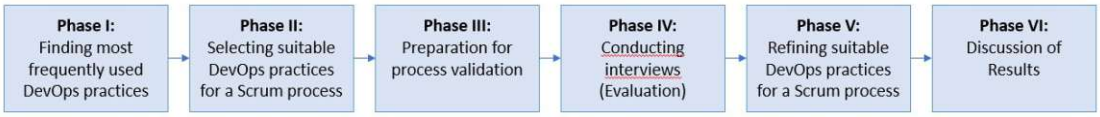

Figure 3.1: Methodological Approach

Phase I - Finding most frequently used DevOps practices

A set of most frequently used DevOps practices in state-of-the-art research, in the context of agile software development, is defined. To this end a systematic literature review, following the process created by Petersen et al. [PFMM08], is conducted as outlined above (3.1.1).

Outcome: A set of most frequently used DevOps practices in the context of agile software development

Phase II - Selecting suitable DevOps practices for a Scrum process

A common Scrum process is selected and extended using the previously established DevOps practices. This is done using a conceptual-deductive analysis (3.1.2). The selection of used practices is done using criteria which will be defined based on the outcome of Phase I.

Outcome: A Scrum process extended with a set of DevOps practices, in order to reduce the risk of development and operations working towards unaligned goals

Phase III - Preparation for process validation

A set of selection criteria for interview partners is defined in order to identify qualified experts in the context of this study.
Preparation of an interview guideline for semi-structured expert interviews in order to get feedback on the chosen practices.

Outcome: An interview guideline and a set of selection criteria for interview partners

Phase IV - Conducting interviews (Evaluation)

Using the previously defined interview guideline (Phase III), representatives of several companies are interviewed in order to validate the selected DevOps practices in combination with Scrum as outlined above.
The feedback will be based solely on the experts experience with Scrum and DevOps, as the updated Scrum process is not implemented in an organization beforehand.
The collected feedback is summarized and structured using summarizing qualitative content analysis according to Mayring [May15].

Outcome: Summarized and structured feedback from expert interviews (qualitative data collection)

Phase V - Refining suitable DevOps practices for a Scrum process

Based on the results of the interviews (Phase IV) the selected set of DevOps practices that can be used in combination with Scrum (Phase II) is updated and refined.

Outcome: An updated Scrum process with a refined set of suitable DevOps practices, in order to reduce the risk of development and operations working towards unaligned goals

Phase VI - Discussion of Results

Discussion of findings with regard to state-of-the-art research of Phase I.

Outcome: Summary of the results, acknowledgement of limitations, proposal for future work

CHAPTER 4

Systematic Literature Review

This chapter describes the conducted systematic literature review. Firstly, by conducting thematic analysis [DWAJ+05], identifying relevant publications in the focus area and classifying the research field, a systematic map is built following the process created by Petersen et al. [PFMM08]. Thematic analysis is an interesting analysis method, as it helps to see which categories are well covered in terms of number of publications [PFMM08].

Following the systematic mapping process, the literature review is performed by analyzing the identified relevant publications in detail. Therefore, a full-text analysis of the publications is conducted in order to identify and map all mentioned DevOps practices [PFMM08].

Firstly the systematic review process is described. This contains the definition of the research questions, the execution of the search for primary studies as well as its parameters, the defined inclusion and exclusion criteria by which the found papers are screened, as well as the definition of a classification scheme by which the selected studies are sorted. Secondly, the study selection process is described and the results are represented clearly. Finally, the most frequently used DevOps practices found within this systematic literature review are listed and explained.

4.1 Systematic Review Process

The systematic literature review is a secondary study method inspired from medical research. Briefly, a systematic review goes through existing primary reports, reviews them in-depth and describes their methodology and results, aiming at establishing a state of evidence [KC07]. This is not the goal for systematic maps. A systematic map provides a structure of the type of research reports and results that have been published by categorizing them. The analysis of results focuses on frequencies of publications for

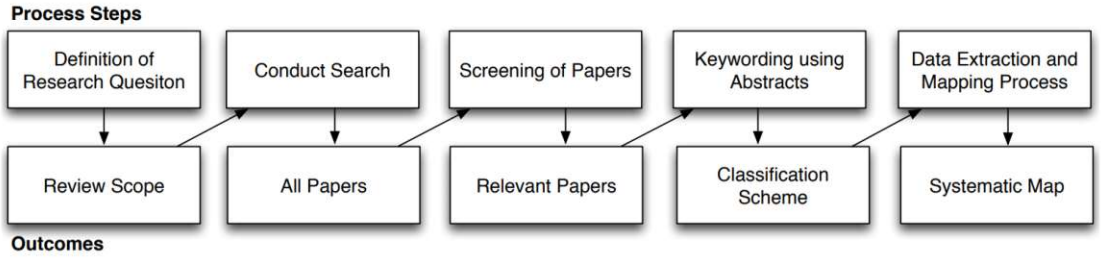

Figure 4.1: The Systematic Mapping Process [PFMM08]

categories within the scheme and often gives a visual summary, the systematic map, of its results. Thereby, the coverage of the research field can be determined. It requires less effort while providing a more coarse-grained overview [KC07] [PFMM08]. Petersen et al. argue the a thematic summary leading to a map can be the first step in a more detailed systematic review [PFMM08] and they created a process for systematic mapping studies in software engineering. The essential process steps are definition of research questions, conducting the search for relevant papers, screening of papers, keywording of abstracts and data extraction and mapping (Figure 4.1). Each process step has an outcome, the final outcome of the process being the systematic map [PFMM08].

4.1.1 Definition of Research Questions (Research Scope)

The main goal of a systematic literature review is to establish a state of evidence by reviewing existing primary reports in-depth and describing their methodology and results. On the other hand the goal of a systematic mapping study is to provide an overview of a research area, and identify the quantity and type of research and results available within it, in order to see trends as well as identify the forums in which research in the area has been published [PFMM08]. The combination of these goals is reflected in RQ1 (3.1).

4.1.2 Conduct Search for Primary Studies (All Papers)

In order to identify the primary studies, a search string on scientific databases is used. Following Peterson et al.'s suggestion, the structure of the search string is driven by the research question [PFMM08] and thereby looks for *DevOps* AND *practices* AND *agile* AND *software development*, with synonyms or variants of all terms separated by OR-operators, as follows:

(devops OR "dev ops") AND (practice OR method) AND (agile OR scrum OR kanban OR "extreme programming" OR XP OR "pair programming" OR "lean development" OR "lean software development" OR crystal OR "feature-driven development" OR "feature-driven software development" OR FDD) AND ("software engineering" OR "software development")

Due to the limited time frame and resources of this study, the used research databases were limited to Scopus. Scopus was chosen as it is freely accessible via the university

library service (Vienna University of Technology). Furthermore, Scopus is the largest abstract and citation database containing both peer-reviewed research literature and quality web sources [Lib24]. As an abstracting and indexing (A&I) provider, Scopus contains publications found in other scientific databases such as IEEE Xplore [Cen22], ACM Digital Library [ACM24] and Springer [Spr24], among others. These scientific databases automatically send updates about their new articles to Scopus, giving a good overview of the available scientific literature of a field. In addition, Scopus content is reviewed and ultimately selected by an independent Content Selection and Advisory Board (CSAB), an international group of scientists and researchers with journal editor experience and who represent the major scientific disciplines indexed on Scopus. Year-round, the CSAB members are responsible for reviewing all new titles that are suggested for indexing on Scopus. They also review and ensure that the quality of existing content is maintained [Els24]. This process further ensures the quality of papers used within this study. Working with A&I services it is recommended to wait 12 weeks from an articles publication to its search [Cen22], however, as by the time of search the end date of the search period lies more than 12 weeks in the past, all indexed publications from other scientific databases should be found correctly.

The publication years are set to 2008–2023, covering the whole time frame from the introduction of DevOps in 2008 up to now, and the language is set to English. Furthermore, the search is limited to abstract, keywords and title to minimize the number of irrelevant hits. Although there are books touching the area, books are excluded on all database searches as they are generally not peer-reviewed.

Additionally the search is extended by studies found in existing literature reviews, which are found using the defined search parameters. As those existing literature reviews were done using different search and filter criteria, the defined search and filter criteria of this study are applied to all papers found in those existing literature reviews.

4.1.3 Screening of Papers for Inclusion and Exclusion (Relevant Papers)

Inclusion and exclusion criteria are used to exclude studies that are not relevant to answer the research questions [PFMM08].

Inclusion criteria are defined in relation to the research question (RQ1). Included papers need to address DevOps practices in agile software development and be available via the university library service (Vienna University of Technology) during the time of search or be freely available on the web.

Exclusion criteria are not or not rigorously peer-reviewed literature such as books, theses, workshop papers and gray literature as well as publications in other languages than English. Furthermore, papers covering the extension of DevOps beyond development and operations towards other departments, such as security (DevSecOps), privacy (DevPrivOps) or business (BizDevOps), are excluded.

4.1.4 Keywording of Abstracts (Classification Scheme)

Following the identification of relevant papers, they are classified using a classification scheme. Development of this classification scheme is done using *Keywording*. This is done in two steps. First, the abstracts are examined for keywords and concepts that reflect the contribution of the paper. While doing so the context of the research is identified. When this is done, the set of keywords from different papers are combined together to develop a high level understanding about the nature and contribution of the research. This helps defining a set of categories which is representative of the underlying population. When abstracts are of too poor quality to allow meaningful keywords to be chosen, the introduction or conclusion sections of the paper can also be studied. As soon as a final set of keywords has been chosen, they can be clustered and used to form the categories for the map [PFMM08]. Figure 4.2 shows this systematic process.

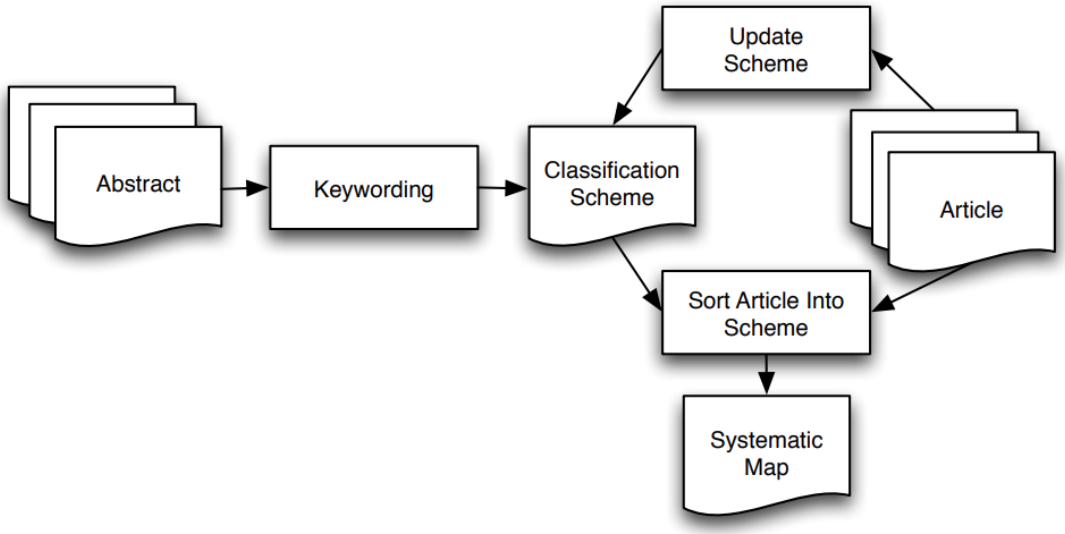

Figure 4.2: Building the Classification Scheme [PFMM08]

In this thesis, four main facets were created. One facet structures the topic by DevOps practices mentioned within the paper. The second facet structures the content by the way DevOps is examined in the paper. The third facet structures the content by agile methodology mentioned within the paper. The fourth facet is general and independent from a specific focus area and structures the research type using an existing classification of research approaches by Wieringa et al. [WMMR06], as shown in Table 4.1.

4.1.5 Data Extraction and Mapping of Studies (Systematic Map)

When having the classification scheme in place, the relevant articles are sorted into the scheme, i.e., the actual data extraction takes place. As shown in Figure 4.2 the

Category	Description
Validation Research	Techniques investigated are novel and have not yet been implemented in practice. Techniques used are for example experiments, i.e., work done in the lab.
Evaluation Research	Techniques are implemented in practice and an evaluation of the technique is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry.
Solution Proposal	A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation.
Philosophical Papers	These papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework.
Opinion Papers	These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodologies.
Experience Papers	Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author.

Table 4.1: Research Type Facet [WMMR06]

classification scheme can still evolve while doing the data extraction, like adding new categories or merging and splitting existing categories.

The analysis of the results focuses on presenting the frequencies of publications for each category. This makes it possible to see which categories have been emphasized in past research and thus to identify gaps and possibilities for future research [PFMM08].

4.2 Study Selection

The selection process is illustrated in Figure 4.3. First the database was searched with the predefined search string and the noted limitations (see Chapter 4.1.2). This conducted search resulted in 223 studies. In the second step the results were filtered by reading the abstracts and applying the noted exclusion criteria (see Chapter 4.1.3). This resulted in 44 studies.

Reading the abstracts, two existing secondary literature studies discussing DevOps practices were found. "What is DevOps? A Systematic Mapping Study on Definitions

and Practices" by Jabbari et al. [JbAPT16], a systematic mapping study, and "DevOps: Concepts, Practices, Tools, Benefits and Challenges" by Ghantous & Gill [GG17], a systematic literature review. However, as those two studies were already published in 2016 and 2017 respectively and their focus is on DevOps in general, without any restrictions on addressing DevOps practices in agile software development, none of them can be seen as a substitute for this studies systematic literature review. Nevertheless, the results of those two studies are analyzed further by applying this studies predefined criteria to all found publications discussing DevOps practices. The two existing secondary literature studies combined settled on 29 individual studies discussing DevOps practices, out of which 3 studies were already found in this studies literature search, leaving 26 additional studies to be considered. Those 26 additional studies were then filtered using this studies predefined criteria, leaving 9 studies which were added for further analysis.

Combining the found studies from the conducted search and existing secondary literature studies a total of 53 studies were selected for data extraction and mapping. The detailed list can be seen in Appendix A.

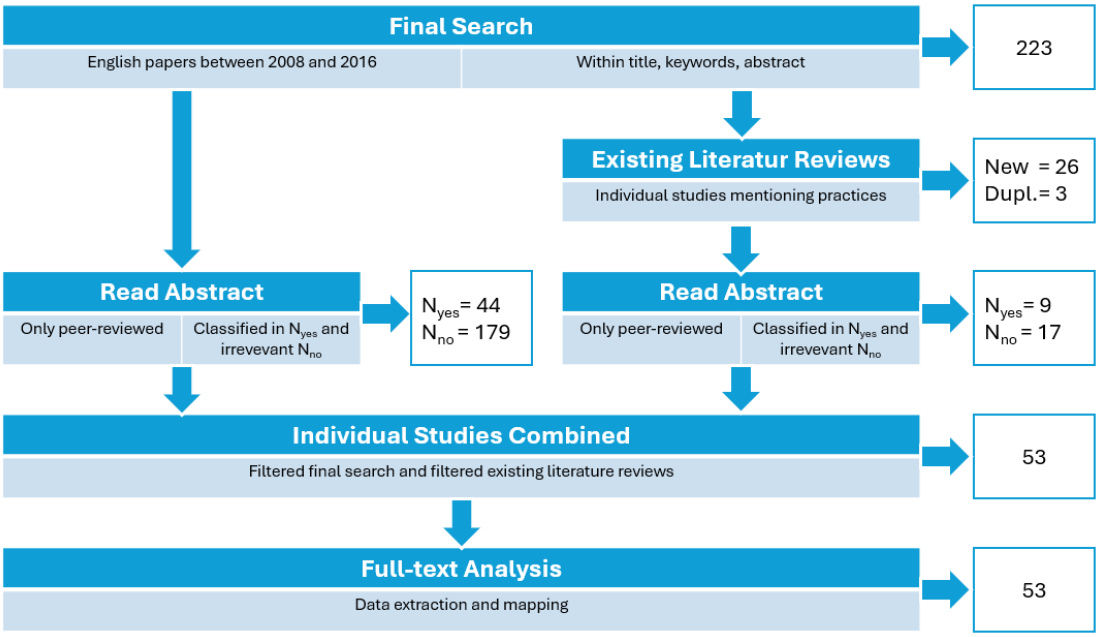

Figure 4.3: Study selection process: N_{yes}/N_{no} show the number of relevant/irrelevant studies after the respective steps

4.3 Data Extraction and Synthesis

Data extraction was done with a full-text analysis of the final set of 53 included studies. A spreadsheet was used to categorize papers using the predefined facets (Chapter 4.1.4). The classes for the two of the facets were predetermined. The facet research type was

defined by Wieringa et al [WMMR06] and the facet agile methodologies was predefined by the agile methodologies used in the search string. The facets for DevOps practices and research topics were continuously extended during the full-text analysis. Whenever a new practice or topic was added as a class to the facet, the previous studies were reevaluated towards this new class.

Figure 4.4 shows the distribution of published papers over the years. The years after the introduction of DevOps in 2008 publications were scarce, followed by a continuous increase of publications over the last couple of years.

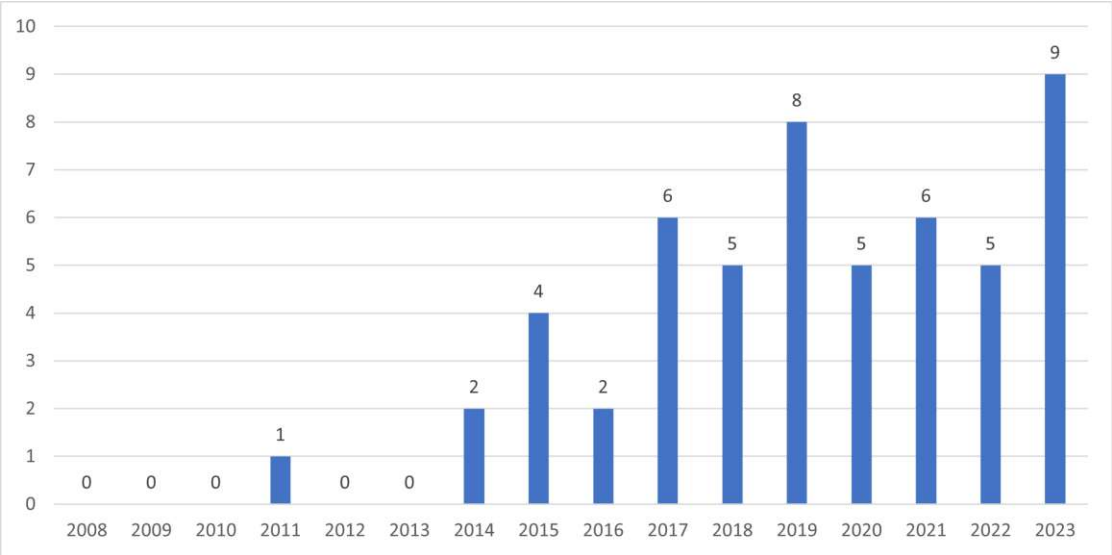

Figure 4.4: Distribution of papers over the years 2008-2023

Table 4.2 shows the mapping of research types over the whole period of 2008-2023. Up until 2013 there were hardly any publications addressing DevOps practices in agile software development. Beginning in 2016 empirical papers conducting validation or evaluation research were picking up. This shows the growing interest in adopting DevOps. However, in the same period there was also a steady flow of new philosophical papers, trying to structure the research field. This proves the point that the scientific community still sees the need to further classify DevOps. A detailed mapping of papers towards their mapped research type can be seen in Table 4.3.

	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023
Validation Research													1		1	
Evaluation Research				1			1		1	2	1	3	3	2	3	2
Solution Proposal								2		2		1		1	1	
Philosophical Paper							1	1	1	2	4	3	1	3		7
Opinion Paper																
Experience Paper								1				1				

Table 4.2: Distribution of research types over the years 2008-2023

This can be seen even better in Figure 4.5, which shows the distribution of research topics

Research Type	Reference	Count
Validation Research	[4], [30]	2
Evaluation Research	[2], [10], [11], [14], [15], [22], [25], [26], [27], [28], [34], [35], [37], [41], [45], [46], [48], [51], [52]	19
Solution Proposal	[8], [19], [32], [38], [40], [50], [53]	7
Philosophical Paper	[1], [3], [5], [6], [9], [12], [13], [16], [17], [20], [21], [23], [24], [29], [31], [33], [36], [39], [42], [43], [44], [47], [49]	23
Opinion Paper		0
Experience Paper	[7], [18]	2

Table 4.3: Research Types and their References (Appendix A)

over the whole period of 2008-2023. It needs to be noted that papers can be mapped to several research topics. Papers analyzing actual experiences adopting DevOps or the practical adoption of DevOps models are scarce, slightly picking up over the last couple of years. However, the number of papers further structuring the research field by laying out definitions and practices, benefits and challenges, as well as examining theoretical DevOps (-models) has continuously grown and makes up most of the existing research. A detailed mapping of papers towards their mapped research topics can be seen in Table 4.4.

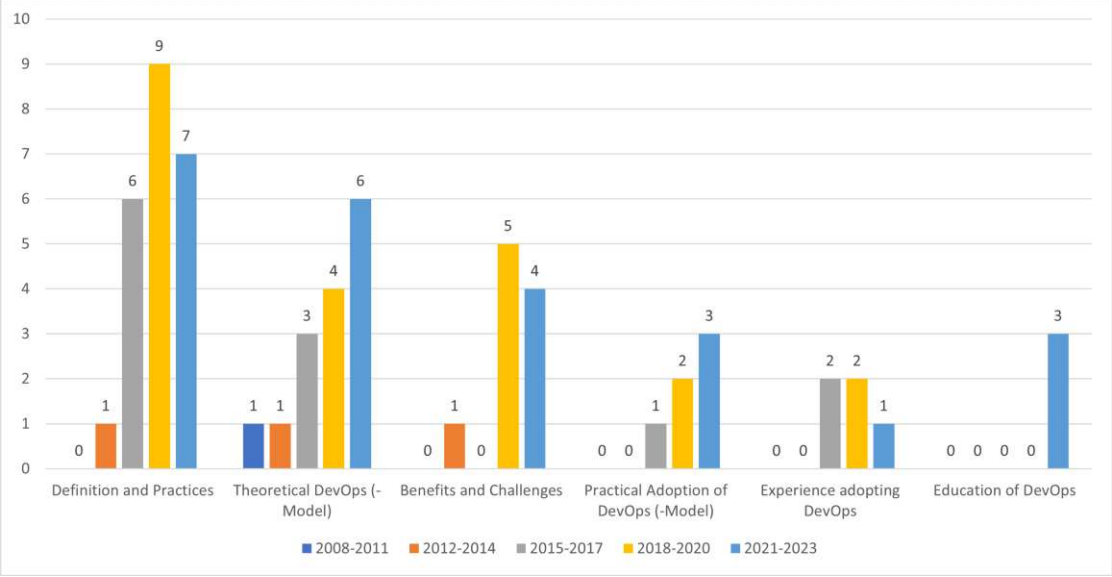

Figure 4.5: Distribution of research topic over the years 2008-2023

Figure 4.6 shows the distribution of agile methodologies mentioned within the analyzed papers over the whole period of 2008-2023.

DevOps Topic	Reference	Count
Definition and Practices	[5], [11], [12], [13], [16], [18], [21], [22], [23], [24], [25], [26], [27], [31], [33], [35], [36], [37], [41], [42], [44], [47], [48], [49]	23
Theoretical DevOps (-Model)	[1], [6], [8], [9], [15], [15], [17], [20], [23], [30], [38], [39], [40], [43], [50], [51]	15
Benefits and Challenges	[10], [11], [13], [16], [18], [29], [31], [33], [37], [42], [46]	10
Practical Adoption of DevOps (-Model)	[6], [19], [28], [32], [34], [45], [53]	6
Experience adopting DevOps	[3], [4], [5], [7], [10], [18]	5
DevOps Education	[2], [3], [14], [52]	3

Table 4.4: Topics by which DevOps is examined and their References (Appendix A)

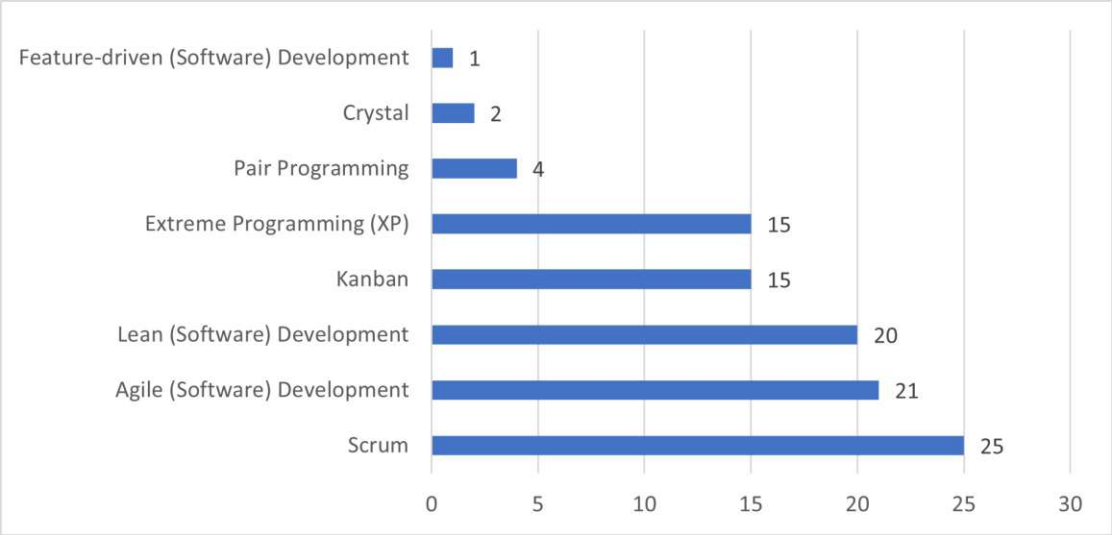

Figure 4.6: Distribution of agile methodologies over the years 2008-2023

Nearly all papers mention the umbrella term agile (software) development and nearly half the papers do so without mentioning any concrete agile methodology. The agile methodology mentioned namely most frequently is Scrum. This underpins this studies selection of a Scrum process as a suitable representative of agile methodologies to extend with DevOps practices. However, despite nearly half of the selected studies mentioning Scrum, hardly any of them go into it in further detail, proving the point that there is still great potential for further research regarding the interplay between the two methodologies DevOps and Scrum. A detailed mapping of papers towards agile methodologies mentioned can be seen in Table 4.5.

Table 4.6 shows all DevOps practices found conducting a full-text analysis of the selected

Agile Methodology	Reference	Count
Scrum	[2], [9], [10], [11], [12], [13], [16], [18], [19], [20], [22], [23], [25], [26], [27], [31], [32], [34], [35], [38], [41], [43], [44], [48], [52]	25
Agile (Software) Development	[1], [4], [5], [6], [7], [14], [15], [17], [21], [28], [29], [30], [36], [37], [39], [40], [45], [46], [47], [51]	21
Lean (Software) Development	[2], [8], [10], [12], [13], [16], [18], [20], [23], [24], [25], [31], [33], [34], [38], [41], [42], [44], [48], [49]	20
Kanban	[2], [9], [10], [12], [13], [16], [22], [23], [25], [27], [31], [33], [34], [38], [41]	15
Extreme Programming (XP)	[10], [11], [12], [13], [16], [19], [22], [31], [34], [35], [38], [41], [44], [48], [52]	15
Pair Programming	[2], [32], [38], [41]	4
Crystal	[34], [38]	2
Feature-driven (Software) Development	[34]	1

Table 4.5: Agile Methodologies and their References (Appendix A)

studies as well as a detailed mapping of which papers included them.

DevOps Practice	Reference	Count
Continuous Integration	[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [29], [31], [32], [33], [34], [35], [36], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [52], [53]	49
Continuous Delivery	[1], [2], [3], [4], [5], [6], [8], [9], [10], [12], [13], [14], [15], [16], [17], [18], [19], [21], [22], [23], [24], [26], [27], [28], [29], [31], [32], [33], [34], [36], [38], [39], [40], [41], [42], [43], [44], [46], [47], [48], [49], [50], [51], [52], [53]	45
Continuous Deployment	[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [15], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [31], [32], [33], [34], [35], [36], [38], [39], [40], [41], [42], [44], [45], [47], [49], [51], [52], [53]	45
Continuous Testing	[1], [2], [3], [4], [5], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [23], [24], [25], [26], [27], [29], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [42], [43], [44], [45], [46], [47], [48], [49], [52], [53]	45
Continuous Monitoring	[1], [2], [6], [8], [9], [11], [12], [13], [16], [17], [18], [19], [20], [21], [23], [24], [25], [26], [27], [28], [30], [31], [32], [33], [34], [35], [36], [37], [40], [42], [43], [45], [46], [47], [48], [49], [50], [53]	38
Collaboration and Communication	[3], [6], [7], [9], [10], [11], [14], [16], [17], [18], [19], [20], [21], [22], [23], [25], [26], [27], [29], [31], [32], [34], [35], [36], [37], [38], [40], [41], [42], [44], [45], [46], [47], [50], [51], [52]	36
Cloud Computing	[1], [2], [3], [5], [7], [10], [11], [12], [15], [16], [18], [19], [20], [23], [24], [25], [26], [27], [29], [32], [36], [39], [40], [42], [44], [45], [46], [47], [49], [50], [51], [52], [53]	33
Continuous Feedback	[1], [2], [6], [8], [11], [14], [16], [19], [20], [24], [26], [28], [30], [32], [33], [34], [35], [37], [40], [44], [46], [47], [49]	23
Code Versioning	[1], [2], [4], [7], [8], [9], [10], [14], [17], [19], [20], [25], [29], [32], [33], [35], [38], [39], [40], [41], [42], [45], [47]	23
Continuous Experimentation	[2], [6], [9], [10], [12], [13], [16], [18], [20], [21], [22], [23], [26], [30], [32], [33], [34], [35], [37], [44], [45]	21

Configuration Management Automation	[1], [2], [8], [16], [18], [19], [22], [23], [25], [27], [29], [33], [34], [36], [39], [40], [42], [45], [46], [52]	20
Infrastructure Automation	[2], [7], [8], [10], [11], [15], [16], [18], [21], [22], [25], [26], [27], [29], [32], [34], [43], [45], [52]	19
Sandboxing	[2], [5], [7], [9], [10], [11], [14], [17], [20], [25], [32], [33], [38], [44], [46], [47], [48]	17
Microservice Architecture	[2], [3], [5], [10], [14], [18], [23], [25], [26], [28], [29], [39], [43], [47]	14
Continuous Release	[9], [18], [20], [27], [31], [32], [33], [35], [44], [46]	10
Continuous Planning	[1], [11], [12], [13], [17], [20], [23], [44], [49]	9
Continuous Security	[1], [6], [12], [13], [16], [23], [25], [34], [35]	9
Quality Assurance	[3], [5], [10], [16], [17], [18], [19], [40]	8
Behavior Driven Development (BDD)	[10], [15], [19], [23], [37], [40], [48]	7
Process Standardization	[11], [12], [19], [26], [27], [36]	6
Static Code Analysis	[2], [9], [35], [37], [48], [53]	6
Test Driven Development (TDD)	[14], [15], [19], [37], [48]	5
Continuous Compliance	[6], [12], [13], [44]	4
Share critical services in team	[7], [18], [26], [40]	4
Change Management	[11], [40]	2
Stakeholder Participation	[11], [40]	2
Data Management	[34]	1
Strategic Direction	[34]	1
Measurement of Culture	[34]	1
Privileged Access Management	[34]	1
Trunk-based Development	[25]	1
Change-based Code Review	[25]	1
Feature Toggles	[10]	1
Reinforce Deployment Process	[31]	1

Table 4.6: DevOps Practices and their references (Appendix A)

4.4 Most frequently used DevOps Practices

Out of all the DevOps practices found during full-text analysis, those which were found in at least 5 different studies are selected as most frequent and therefore further explained in this chapter. Only continuous security is excluded from this list, as it extends DevOps beyond development and operations towards another department (security) and

is therefore excluded from this thesis (see Chapter 4.1.3).

4.4.1 Continuous Integration

While continuous integration is mentioned in 49 out of 53 selected studies, definitions or descriptions for continuous integration can only be found in 15 of them, namely studies [1], [8], [12], [13], [16], [17], [19], [21], [29], [35], [39], [40], [44], [48] and [49]. The other 34 studies, despite mentioning the practice by name, do not give a definition or description of it.

Studies [8], [12], [13], [19], [29], [40], [44], [48] and [49] state that developers working with continuous integration need to integrate their work into a shared repository frequently. Usually each person integrates at least daily, leading to multiple integrations per day.

Furthermore, studies [8], [12], [13], [16], [17], [21], [29], [35], [39], [40], [48] and [49] state that continuous integration automatically triggers when code is added to the code base and comprises interconnected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking compliance with coding standards, and building deployment packages, so teams can quickly collaborate on a single project.

Studies [1], [12], [13], [17] and [48] further state that the integration process of developer's code needs to detect errors quickly and avoid failure of builds. Any continuous integration failure is also an important event which may have a number of ceremonies and highly visible artifacts to help ensure that problems leading to integration failures are solved as quickly as possible by those responsible.

According to studies [8] and [48] continuous integration increases developer productivity, improves communication and enables software companies to have more frequent and shorter release cycles.

4.4.2 Continuous Delivery

While continuous delivery is mentioned in 45 out of 53 selected studies, definitions or descriptions for continuous delivery can only be found in 14 of them, namely studies [1], [8], [12], [13], [16], [19], [21], [39], [40], [43], [44], [46], [47] and [48]. The other 31 studies, despite mentioning the practice by name, do not give a definition or description of it.

Study [43] simply defines continuous delivery as a set of practices that aims to deliver value to customers rapidly, reliably and repeatedly with minimal manual overhead.

A more detailed definition can be found in studies [8], [16], [19], [21], [44], [46], [47] and [48], which state that continuous delivery treats every change as a potential release candidate to be frequently and rapidly evaluated through one's continuous delivery pipeline. This typically involves activities such as code analysis, documentation generation, acceptance testing, regulatory compliance assessments, license scanning and requirement verification.

Furthermore, studies [1], [40], [44], [46] and [47] elaborate that using continuous delivery, software always needs to be in a release state, so that it is always possible to deploy and/or release the latest working version.

While studies [8], [12], [39], [46] and [47] state that software is automatically deployed and delivered to a production-like environment, study [13] states that software is continuously ready for release and deployed to actual customers. However, studies [12] and [13] are both written by Brian Fitzgerald and Klaas-Jan Stol, with study [12] being the newer publication. It therefore seems like Fitzgerald and Stol updated their definition according to the scientific consensus and the listed definition of study [13] is no longer valid.

Study [44] distinguishes continuous delivery from continuous integration as an organization may practice continuous delivery and implement a continuous delivery pipeline, yet at the same time fail to convince its individual developers to adopt the practice of continuously integrating.

4.4.3 Continuous Deployment

While continuous deployment is mentioned in 45 out of 53 selected studies, definitions or descriptions for continuous deployment can only be found in 12 of them, namely studies [1], [8], [12], [13], [17], [19], [21], [29], [35], [39], [44], [49]. The other 33 studies, despite mentioning the practice by name, do not give a definition or description of it.

With studies [1], [8], [12], [17], [19], [21], [29], [39], [44] and [49], nearly all studies define continuous deployment as the process of deploying to production as soon as qualified changes in software code or architecture are ready, and without human intervention. They state that continuous deployment implies continuous delivery, therefore, release candidates evaluated and approved in continuous delivery are automatically placed in a production environment. Study [44] further adds that this often, but not necessarily, implies making it generally available to users.

A slight deviation from this definition can be found in study [35], which agrees that continuous deployment aims to automate the entire workflow to simplify the fast release of the software, however, states that the deliverable can be deployed in a desired environment (test or production).

Contrary to most definitions study [13] states that continuous deployment automatically deploys and delivers software to a production-like environment. However, studies [12] and [13] are both written by Brian Fitzgerald and Klaas-Jan Stol, with study [12] being the newer publication. It therefore seems like Fitzgerald and Stol updated there definition according to the scientific consensus and the listed definition of study [13] is no longer valid.

4.4.3.1 Continuous Release

The term continuous release is mentioned in 10 out of 53 selected studies. It is mainly used synonymously to continuous deployment, with the exception of study [44] which

clearly distinguishes the two terms. The other 9 studies, despite mentioning the practice by name, do not give a definition or description of it.

Study [44] defines continuous release as a business practice, where release candidates evaluated in continuous delivery are frequently and rapidly made available to users/customers [SMB17]. This study clearly distinguishes it as critically different from continuous deployment, depending on the context. In many Software-as-a-Service (SaaS) environments, release is achieved through deployment, however, in other contexts, where an organization is deploying for its own internal operational needs and thereby is its own user, the concept of a release is arguably not even applicable. Conversely, in the case of user installed software, continuous deployment is not an applicable concept even though continuous release may very well be [SMB17].

4.4.4 Continuous Testing

While continuous testing is mentioned in 45 out of 53 selected studies, definitions for continuous testing can only be found in 7 of them, namely studies [1], [12], [13], [19], [21], [37] and [49]. The other 38 studies, despite mentioning the practice by name, do not give a definition or description of it.

The listed definitions are as follows:

Study [1] defines continuous testing as a process of testing early, testing often, testing everywhere, and automating. It further states that one of its challenges is the environment of heterogeneity, which will never reflect the actual production environment and application architecture [AR19].

Studies [12] and [13] are both written by Brian Fitzgerald and Klaas-Jan Stol and define continuous testing as a process typically involving some automation of the testing process, or prioritization of test cases, to help reduce the time between the introduction of errors and their detection, with the aim of eliminating root causes more effectively [BSMG12].

Study [19] defines continuous testing as a process that verifies the correctness, quality and performance of developed software for business-critical processes and adherence to business' specification [AMB+16]. It bases continuous testing on a set of tools with varying capabilities, ranging from isolated code correctness tests to simulating a human-driven manual testing experience [GBM19].

Study [21] defines continuous testing as a process of executing tests and repeating activities using software [WFW+19]. This reduces manual testing, accelerates the testing process significantly [H+16] and strongly affects deployment speed [SBO18b].

Study [37] defines continuous testing as the practice of integrating test automation into the software delivery pipeline to promptly assess the business risks linked to a software release candidate [GHM18].

Study [49] defines continuous testing as building a test suite which should automatically be executed on every software build generated without any user intervention. This moves

the testing process earlier in the development cycle and furthermore allows the tests to be carried out on production-like system. Furthermore, any process that has to be repeated over time should be evaluated for possibilities of automation [Vir15].

4.4.5 Continuous Monitoring

While continuous monitoring is mentioned in 38 out of 53 selected studies, definitions or descriptions for continuous monitoring can only be found in 4 of them, namely studies [12], [21], [26] and [49]. The other 34 studies, despite mentioning the practice by name, do not give a definition or description of it.

Study [12] describes continuous monitoring as an operations practice, where, in the context of continuously running cloud services, run-time behaviors of all kinds must be monitored to enable early detection of quality-of-service problems, such as performance degradation, and also the fulfillment of service level agreements [vHHW+09]. This should be done through automated tests, gates and checks [NS13].

Study [21] defines continuous monitoring as a practice whose contribution to the DevOps environment hinges on the presence of another factor, namely, measurement. Once metrics in measurement have been set, continuous monitoring commences. Continuous improvement in DevOps is enabled by constantly monitoring processes and measuring them against metrics [dF17].

Furthermore, study [21] expresses the importance of continuous monitoring by citing findings of other studies. Katal et al. [KBD19] state that DevOps success possibly hinges on how a system is monitored, as monitoring aids in understanding a system inside-out. Riungu-Kalliosaari et al. [RK16] identified that developers emphasized real-time monitoring as a factor that helped create fault-aware systems and that well-timed feedback from monitoring enabled continuous delivery of software. Feijter et al. [dF17] outline how monitoring for incidents aids in continually improving a product inherent to continuous improvement. Similarly, Rafi et al. [RYA20] cited that 'Monitoring automation for continuous delivery' was an essential factor in DevOps and a suitable solution to delivering products on time. Therefore, continuous monitoring contributes to a software organization's DevOps environment by enabling continuous improvement [GH23].

Study [26] defines continues monitoring as the instrumentation of an application and the aggregation of monitored data into insights. This involves having a continuous feedback loop that runs from the production environment to the start of the development cycle, including a complete timeline of development and operations events. It involves proactive detection and awareness of events in critical environments, such as test and production, in order to expose (know the state of) issues before they cause failures [LKO16].

Determining and implementing monitoring metrics of a system is done by developers and operations alike and can include activities like [LKO16]:

- Using a small subset of high-priority test cases executed in production to actively monitor the environment
- Effective instrumentation of software by development in collaboration with operations to give information about its health and performance
- The use of aids, such as feature flags, to quickly recover code failures in production

Study [49] defines continuous monitoring as the practice of observing quality parameters throughout and hence the ability to react to any surprises in timely manner. This is enabled by testing early and on a production-like system [Vir15].

4.4.6 Collaboration and Communication

While collaboration and communication are mentioned in 36 out of 53 selected studies, definitions for them can only be found in study [21]. The other 35 studies, despite mentioning the practice by name, do not give a definition or description of it.

4.4.6.1 Collaboration

Study [21] defines collaboration as a cultural success factor of DevOps which focuses on working together and the exchange of knowledge and experience in and between teams [GH23]. DevOps is all about connecting development and operations, which entails collaboration [SG18]. Advantages of a strong culture of collaboration include enhanced development and operations processes, as well as improved quality of software services and products [dFJT16]. Cross-functional collaboration can result in competitive advantages [Wie18]. Furthermore, collaboration leads to significant business improvements, enhanced operation software quality, and better-quality IT services [vBdKT19].

4.4.6.2 Communication

Study [21] defines communication as a cultural success factor of DevOps which focuses on the communication between the operations and development teams, which is essential for DevOps to succeed [AMS+23]. For the teams to work together effectively, information within and between the teams needs to be shared effectively and frequently. The presence of quality communication helps establish a collaborative culture which contributes to a successful DevOps environment [GH23]. Communication is crucial for DevOps to succeed because not sharing important information may have a significant negative impact, therefore, managers must impose regular communication [AMS+23].

4.4.7 Cloud Computing

While cloud computing is mentioned in 33 out of 53 selected studies, definitions for cloud computing can only be found in 3 of them, namely studies [24], [40], and [52]. The

other 30 studies, despite mentioning the practice by name, do not give a definition or description of it.

Studies [24] and [52] use a cloud computing definition provided by the National Institute of Standards and Technology (NIST). It describes cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [Clo11]. Hence, although cloud computing is sometimes regarded as a type of outsourcing or contrasted with on-premises infrastructure, it is a model for organizing shared computing resources such that automatic, on-demand access to these resources is possible, irrespective of who owns them and whether they are located "on or off premises" [KLJ18].

The essential characteristics of cloud computing according to the NIST definition are [Clo11]:

- On-demand self-service
  A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- Broad network access
  Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- Resource pooling
  The provider's computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
- Rapid elasticity
  Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
- Measured service
  Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource use can be monitored, controlled and reported.

Furthermore, cloud computing comprises three service models [Clo11] [KLJ18]:

- Software-as-a-Service (SaaS): cloud-based applications

- Platform-as-a-Service (PaaS): cloud-based application environments
- Infrastructure-as-a-Service (IaaS): cloud-based hardware

Study [40] defines cloud computing as a paradigm that allows to acquire pay per use and seemingly infinite resources on demand. Together with agile methodologies and DevOps culture the cloud can help organizations to innovate and quickly respond to market demand for continuous delivering of services. Some of the most essential requirements to make use of the benefits of cloud computing are fully automated provisioning, configuration management, and deployment into different environment, such as QA, pre-prod, prod, etc [Son15].

4.4.8 Continuous Feedback

While continuous feedback is mentioned in 23 out of 53 selected studies, a definition can only be found in study [24], which deals with continuous feedback in depth. Meanwhile the other 22 studies, despite mentioning the practice by name, do not give a definition or description of it.

According to study [24] feedback denotes "information about actions returned to the source of the actions" [KLJ18]. Feedback includes internal feedback (i.e. feedback that actors directly perceive from monitoring the outcomes of their actions) and external feedback (i.e. feedback provided by another person) [BW95]. Feedback processes operate during many software development activities, such as compiling, deploying, testing, experimenting, and reviewing [KLJ18].

Continuous feedback tries to reduce the waiting time for feedback through time-boxed iterations [Vid09], which are seen as learning cycles or feedback loops [Sch04]. The shorter these cycles are, the more can be learned [PP04]. Using Continuous integration and Continuous delivery/deployment (CI/CD) technology allows small code changes to trigger the feedback process [Hum10]. Such technology automatically integrates changes into the shared code base and automatically tests and deploys them [KLJ18].

4.4.9 Code Versioning

While code versioning is mentioned in 23 out of 53 selected studies, definitions or descriptions for code versioning can only be found in 3 of them, namely studies [8], [17], and [39]. The other 20 studies, despite mentioning the practice by name, do not give a definition or description of it.

Study [8] defines version control, also known as source control, as the practice of tracking and managing changes to software code. Version control systems are software tools in which developers manage changes to source code over time. The most used and popular version controlling system with CI/CD approaches is Git. A repository is a central place in which an aggregation of data is kept and maintained in an organized way, usually in computer storage. Semantic Versioning is a widely adopted version scheme; it uses a

three-part version number (Major. Minor. Patch), an optional pre-release tag, and an optional build meta tag [ZND18].

Study [17] describes version control as the solution to the problem of code submission and merging of different members. Members are required to pull the latest code from the code base and combine it with the local code every day to implement development work. After completing a task and before submitting a code to the code base, one needs make sure that the submitted code is runnable without any bugs, as this would affect others' development work and operation of the entire system otherwise. Furthermore, the number of code submissions of the entire team and each developer can be viewed on a periodic basis, thereby making the contribution data of team members clear at a glance, which facilitates convenient and intuitive team work assessment [GGL21].

Study [39] defines a Version Control System (VCS) as the management of changes to any kind of document, but it is usually used to control and manage software source code or scripts [Atl24]. In addition, a VCS is also known as revision control or source control.

There are two types of version control systems: Centralized VCS and distributed VCS. A centralized VCS is a client-server approach. The client checks out a copy from the centralized repository. All these data and history is stored and retrieved from the central repository. On the other hand, we have the distributed VCS which is a peer to peer approach. Over copy is a repository and contains full project history [Git14]. Network connection is needed when it is to be shared.

Version controlling code provides several benefits [SDW18]:

1. The ability to work on the same project by more than one person at the same time. Each person will have a copy of the source code to do whatever changes required, then changes are merged with the main code, after the necessary approvals.

2. The history stored by a version control system would allow a developer to examine changes to faulty code.

3. The code can be branched. Developers can work on separate branches of the source code to create and test new features. In the end, a branch can be merged with the main branch (called master).

4.4.10 Continuous Experimentation

While continuous experimentation is mentioned in 21 out of 53 selected studies, definitions for continuous experimentation can only be found in 3 of them, namely studies [12], [13], and [21]. The other 18 studies, despite mentioning the practice by name, do not give a definition or description of it.

Continuous experimentation is often used synonymously to continuous innovation. While some studies distinguish between the two practices, others see them as the same.

Studies [12] and [13] are both written by Brian Fitzgerald and Klaas-Jan Stol. While study [13], which was written in 2014, only mentions continuous innovation, study [12], which was written in 2017, adds continuous experimentation as a separate practice.

Both studies define continuous innovation as a sustainable process that is responsive to evolving market conditions and based on appropriate metrics across the entire lifecycle of planning, development and run-time operations [FS14] [FS17].

An early activity in the continuous innovation space was that of beta testing, which became a widespread practice in the software industry, where it was used to elicit early customer feedback prior to formal release of software products [Col01]. The concept has matured considerably over the years, and now techniques such as A/B testing are widely used where features such as text, layouts, images and colors are manipulated systematically and customer reaction is monitored [OAB12]. This can be an effective way to identify value-adding features. Furthermore, planning has been identified as a prerequisite for continuous innovation, in that inadequate planning and strategic alignment at the front-end of the development process is a major cause of failure for consumer products companies [FS14] [FS17].

Study [12] further defines continuous experimentation as a software development approach based on experiments with stakeholders consisting of repeated Build-Measure-Learn cycles [FS17].

Study [21] defines continuous experimentation as the continual testing of propositions to determine the benefits for the customer and the software organization [RK16]. Experimentation is one of the attributes DevOps teams must exhibit to support DevOps capabilities [SBO18b]. Another aspect of continuous experimentation that positively impacts a DevOps environment is giving developers time to experiment with new ideas, driving the discovery process, and inspiring team members to share their discoveries with the rest of the organization [H+16]. In addition, an experimental culture involves more customer interactions [RK16].

An in-depth analysis by Lwakatare [Lwa17] of their primary studies revealed continuous and rapid experimentation as one of the recurring factors in continuous deployment. Furthermore, Sánchez-Gordón and Colomo-Palacios [SG18] identified experimentation as one of the characteristics of a healthy DevOps environment.

4.4.11 Infrastructure Automation

While infrastructure automation, or Infrastructure as Code (IaC), is mentioned in 19 out of 53 selected studies, definitions for infrastructure automation can only be found in 4 of them, namely studies [21], [26], [29] and [52]. The other 15 studies, despite mentioning the practice by name, do not give a definition or description of it.

Study [21] defines infrastructure automation as the provision of infrastructure elements as code and the configuration of application servers using pre-set automation logic [H+16],

in order to improve the consistency of infrastructure, team productivity, and repeatability of activities [dFJT16].

Study [26] defines IaC as a practice central to automation, which entails treating infrastructure the same way developers treat code. This includes making the process of infrastructure provisioning, orchestration and application deployment reproducible and programmatic. Application code has a defined format and syntax that follows the rules of the programming language in order to be created. Typically, the code is stored in a version-management system that logs a history of code development, changes and fixes. When code is compiled (built) into an application, the expectation is that a consistent application is created. When the latter occurs, the build process is said to be repeatable and reliable. Practicing IaC means applying the same rigor of application code development to infrastructure provisioning, orchestration and deployment. All configurations should be defined in a declarative way and stored in a version management system, just like application code. With IaC, it becomes possible to quickly provision application environments and deploy application automatically and at scale as needed [LKO16].

Study [29] defines IaC as a technique in which the environment required for production or testing is created from tools that use configuration files. These configuration management tools are the main resources for implementing IaC strategies [SLC+19].

Study [52] defines IaC as a principle which denotes that infrastructure is managed and provisioned by code, not by manual processes. Configuration files contain all infrastructure specifications, ensuring that the same environment is provisioned every time. It aims at minimizing human interaction in the build-deploy cycle, which thereby becomes faster, less error-prone and better auditable [Mer21].

4.4.12 Microservice Architecture

While microservice architecture is mentioned in 14 out of 53 selected studies, a description for it can only be found in study [28]. The other 13 studies, despite mentioning the practice by name, do not give a definition or description of it.

According to study [28] microservice architectures have many desirable properties from a DevOps point of view, as they introduce high agility, resilience, scalability, maintainability, separation of concerns, and ease of deployment and operations [AAE16]. Furthermore, due to the greater amount of modularization that microservices provide, new features can be placed in the hands of users faster [New21]. The benefits emphasize the reason why over the last decade, leading software development and consultancy companies have found this software architecture to be an appealing approach that leads to more productive teams in general and to more successful software products [KM18].

Even though microservices release the rigid structure of monolithic systems due to the independent deployment, this style also introduces a high level of complexity. It is more and more demanded to deploy new code quickly, while guaranteeing reliability

and effectiveness. As microservices have more integration points between systems, they suffer from a higher possibility of failure than monolithic systems. It is possible to partially mitigate this problem by making an effort to monitor the services and take appropriate action to malfunctions. Hence, many different monitoring approaches and service discovery mechanisms are developed that pull the status of services dynamically over the network [KM18]. This simplifies many aspects of tracking the system, but lacks in connecting their obtained monitoring data in order to achieve an integrated and holistic view of the behavior and status of the whole microservice infrastructure [BGP11]. In addition, monitoring systems are mostly used in the production environment in order to react quickly to production failures. It is hardly considered to apply monitoring already in the development and testing phase for documentation and in particular architecture comparison in order to receive valuable feedback through the delivery pipeline [KM18].

4.4.13 Continuous Planning

While continuous planning is mentioned in 9 out of 53 selected studies, definitions for continuous planning can only be found in 3 of them, namely studies [12], [13], and [49]. The other 6 studies, despite mentioning the practice by name, do not give a definition or description of it.

Studies [12] and [13] are both written by Brian Fitzgerald and Klaas-Jan Stol and define continuous planning as a holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution [FS14] [FS17]. In addition to iteration and release planning, product and portfolio planning activities would also be conducted [Ruh10].

They clarify that when addressing an ongoing planning problem, time is divided into a number of planning horizons, each lasting a significant period of time. The only form of continuous planning is that which emerges from agile development approaches and is related to sprint iterations or at best, software releases, and is not widespread throughout the organization. However, just as Agile seeks to enable software development to cope with frequent changes in the business environment, the nature of the business environment also requires that planning activities should be done more frequently to ensure alignment between the needs of the business context and software development [LKVK09], and also requires a tight integration between planning and execution [Cas01].

Study [49] describes continuous planning as follows. As businesses plans have to be agile i.e they need to be able to adjust quickly to the changing market conditions, it is important to have interim checkpoints in plans to reassess the situation and modify/adjust the plans as needed based on market feedback. By having a prioritized product backlog, a continuous channel of feedback with customers and the ability to prioritize the product backlog continuously, the business angle can directly be taken into consideration at all times [Vir15].

This can be achieved using a continuous process of [Vir15]:

1. Planning in small portions
2. Executing the plan
3. Collecting feedback
4. Reacting to feedback
5. Adjusting the plan if needed

4.4.14 Quality Assurance

While quality assurance is mentioned in 8 out of 53 selected studies, a description for it can only be found in study [17]. The other 7 studies, despite mentioning the practice by name, do not give a definition or description of it.

According to study [17] quality assurance helps to ensure that software is delivered not only fast, but also the overall quality of the delivered products is guaranteed. Problems and defects should be found in time, before having a negative impact on production [GGL21].

The following actions can help to achieve that goal [GGL21]:

1. Local code analysis:
   Before submitting the code, each developer can perform a static analysis to analyze the code quality locally and find the code that contains errors or does not meet the defined code style standards and has potential problems.
2. Unit tests:
   Each developer can compile the corresponding unit test code and submit it after the local test is passed. These unit tests can also be used in the integration stage and be added the automated delivery pipeline.
3. Automation tests:
   An automation test can be carried out in the continuous integration stage. According to the definition of the CI/CD pipeline, the corresponding test script is executed to perform module function tests and interface tests. After the tests are passed, it enters the next stage. Before the code is integrated, if the system captures an error, an automatic rollback will be triggered.
4. Automated code quality analysis:
   Automated code quality analysis can be used to generate code quality reports. When the previously performed tests are passed, the quality of the startup code is comprehensively scanned and analyzed. The scan results will present data like the number of bugs, defects, coverage, repetition rate in the form of reports.

5. Code review:
   In addition to automation tests and automated code quality analysis, a code review link can be set up. After the code is submitted, a dedicated person reviews the submitted code to check whether there are logic or business problems, or other coding deficiencies. After problem submissions, the code is fed back to the corresponding developers for improvement.

6. System tests:
   System tests are mainly used to test whether the system meets the users' needs. After a series of previous tests, the user environment is simulated to conduct a complete test of the system. System tests can use a combination of manual and automated tests.

4.4.15 Practices without Definitions in Selected Studies

There are a number of practices that, despite being mentioned as DevOps practices in several of the selected studies, were not defined in any of them. These practices are the following:

- Configuration Management Automation
- Sandboxing
- Test Driven Development (TDD)
- Behavior Driven Development (BDD)
- Process Standardization
- Static Code Analysis

CHAPTER 5

Process Definition

This chapter examines the previously selected DevOps practices towards their suitability to be used in connection with Scrum. Firstly, Chapter 5.1 examines the selected DevOps practices individually. Afterwards Chapter 5.3 goes into detail about the synergy between the practices themselves.

5.1 DevOps practices in conjunction with Scrum

This chapter examines the previously selected DevOps practices (see Chapter 4.4) towards their suitability to be used in connection with Scrum.

Firstly, a valid practice definition in the context of this thesis is chosen, based on the different practice definitions found in Chapter 4.4). If no definition was found during the systematic literature review, one will be added from different sources. Secondly, it is described if and how these DevOps practices can extend Scrum in order to reduce the risk of a gap between development and operations within the software engineering process.

5.1.1 Continuous Integration

Practice Definition

Within this thesis continuous integration is defined as a developer practice where developers need to integrate their work into a shared repository frequently. Usually each person integrates at least daily, leading to multiple integrations per day [DSM+22] [FS17] [FS14] [GBM19] [MMA+20] [Son15] [SMB17] [VZR+17] [Vir15]. When code is added to the code base, interconnected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking compliance with coding standards, and building deployment packages are automatically triggered, so teams can quickly

collaborate on a single project [DSM+22] [FS17] [FS14] [GS21] [GGL21] [GH23] [MMA+20] [PEL21] [SDW18] [Son15] [VZR+17] [Vir15]. Furthermore, the integration process of developer's code needs to detect errors quickly and avoid failure of builds. Any continuous integration failure is also an important event which may have a number of ceremonies and highly visible artifacts to help ensure that problems leading to integration failures are solved as quickly as possible by those responsible [AR19] [FS17] [FS14] [GGL21] [VZR+17].

Integration with Scrum

Continuous integration can be integrated in a Scrum process very well. Using Scrum the project is already broken down into small and manageable tasks which are worked on by the developers producing thoroughly verified increments. By integrating these increments more frequently into a common code base and adding the previously described automations, continuous integration can be achieved.

5.1.2 Continuous Delivery

Practice Definition

Within this thesis continuous delivery is defined as a development practice where every change is treated as a potential release candidate to be frequently and rapidly evaluated through one's continuous delivery pipeline. This typically involves activities such as code analysis, documentation generation, acceptance testing, regulatory compliance assessments, license scanning and requirement verification [DSM+22] [GS21] [GBM19] [GH23] [SMB17] [TSM19b] [TSB21] [VZR+17]. Using continuous delivery, software always needs to be in a release state, so that it is always possible to deploy and/or release the latest working version [AR19] [Son15] [SMB17] [TSM19b] [TSB21]. Furthermore, software is automatically deployed and delivered to a production-like environment [DSM+22] [FS17] [SDW18] [TSM19b] [TSB21]. As this production-like environment matches a definition for a sandbox [Tec24] [Ven05] and no specific definition for sandboxes in the context of DevOps was found during the systematic literature review, the two practices are merged within this thesis.

Integration with Scrum

As Scrum increments are completed developments additive to all prior increments they can always be deployed by themselves, thereby being ideal to work in conjunction with continuous delivery. By implementing a continuous delivery pipeline and evaluating all finished increments through it, each increment is treated as a potential release candidate and it is thereby always possible to deploy and/or release the latest working version. Furthermore, using continuous delivery each evaluated increment can automatically be deployed to a working test environment, allowing stakeholders to test the increment as soon as possible and give valuable feedback without needing to wait for the end of the sprint.

5.1.3 Continuous Deployment

Practice Definition

Within this thesis continuous deployment is used synonymously to continuous release and is defined as the process of deploying to production as soon as qualified changes in software code or architecture are ready, and without human intervention. Continuous deployment implies continuous delivery, therefore, release candidates evaluated and approved in continuous delivery are automatically placed in a production environment [AR19] [DSM+22] [FS17] [GGL21] [GBM19] [GH23] [MMA+20] [SDW18] [SMB17] [Vir15].

Integration with Scrum

There are several possibilities on how to use continuous deployment in connection with Scrum, depending on the use case of the delivered product.

As Scrum usually deploys code at the end of a sprint, the continuous deployment process can be triggered once the sprint review is done and the outcome of the sprint is inspected, thereby automatically deploying code to production once per sprint cycle. This might be a scaled-down approach to continuous deployment, as not every increment is automatically and instantly deployed to production, however, the manual overhead of deploying to production is still massively reduced and as continuous deployment implies continuous delivery, each evaluated increment can still automatically be deployed to a working test environment, allowing stakeholders to test the increment as soon as possible and give valuable feedback without needing to wait for the end of the sprint.

Alternatively, if the use case of the delivered product allows it, the continuous deployment process can also be triggered every time an increment is integrated into a common code base, directly pushing the code to production as soon as it is evaluated. The total progress of the sprint can still be discussed during the sprint review, despite the code already being deployed to production.

5.1.4 Continuous Testing

Practice Definition

Within this thesis continuous testing is defined as a process of testing early, testing often, testing everywhere, and automating [AR19]. This is done by building a test suite which can automatically be executed on every software build generated without any user intervention [Vir15] in order to verify the correctness, quality and performance of developed software [AMB+16] and promptly assess the business risks linked to a software release candidate [GHM18]. This test suite can then be integrated into the software delivery pipeline [Vir15]. All this reduces manual testing, accelerates the testing process significantly [H+16], reduce the time between the introduction of errors and their detection, with the aim of eliminating root causes more effectively [BSMG12] and strongly affects deployment speed [SBO18b].

Integration with Scrum

Continuous testing and Scrum can work together well. One of the main duties of developers working in a Scrum team is to instill quality by adhering to a Definition of Done (DoD). Continuous testing can help achieve this task. By writing a test suit covering the DoD and continuously testing the produced code against it, a better test coverage can be achieved and the overall stability of the product is improved. Furthermore, as tests of previous increments are continuously run on the current code base, unwanted ramifications of new developments on preexisting functions can be detected and thereby avoided early. Finally, as manual testing is reduced and the testing process is significantly accelerated, more time can be spent on other tasks.

5.1.5 Continuous Monitoring

Practice Definition

Within this thesis continuous monitoring is defined as the instrumentation of an application and the aggregation of monitored data into insights. This involves having a continuous feedback loop that runs from the production environment to the start of the development cycle, including a complete timeline of development and operations events. It involves proactive detection and awareness of events in critical environments, such as test and production, in order to expose (know the state of) issues before they cause failures [LKO16], enable early detection of quality-of-service problems, such as performance degradation, and also the fulfillment of service level agreements [vHHW+09]. Continuous improvement in DevOps is enabled by constantly monitoring processes and measuring them against metrics [dF17] [GH23].

Determining and implementing monitoring metrics of a system is done by developers and operations alike and can include activities like [LKO16]:

- Using a small subset of high-priority test cases executed in production to actively monitor the environment
- Effective instrumentation of software by development in collaboration with operations to give information about its health and performance
- The use of aids, such as feature flags, to quickly recover code failures in production

Integration with Scrum

Two of the three empirical Scrum pillars are inspection and adaptation. Where inspection means to frequently and diligently inspect all Scrum artifacts to detect potentially undesirable variances or problems and adaption means adjusting the process, if any aspects of it deviate outside acceptable limits or if the resulting product is unacceptable [SS20]. These Scrum pillars are congruent with the goals of continuous monitoring, as continuous monitoring implements monitoring metrics in order to control the system and detect deviations or abnormalities early (inspection) and uses these measurements in

order to promote continuous improvement (adaptation). Continuous monitoring, and subsequently continuous improvement, are therefore a concrete practice to implement these vague ideas.

5.1.6 Collaboration and Communication

As only one definition for collaboration and communication was found during the systematic literature review, this definition is also adopted by this thesis.

5.1.6.1 Collaboration

Practice Definition

Within this thesis collaboration is defined as a cultural success factor of DevOps which focuses on working together and the exchange of knowledge and experience in and between teams [GH23]. DevOps is all about connecting development and operations, which entails collaboration [SG18]. Advantages of a strong culture of collaboration include enhanced development and operations processes, as well as improved quality of software services and products [dFJT16]. Cross-functional collaboration can result in competitive advantages [Wie18]. Furthermore, collaboration leads to significant business improvements, enhanced operation software quality, and better-quality IT services [vBdKT19].

Integration with Scrum

As collaboration is a cultural success factor it does not change the flow of the Scrum process, but incorporating it still has an impact by extending boundaries of the team and the circle of involved stakeholders. As collaboration focuses on working together and the exchange of knowledge and experience in and between teams, it is already partially congruent with Scrum, as Scrum places a strong focus on transparency and sharing information and knowledge within the team. Using Scrum, teams are already self-managing, self-organizing and cross-functional in order to maximize team performance, meaning the Scrum team is already a self-contained group of people with different functional expertise working toward a common goal [Sch04] [SS20], therefore enforcing collaboration within the team and reducing external dependencies. As DevOps is all about connecting development and operations, these two point can tie together by adding an employee with operational expertise to this cross-functional team, thereby adding an operations point of view to the ongoing software development process. This employee does not necessarily need to be a developer, as providing a basic infrastructure might still be done in a separate team, even when working with DevOps, but using this infrastructure can move inside the Scrum team by employing the right personnel.

In order to fully implement a culture of collaboration, lessons learned within the Scrum team also need to be communicated to the rest of the organization, so that other teams can learn from the Scrum team and adapt their process improvements. Furthermore, by implementing a strong culture of collaboration the Scrum team itself can profit from

lessons learned by other teams and can take feedback from other departments into account, which are directly affected by the teams output.

5.1.6.2 Communication

Practice Definition

Within this thesis communication is defined as a cultural success factor of DevOps which focuses on the communication between development and operations, which is essential for DevOps to succeed [AMS+23]. For the teams to work together effectively, information within and between the teams needs to be shared effectively and frequently. The presence of quality communication helps establish a collaborative culture which contributes to a successful DevOps environment [GH23]. Communication is crucial for DevOps to succeed because not sharing important information may have a significant negative impact, therefore, managers must impose regular communication [AMS+23].

Integration with Scrum

As communication is a cultural success factor it does not change the flow of the Scrum process, but incorporating it still has an impact by extending boundaries of the team and the circle of involved stakeholders. As communication focuses on effectively and frequently sharing information between development and operations, operations should be seen as a crucial stakeholder of a successful software project. Therefore, an employee with operational expertise should be added to the existing cross-functional Scrum team, thereby adding an operations point of view to the ongoing software development process, giving valuable feedback during planning and development and thereby eliminating misalignment between development and operations before it occurs. This employee does not necessarily need to be a developer, as providing a basic infrastructure might still be done in a separate team, even when working with DevOps, but using this infrastructure can move inside the Scrum team by employing the right personnel.

5.1.7 Cloud Computing

Practice Definition

Within this thesis the cloud computing definition provided by the NIST is used. It describes cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [Clo11]. Hence, although cloud computing is sometimes regarded as a type of outsourcing or contrasted with on-premises infrastructure, it is a model for organizing shared computing resources such that automatic, on-demand access to these resources is possible, irrespective of who owns them and whether they are located “on or off premises” [KLJ18].

The essential characteristics of cloud computing according to the NIST definition are [Clo11]:

- On-demand self-service
  A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

- Broad network access
  Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

- Resource pooling
  The provider's computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

- Rapid elasticity
  Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.

- Measured service
  Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource use can be monitored, controlled and reported.

Furthermore, cloud computing comprises three service models [Clo11] [KLJ18]:

- Software-as-a-Service (SaaS): cloud-based applications
- Platform-as-a-Service (PaaS): cloud-based application environments
- Infrastructure-as-a-Service (IaaS): cloud-based hardware

Integration with Scrum

Cloud computing is an operations practice. Using cloud computing in order to manage an application can be seen as a developers task in the context of Scrum and it can therefore work in conjunction with Scrum. As cloud computing still needs to be set up, managed and the computing resources need to be configured, its usage does not make operations obsolete, but rather changes the resources they are working with. Depending on the size of the company these operations tasks might either be done by an employee with operational expertise who is part of the cross-functional Scrum team, or by a separate team, tasked with providing a basic infrastructure.

5.1.8 Continuous Feedback

Practice Definition

As only one definition for continuous feedback was found during the systematic literature review, this definition is also adopted by this thesis. Therefore, within this thesis feedback denotes "information about actions returned to the source of the actions" [KLJ18] and includes internal feedback (i.e. feedback that actors directly perceive from monitoring the outcomes of their actions) and external feedback (i.e. feedback provided by another person) [BW95]. Feedback processes operate during many software development activities, such as compiling, deploying, testing, experimenting, and reviewing [KLJ18].

Continuous feedback tries to reduce the waiting time for feedback through time-boxed iterations [Vid09], which are seen as learning cycles or feedback loops [Sch04]. The shorter these cycles are, the more can be learned [PP04]. Using CI/CD technology allows small code changes to trigger the feedback process [Hum10]. Such technology automatically integrates changes into the shared code base and automatically tests and deploys them [KLJ18].

Integration with Scrum

A Scrum process by definition already uses continuous feedback. A key pillar of Scrum is transparency, where the emergent process and work must be visible to those performing the work as well as those receiving the work. Consequently Scrum employs several events promoting continuous feedback. The daily Scrum is about updating the team daily on what has been accomplished, what needs to be done next and what obstacles developers are currently presented with. The sprint review is about presenting the results of the teams work to key stakeholders, discussing progress toward the product goal and consequently collaborating on what to do next. Finally, the sprint retrospective is about planning ways to increase quality and effectiveness. Therefore, the daily Scrum and the sprint retrospective present internal feedback, while the sprint review presents external feedback [SS20].

Continuous feedback is all about shortening learning cycles or feedback loops. Within a Scrum process continuous feedback can be further improved by using CI/CD technology, which allows small code changes to trigger the feedback process. Thereby, feedback on the current work can be received and incorporated much sooner.

5.1.9 Code Versioning

Practice Definition

Within this thesis version control, also known as source control, is the management of changes to any kind of document, but it is usually used to control and manage software source code or scripts [Atl24].

There are two types of version control systems: Centralized VCS and distributed VCS. A centralized VCS is a client-server approach. The client checks out a copy from the

centralized repository. All these data and history is stored and retrieved from the central repository. On the other hand, we have the distributed VCS which is a peer to peer approach. Over copy is a repository and contains full project history [Git14]. Network connection is needed when it is to be shared.

Version controlling code provides several benefits:

1. The ability to work on the same project by more than one person at the same time. Each person will have a copy of the source code to do whatever changes required, then changes are merged with the main code, after the necessary approvals [SDW18].
2. The history stored by a version control system would allow a developer to examine changes to faulty code [SDW18].
3. The code can be branched. Developers can work on separate branches of the source code to create and test new features. In the end, a branch can be merged with the main branch (called master) [SDW18].
4. The number of code submissions of the entire team and each developer can be viewed on a periodic basis, thereby making the contribution data of team members clear at a glance, which facilitates convenient and intuitive team work assessment [GGL21]

The most used and popular version controlling system with CI/CD approaches is Git [ZND18].

Integration with Scrum

Code versioning is a practice enabling development teams to manage changes made to the source code or scripts, enabling more than one person to work on a project at the same time and tracking the codes history. Using code versioning does not impact the overall workflow of a Scrum team and can therefore easily be used as a development practice by developers of a Scrum team.

5.1.10 Continuous Experimentation

Practice Definition

Within this thesis continuous experimentation is used synonymously to continuous innovation and is defined as the continual testing of propositions to determine the benefits for the customer and the software organization [RK16]. Furthermore, techniques such as A/B testing, where features such as text, layouts, images and colors are manipulated systematically and customer reaction is monitored, can be used to effectively identify value-adding features [OAB12]. Continuous experimentation consists of repeated Build-Measure-Learn cycles [FS17] and is one of the attributes DevOps teams must exhibit to

support DevOps capabilities [SBO18b]. Another aspect of continuous experimentation that positively impacts a DevOps environment is giving developers time to experiment with new ideas, driving the discovery process, and inspiring team members to share their discoveries with the rest of the organization [H+16]. In addition, an experimental culture involves more customer interactions [RK16].

Integration with Scrum

Continuous experimentation can easily be integrated into a standard Scrum process. In order for continuous experimentation to work developers need to be given the time to experiment with new ideas, driving the discovery process, and inspiring team members to share their discoveries with the rest of the organization. During sprint planning this time can be allocated to the developers. At the end of the sprint, during sprint retrospective, developers then have the time to reflect on the changes and their outcome. Consequently, the changes can be discarded or permanently added to the workflow.

5.1.11 Infrastructure Automation

Practice Definition

Within this thesis infrastructure automation, also known as Infrastructure as Code (IaC), is defined as the provision of infrastructure elements as code and the configuration of application servers using pre-set automation logic [H+16], in order to accelerate build-deploy cycles [Mer21], improve the consistency of infrastructure, team productivity, and repeatability of activities [dFJT16]. As this matches the definition for configuration management automation [TV14], which was not found during the systematic literature review and therefore added from different sources, the two practices are merged within this thesis.

Application code has a defined format and syntax that follows the rules of the programming language in order to be created. Typically, the code is stored in a version-management system that logs a history of code development, changes and fixes. When code is compiled (built) into an application, the expectation is that a consistent application is created. When the latter occurs, the build process is said to be repeatable and reliable. Practicing IaC means applying the same rigor of application code development to infrastructure provisioning, orchestration and deployment. All configurations should be defined in a declarative way and stored in a version management system, just like application code. With IaC, it becomes possible to quickly provision application environments and deploy application automatically and at scale as needed [LKO16].

Integration with Scrum

Infrastructure automation is an operations practice used to accelerate and stabilize application environment provisioning and automate application deployment. As infrastructure automation is done by applying the same rigor of application code development to infrastructure provisioning, orchestration and deployment, setting it up can be seen as

a developers task in the context of Scrum and infrastructure automation can therefore work in conjunction with Scrum.

A set up infrastructure automation makes it much easier and faster to deploy an application, therefore the individual tasks of deploying a new version of an application can be redistributed. Depending on the size of the company setting up infrastructure automation can either be done by an employee with operational expertise who is part of the cross-functional Scrum team, or by a separate team, tasked with providing a basic infrastructure. Either way, usage of the automated application deployment can be done by the developers within the Scrum team. This allows developers to deploy their products faster, without needing to wait on another team to perform deployment tasks.

5.1.12 Microservice Architecture

Practice Definition

As only one description for microservice architecture was found during the systematic literature review and this description focuses more on benefits and downsides than describing what a microservice architecture is, an additional definition was chosen from literature in order to cover the basic ideas. Therefore, within this thesis a microservices architecture is defined as a cloud-native architecture that aims to realize software systems as a package of small services. Each service is independently deployable on a potentially different platform and technological stack. It can run in its own process while communicating through lightweight mechanisms such as RESTful or RPC-based APIs. In this setting, each service is a business capability that can utilize various programming languages and data stores and is developed by a small team [FL15].

From a DevOps point of view microservice architectures have many desirable properties, as they introduce high agility, resilience, scalability, maintainability, separation of concerns, and ease of deployment and operations [AAE16]. Furthermore, due to the greater amount of modularization that microservices provide, new features can be placed in the hands of users faster [New21].

Even though microservices release the rigid structure of monolithic systems due to the independent deployment, this style also introduces a high level of complexity. It is more and more demanded to deploy new code quickly, while guaranteeing reliability and effectiveness. As microservices have more integration points between systems, they suffer from a higher possibility of failure than monolithic systems. It is possible to partially mitigate this problem by making an effort to monitor the services and take appropriate action to malfunctions. Hence, many different monitoring approaches and service discovery mechanisms are developed that pull the status of services dynamically over the network [KM18]. This simplifies many aspects of tracking the system, but lacks in connecting their obtained monitoring data in order to achieve an integrated and holistic view of the behavior and status of the whole microservice infrastructure [BGP11]. In addition, monitoring systems are mostly used in the production environment in order to react quickly to production failures. It is hardly considered to apply monitoring already

in the development and testing phase for documentation and in particular architecture comparison in order to receive valuable feedback through the delivery pipeline [KM18].

Integration with Scrum

Microservice architecture can be used in conjunction with Scrum, having no negative impact on the overall Scrum process. In order to use a microservice architecture the developed product needs to be adjusted accordingly, meaning the developed product needs to be set up as a package of small services. Within a Scrum process these small services can then be worked on individually and incrementally by developers, producing deployable increments to the overall product within a sprint.

5.1.13 Continuous Planning

Practice Definition

Within this thesis continuous planning is defined according to Fitzgerald and Stol, as their definition is the most detailed one found during the systematic literature review and it includes/implies other definitions which were found. Therefore, continuous planning is defined as a holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution [FS14] [FS17]. In addition to iteration and release planning, product and portfolio planning activities would also be conducted [Ruh10].

When addressing an ongoing planning problem, time is divided into a number of planning horizons, each lasting a significant period of time. The only form of continuous planning is that which emerges from agile development approaches and is related to sprint iterations or at best, software releases, and is not widespread throughout the organization. However, just as Agile seeks to enable software development to cope with frequent changes in the business environment, the nature of the business environment also requires that planning activities should be done more frequently to ensure alignment between the needs of the business context and software development [LKVK09], and also requires a tight integration between planning and execution [Cas01].

Integration with Scrum

As defined the only existing form of continuous planning is that which emerges from agile development approaches, ideally related to sprint iterations, which is the case when working with Scrum. Furthermore, continuous planning already is one of the basic ideas of working with Scrum, as the whole Scrum process revolves around continuous planning. Scrum uses sprints as short planning horizons, with sprint reviews and sprint plannings as interim checkpoints where the situation can be reassessed and plans can be modified/adjusted based on market feedback. Furthermore, Scrum uses a prioritized backlog, which is managed by the Product Owner (PO) and can be reordered continuously, following current requirements. When using continuous planning in conjunction with Scrum and DevOps an additional aspect can be focused on in regards to release planning.

As DevOps adds employees with operational expertise to cross-functional Scrum teams, the operational aspects of a release can be considered more diligently and new automations regarding operational tasks can be planned and implemented in an agile manner.

5.1.14 Quality Assurance

Practice Definition

As only one definition for quality assurance was found during the systematic literature review, this definition is also adopted by this thesis. Therefore, within this thesis quality assurance is defined as a practice that helps to ensure that software is delivered not only fast, but also the overall quality of the delivered products is guaranteed. Problems and defects should be found in time, before having a negative impact on production [GGL21].

The following actions can help to achieve that goal [GGL21]:

1. Local code analysis:
   Before submitting the code, each developer can perform a static analysis to analyze the code quality locally and find the code that contains errors or does not meet the defined code style standards and has potential problems.

2. Unit tests:
   Each developer can compile the corresponding unit test code and submit it after the local test is passed. These unit tests can also be used in the integration stage and be added the automated delivery pipeline.

3. Automation tests:
   An automation test can be carried out in the continuous integration stage. According to the definition of the CI/CD pipeline, the corresponding test script is executed to perform module function tests and interface tests. After the tests are passed, it enters the next stage. Before the code is integrated, if the system captures an error, an automatic rollback will be triggered.

4. Automated code quality analysis:
   Automated code quality analysis can be used to generate code quality reports. When the previously performed tests are passed, the quality of the startup code is comprehensively scanned and analyzed. The scan results will present data like the number of bugs, defects, coverage, repetition rate in the form of reports.

5. Code review:
   In addition to automation tests and automated code quality analysis, a code review link can be set up. After the code is submitted, a dedicated person reviews the submitted code to check whether there are logic or business problems, or other coding deficiencies. After problem submissions, the code is fed back to the corresponding developers for improvement.

6. System tests:
   System tests are mainly used to test whether the system meets the users' needs. After a series of previous tests, the user environment is simulated to conduct a complete test of the system. System tests can use a combination of manual and automated tests.

Integration with Scrum

Using Scrum, quality assurance can be integrated as a part of the tasks performed by developers. It does not contradict any parts of the Scrum process, rather it further promotes inspection, one of the key pillars of Scrum, which states that artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems [SS20].

This is particularly true for automated tests like unit tests, automation tests, automated code quality analysis and system tests, as those tests can be written independently from the developed code, describing and testing the desired outcome. As those tests are automated, they can be added to a deployment pipeline and be run frequently, thereby consistently testing whether the developed product deviates from the desired output.

Manual quality assurance tasks like local code analysis and code review can be integrated as a mandatory part of the software development process.

Using and integrating all those tasks into the software development process leads to better software quality and a more stable product. Problems within the product are detected earlier, thereby not negatively effecting the code run in production. Furthermore, it helps to keep the development progress on the right track towards the desired final product.

5.1.14.1 Static Code Analysis

Practice Definition

Static code analysis was found as a separate practice during the systematic literature review, however, as by definition it is also part of another practice, this thesis does not list it separately, treating it as part of quality assurance and adding a more detailed description for it. However, as no more detailed definition for static code analysis could be found during the systematic literature review, one was added from different sources. Within this thesis static code analysis is defined as a way to find bugs and reduce the defects in a software application. Most errors fall into known categories, as people tend to fall into the same traps repeatedly. Therefore, a static analyzer or checker is a program written to analyze other programs for flaws. These type of programs are scanning the source code, the byte code or the binaries of a program in order to match patterns. Static analyzers have the potential to find rare occurrences or hidden back doors [B+10]. Since they consider the code independently of any particular execution, they can enumerate all possible interactions between the different components or modules [Bla09].

Integration with Scrum

The reasoning on how to incorporate quality assurance with Scrum also holds for this subitem.

5.1.15 Test Driven Development (TDD)

Practice Definition

As no definition for Test Driven Development (TDD) could be found during the systematic literature review, one was added from different sources. Within this thesis Test Driven Development (TDD) is defined as an iterative software development methodology that emphasizes writing automated tests before writing actual production code. In contrast to traditional software development where software is produced first and test cases are created later, this approach interlaces programming, the creation of unit tests, and refactoring on the source code. In TDD, developers write a failing test case first, then write the code that passes the test intending to create a suite of tests that ensure the code behaves correctly [Bec03]. When compared to traditional methods, TDD can deliver higher-quality projects in lesser time. TDD implementation needs developers and testers to precisely predict how the program and its features will be utilized in practice. An advantage of TDD is the generation of a regression-test suite as a byproduct; this reduces human manual testing while detecting errors earlier, resulting in faster repairs. Also, as tests are performed at the start of the design cycle in this approach, the time and money spent on debugging at later stages is reduced. This is also why TDD is referred to as test-first development. It needs to be noted that refactoring plays a key role in the TDD process since it allows the improvement in the internal structure of the code and design whilst preserving the external behavior of the code [BCF+21].

TDD is also famously known by the red-green-refactor cycle:

1. Add a test to the test suite
2. (Red) Run all the tests to ensure the new test fails
3. (Green) Write just enough code to get that single test to pass
4. Run all tests
5. (Refactor) Improve the initial code while keeping the tests green
6. Repeat

Integration with Scrum

TDD is a developer practice which alters the order in which developers produce and test software, therefore it can be used in conjunction with Scrum without altering the overall Scrum process. Using TDD developers start software development by first writing tests and developing the actual software afterwards. This also has the benefit of producing a test suit which can be used as a basis for continuous testing.

5.1.16 Behavior Driven Development (BDD)

Practice Definition

As no definition for Behavior Driven Development (BDD) could be found during the systematic literature review, one was added from different sources. Within this thesis Behavior Driven Development (BDD) is defined as a method of software development that involves collaboration among both technical and non-technical team members working on a software product or project [BEK20]. BDD is based on the approaches used in TDD [DKSD14], however, it operates at a higher level than TDD and can be considered as a refinement of TDD as it shifts the focus from testing to the identification of the expected behaviors of a system. Furthermore, it is considered a valuable approach for managing client requirements and scenarios, which are expressed in the form of test scenarios. Test scenarios that are derived through requirements play a very important role for verbal interactions between the agile team and end-user. In BDD, a series of inputs is provided to interact with the application or software system, written in a form of plain language phrases and organized in the defined pattern "Given-When-Then" [Mis17], depicting the details as:

- Given - shows the initial context
- When - presents an occurring of an event
- Then - demonstrates a promise of an outcome as expected

BDD scripting utilizes a pattern that facilitates the automation of testing. Traditional methods of clarifying requirements or reducing ambiguities often results in miscommunication between project stakeholders and the technical team, making direct communication with stakeholders difficult [AI13]. BDD addresses this issue by incorporating a narrative collaboration between all team members and the client.

The three objectives of BDD are events, outcomes and context. Events refer to the actions that take place to achieve the final outcome, which is the expected result. In other words, an event is an action a user performs, and an outcome is the expected result that will be obtained from the action. The primary goal of BDD is to determine the set of behaviors that the system can expect from the user. BDD tests can be written and executed at any stage of the development process, whether it be before, during or after the development process. This results in a specification that is easily understandable by the consumers. The examples in BDD are also executable due to the "glue code" which connects plain human-readable language statements to a test which is a piece of code. This not only defines the program's requirements but also serves as an acceptance test for comparing the implementation to the specification [FOR+23].

Integration with Scrum

As BDD manages client requirements and scenarios by expressing them in the form of test scenarios, collaboration between technical and non-technical team members as well

as stakeholders needs to be deepened in order to incorporate BDD with Scrum. The stakeholders using the finished product need to have a very clear understanding about the products behavior before development is done. This behavior needs to be agreed upon between the stakeholders and the PO before placing the software requirement in the product backlog, so that the PO can later on communicate it to the developers clearly during the sprint planning. After the products behavior is defined BDD can be treated as an extension of TDD, which alters the order in which developers produce and test software. Like with TDD, developers using BDD start software development by first writing tests and developing the actual software afterwards. This also has the benefit of producing a test suit which can be used as a basis for continuous testing.

5.2 Adaptations from SLR Practices

Looking at the practice definitions found during the systematic literature review (see Chapter 4) and adding definitions for practices where no definitions were found, it can be seen that some of the practices overlap or no clear connection to DevOps could be drawn. Therefore, within this thesis, some of the found practices were either merged or discarded (see Figure 5.1).

- Continuous deployment is used synonymously to continuous release
  As only one paper found during the systematic literature review distinguishes between continuous deployment and continuous release and all the other papers use the terms synonymously, this thesis follows the found scientific majority and merges the two practices.
- Sandboxing as part of continuous delivery
  As no definition for sandboxing was found during the systematic literature review one was added from different sources. This definition matches the usage of production-like environments described in continuous delivery and therefore, within this thesis, sandboxing is seen as a part of continuous delivery and the two practices were merged.
- Static code analysis as part of quality assurance
  As no definition for static code analysis was found during the systematic literature review one was added from different sources. This definition matches one of the described actions for quality assurance and therefore, within this thesis, static code analysis is seen as a part of quality assurance and the two practices were merged.
- Infrastructure automation is used synonymously to configuration management automation
  As no definition for configuration management automation was found during the systematic literature review one was added from different sources. This definition matches the definition for infrastructure automation and therefore, within this thesis, the two practices were merged.

- Process standardization was removed
  As no definition for process standardization in the context of DevOps was found during the systematic literature review and different sources defined the practice too vaguely to be used reasonably in the context of DevOps and Scrum, the practice was removed from further consideration within this thesis.

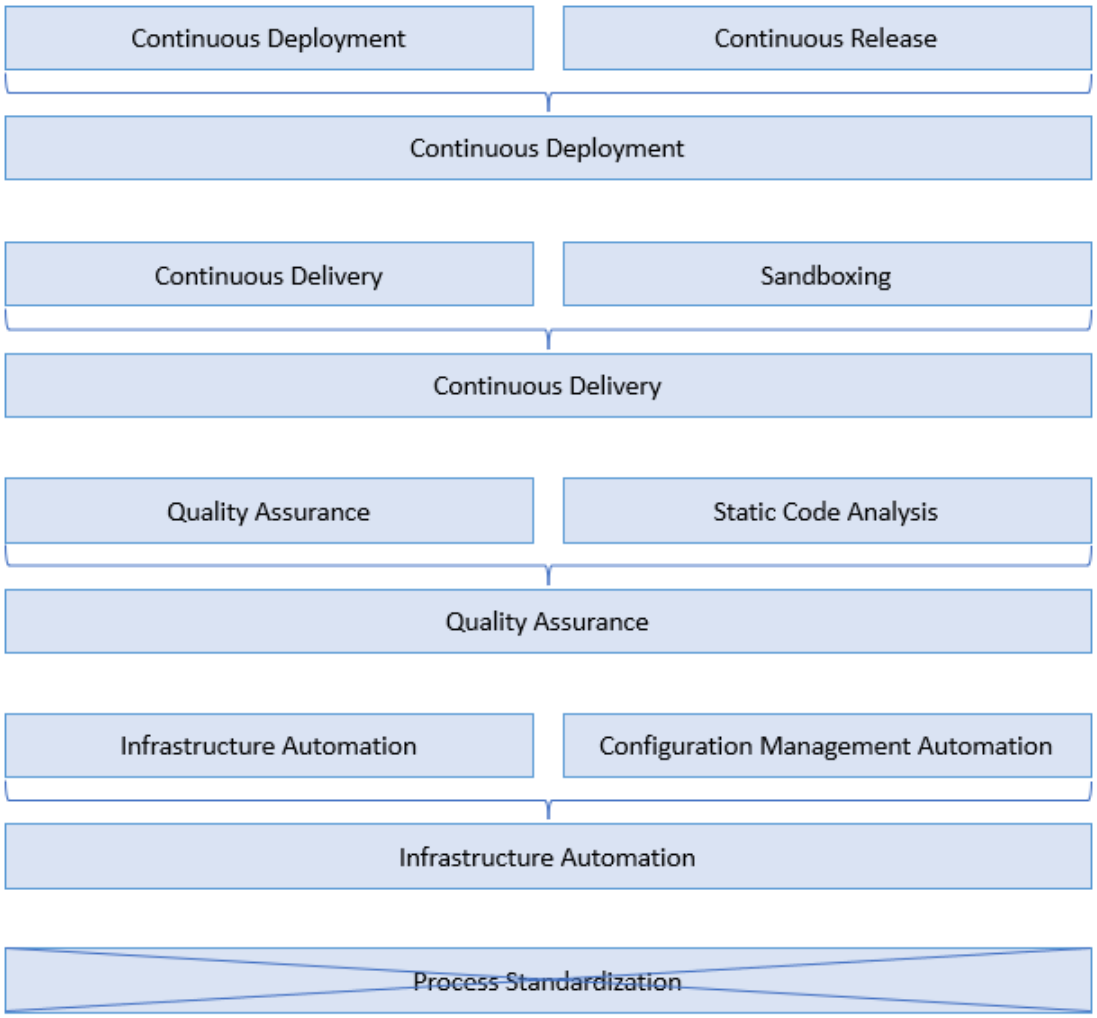

Figure 5.1: Adaptations from SLR practices

5.3 Synergy between Practices

Looking at the practices and their possible integration into a Scrum process it becomes clear that some of them synergize well with each other, if not being prerequisites for each other. This does not mean that they cannot be implemented together with other

practices, but simply that synergies can be utilized when implementing these practices together.

Code versioning is a prerequisite for most other development practices as it separates work done by different people. Increments developed on the product cannot be tested and deployed individually, if they are intertwined with other, possibly unfinished, developments. Furthermore, code versioning enables the historization of code, may it be software code or code being used for infrastructure automation.

Furthermore, microservice architecture synergizes with most other development practices, as it aims to realize software systems as a package of small services and DevOps aims to accelerate software development by deploying smaller, working software packages more often. Setting up the developed product in some sort of microservice architecture makes further automations like testing and deployment much easier.

Continuous integration and continuous delivery or deployment work together by setting up a CI/CD pipeline, which automatically pushes all integrated code changes to production or a production-like environment. In this regard, continuous delivery is a prerequisites for continuous deployment. This also synergizes with continuous testing and automated quality assurance tests, as test suits can be integrated into the CI/CD pipeline. Furthermore, continuous feedback relates to this set of practices as the feedback process can be triggered using CI/CD technology.

Continuous monitoring also synergizes with several other practices. Firstly, monitoring is the basis for feedback, therefore, without continuous monitoring there will not be a continuous feedback loop. Secondly, continuous monitoring enables improvement and is therefore necessary to determine, whether changes made during continuous experimentation had a positive impact. Finally, continuous monitoring synergizes well with cloud computing, as cloud computing already provides measured services which can be used to set up continuous monitoring.

Another set of practices that synergize with each other are cloud computing and infrastructure automation. Cloud computing moves resources off-site, however those resources still need to be managed and configured. SaaS, used for cloud-based applications, and Platform-as-a-Service (PaaS), used for cloud-based application environments, can be managed using infrastructure automation.

Finally, continuous testing synergizes well with Test Driven Development (TDD) and Behavior Driven Development (BDD), as both TDD and BDD focus on writing tests and test suits before the actual code. These tests can then be used for continuous testing.

Table 5.1 visualizes these described synergies. Within this visualization there are 3 different labels describing the type of connection:

- 'X': The two practices generally synergize with each other
- 'E': Practice A enables practice B
  e.g. continuous delivery enables continuous deployment

- 'R': Practice B requires practice A
  e.g. continuous deployment requires continuous delivery

	Continuous Integration	Continuous Delivery	Continuous Deployment	Continuous Testing	Continuous Monitoring	Collaboration & Communication	Cloud Computing	Continuous Feedback	Code Versioning	Continuous Experimentation	Infrastructure Automation	Microservice Architecture	Continuous Planning	Quality Assurance	Test Driven Development	Behavior Driven Development
Continuous Integration		X	X	X				X	R			X		X		
Continuous Delivery	X		E	X				X	R			X		X		
Continuous Deployment	X	R		X				X	R			X		X		
Continuous Testing	X	X	X						R			X		X	X	X
Continuous Monitoring							X	E		E						
Collaboration & Communication																
Cloud Computing					X						X	X				
Continuous Feedback	X	X	X		R					X						
Code Versioning	E	E	E	E							E	E			E	E
Continuous Experimentation					R			X								
Infrastructure Automation							X		R							
Microservice Architecture	X	X	X	X			X		R					X		
Continuous Planning																
Quality Assurance	X	X	X	X								X				
Test Driven Development				X					R							
Behavior Driven Development				X					R							

Table 5.1: Synergies between practices
'X': The two practices generally synergize with each other
'E': Practice A enables practice B
'R': Practice B requires practice A

CHAPTER 6

Process Validation

This chapter aims to validate the practices defined in chapter 5. Firstly, criteria for suitable interview partners are defined such as experience working with or implementing DevOps. Secondly, an interview guideline is defined and used in order to get semi-structured feedback on the chosen practices. Individual feedback is then summarized and structured using a matrix. Finally, the feedback is used in order to update and refine the previously defined set of DevOps practices.

6.1 Survey

As the process defined in Chapter 5 is solely based on literature and not implemented in practice, it is necessary to validate it using feedback from experts. An important part to be clarified during the interviews is importance of the individual practices for a DevOps implementation. Chapter 5 does not state any hierarchy between the practices. After the interviews are conducted and analyzed it must be clear how important the individual practices are for a DevOps implementation and whether they are a must- or a nice-to-have.

6.1.1 Survey Development

In order to ensure that all people being interviewed are qualified to remark on the defined process, criteria for suitable interview partners are defined. These criteria are defined as follows:

- Several years experience of working with DevOps and/or Scrum in software development.
  DevOps experience leads to qualified feedback about the importance and implementation of a practice.

Scrum experience leads to qualified feedback about the described integration of a DevOps practice in a Scrum process and the possible positive effects in doing so.

- Experts should (partially) work for different companies.
  In order to prevent feedback from being to one-sided, as they work for the same company and therefore have similar experiences, employers of experts should be taken into consideration.

- Experts should (partially) hold different positions within their respective teams.
  In order to get feedback from different points of view, experts positions should be taken into consideration.

Furthermore, in order to ensure a consistent approach during all the interviews, an interview guideline for semi-structured interviews is defined as follows:

1. The problem statement and the goal of the thesis are described shortly
2. It is described how the literature review was conducted and how the DevOps practices selected within this thesis were chosen
3. The expert is asked to
   - Rank the importance of each practice on a scale from 1 - 6 (1: unimportant; 6: very important)
   - Declare whether the practice is a must- or a nice-to-have for a successful DevOps implementation
   - Make remarks based on personal experience and expertise
4. The expert is asked about his/her experiences with DevOps and/or Scrum (duration, role, sector)
5. For each practice a definition within this thesis is given and it is described how the practice can be integrated into a common Scrum process (Chapter 5 serves as a guide for these descriptions)
6. The expert is asked whether, according to personal experience and expertise, any practices were overlooked

Interviews are held in German and the results are translated during evaluation. Furthermore, interview partners are anonymized and described by their role and experience.

Title	Role	Sector	Experience
Team lead 1	Team lead Product owner (company 1)	Finance/Banking	- Several years as a product owner and Scrum master in a Scrum team - Team lead in a DevOps team
Team lead 2	Team lead Product manager Developer (company 2)	Cloud computing	10 years of DevOps experience 1) Software developer 2) Team lead in a DevOps team 3) Organizational lead in a DevOps team 4) Responsible for product delivery and infrastructure monitoring
CEO	CEO (company 3)	Server operation	- 20+ years of DevOps experience Usage of DevOps practices before term DevOps was established - 10 years of Scrum experience
Developer 1	Developer Scrum master (company 4)	Finance/Banking	20 years of DevOps experience 1) Software and database developer 2) Scrum master in a DevOps team 3) Developer for infrastructure and operations
Developer 2	Developer Product owner (company 5)	Finance/Banking	- Several years as a developer and product owner in a Scrum team - 2 years as a DevOps developer
Developer 3	Developer Technical lead (company 5)	Finance/Banking	- 2 years as a DevOps developer - 1 year as technical lead in a DevOps team
IT-manager	IT-manager (company 6)	Finance/Banking	15+ years as IT-manager for several Scrum and DevOps teams
Agile coach	Agile coach Team lead Product owner Scrum master (company 6)	Finance/Banking	24 years in IT and 10 years of experience with Scrum and DevOps in different roles (Agile coach, project manager, team lead, product owner, Scrum master)

Table 6.1: Characterization of interview partners (experts)

6.1.2 Survey Execution and Evaluation

Firstly, using the previously defined criteria for suitable interview partners (see Chapter 6.1.1), a set of experts was chosen. Table 6.1 characterizes these selected experts.

Afterwards, the previously defined interview guideline (see Chapter 6.1.1) was used in order to conduct semi-structured interviews and validate the defined process. The

interviews are evaluated using summarizing qualitative content analysis according to Mayring [May15]. After transcribing the interviews the transcriptions are structured and the content is summarized and categorized. This is done using matrices, which display the experts categorizations for each practice, as well as quotations to adequately convey the experts opinions on the defined process. As interviews were held in German, the used quotations were translated into English.

Table 6.2 shows the importance of each DevOps practice according to each expert and Figure 6.1 shows the distribution of the rankings as a bar chart. As more than half of the practices are unanimously seen is quite important, having pretty high scores overall without any significant outliers, it can be seen that one of the goals of this study, namely finding the most frequently used DevOps practices, was achieved. Those practices are continuous integration, continuous delivery, continuous testing, collaboration and communication, continuous feedback, code versioning, infrastructure automation, continuous planning and quality assurance. However, there are two practices unanimously being seen as less important, namely test driven development and behavior driven development. This also relates to their occurrences during the systematic literature review performed in Chapter 4, as those practices were both only mentioned in approximately 10% of the selected papers, which was the cutoff for selecting a practice. For the remaining practices the experts were not in complete agreement. Continuous deployment, while being considered a 4 or 5 on the scale by most experts, has two outliers deeming the practice as not important at all. The same holds for continuous monitoring, which has a tendency of being seen as rather important by the experts, however, still has an outlier seeing this practice as unimportant. The opposite can be said for cloud computing, which tends to be seen as less important, but is still seen as very important by two experts. The most disputed practices are continuous experimentation and microservice architecture, which are both ranked over the entire spectrum, from being seen as very important to unimportant.

Table 6.3 shows whether each expert sees each DevOps practice as a must- or a nice-to-have for a DevOps implementation or whether the expert thinks the practice should be removed entirely and Figure 6.2 shows the distribution of this categorization as a bar chart. It can be seen that, despite there being tendencies, for many practices the experts are not in complete agreement. While the practices continuous integration, continuous delivery, code versioning and continuous planning are almost unanimously seen as must-haves and the practices continuous deployment, cloud computing, continuous experimentation, microservice architecture, test driven development and behavior driven development are almost unanimously seen as nice-to-haves for a successful DevOps implementation, the votes are split for the practices continuous testing, continuous monitoring, collaboration and communication, continuous feedback, infrastructure automation and quality assurance. Two practices stand out further as they are the only ones that each have one vote to be removed entirely. The first practice being behavior driven development, which is mainly classified as a nice-to-have for a successful DevOps implementation, but also has one vote for it to be removed entirely. The second practice being microservice architecture,

	Continuous Integration	Continuous Delivery	Continuous Deployment	Continuous Testing	Continuous Monitoring	Collaboration & Communication	Cloud Computing	Continuous Feedback	Code Versioning	Continuous Experimentation	Infrastructure Automation	Microservice Architecture	Continuous Planning	Quality Assurance	Test Driven Development	Behavior Driven Development
Team lead 1	6	6	4	6	3	6	4	4	6	1	4	5	6	6	4	3
Team lead 2	6	6	5	6	6	4	6	6	6	3	6	1	6	6	3	2
CEO	5	5	5	6	4	3	2	4	6	3	6	1	5	5	3	2
Developer 1	6	6	4	5	6	6	3	5	6	4	3	4	6	6	2	2
Developer 2	6	6	5	5	4	6	2	6	6	5	6	3	6	4	3	2
Developer 3	6	4	5	6	6	6	4	6	6	5	5	3	6	6	3	3
IT-manager	6	6	1	6	5	5	6	5	6	4	6	6	6	6	4	1
Agile coach	6	6	2	4	1	6	2	6	6	2	4	3	5	6	1	1
Average	5,9	5,6	3,9	5,5	4,4	5,3	3,6	5,3	6	3,4	5	3,3	5,8	5,6	2,9	2
Median	6	6	4,5	6	4,5	6	3,5	5,5	6	3,5	5,5	3	6	6	3	2

Table 6.2: Importance of practices for a DevOps implementation according to experts (1: unimportant; 6: very important)

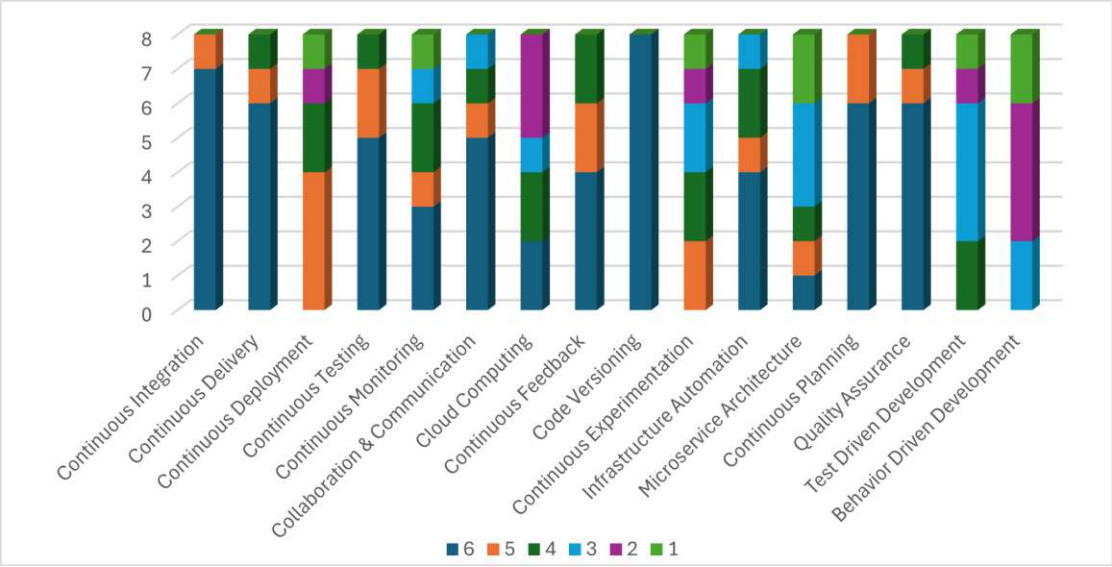

Figure 6.1: Distribution of importance of practices for a DevOps implementation according to experts (1: unimportant; 6: very important)

which is also mainly classified as a nice-to-have for a successful DevOps implementation, but also has one vote for it to be a must-have and one vote for it to be removed entirely at the same time, making it clear that there is no clear consensus about this practice among experts.

Overall, this categorization also correlates with the ranking on a scale from 1 to 6. Practices almost unanimously seen as must-haves are all ranked close to a 6 (very important) on average, while practices almost unanimously seen as nice-to-haves average below a 4, mainly around a 3 on the scale. The same holds for the practices with split votes. Despite experts not being in complete agreement on whether these practices are must- or nice-to-haves for a successful DevOps implementation, on average they all rank 5 or higher, making them very important, be they mandatory or not.

	Continuous Integration	Continuous Delivery	Continuous Deployment	Continuous Testing	Continuous Monitoring	Collaboration & Communication	Cloud Computing	Continuous Feedback	Code Versioning	Continuous Experimentation	Infrastructure Automation	Microservice Architecture	Continuous Planning	Quality Assurance	Test Driven Development	Behavior Driven Development
Team lead 1	M	M	N	N	N	M	N	M	M	N	N	N	M	N	N	N
Team lead 2	M	M	N	M	M	N	N	M	M	N	M	N	M	M	N	N
CEO	N	N	N	M	N	N	N	N	M	N	M	R	N	N	N	N
Developer 1	M	M	N	N	M	M	N	N	M	N	N	N	M	M	N	N
Developer 2	M	M	N	N	N	M	N	M	M	N	M	N	M	N	N	N
Developer 3	M	N	N	M	M	M	N	M	M	N	N	N	M	M	N	N
IT-manager	M	M	N	M	N	N	N	N	M	N	M	M	M	M	N	R
Agile coach	M	M	N	N	N	M	N	M	M	N	N	N	M	N	N	N
Must-have (%)	88	75	0	50	38	63	0	63	100	0	50	13	88	50	0	0

Table 6.3: Importance of practices for a DevOps implementation according to experts (M: must-have; N: nice-to-have; R: should be removed)

Table 6.4 shows the practices ranked according to their importance based on the expert feedback.

The following chapters show a detailed evaluation of the expert feedback for each DevOps practice. Furthermore, each chapter lists adaptions to the defined process resulting from the expert feedback.

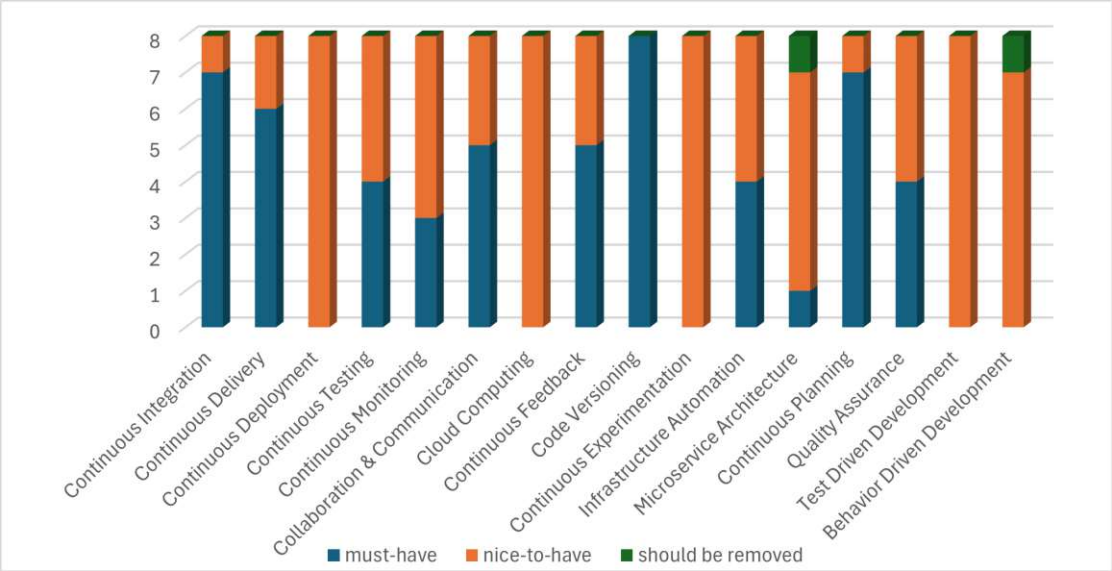

Figure 6.2: Distribution of importance of practices for a DevOps implementation according to experts (M: must-have; N: nice-to-have; R: should be removed)

Practice	**Importance (Average)**	**Importance (Median)**	**Must-have (%)**
Code Versioning	6	6	100
Continuous Integration	5,875	6	87,5
Continuous Planning	5,75	6	87,5
Continuous Delivery	5,625	6	75
Quality Assurance	5,625	6	50
Continuous Testing	5,5	6	50
Collaboration & Communication	5,25	6	62,5
Continuous Feedback	5,25	6	62,5
Infrastructure Automation	5	5,5	50
Continuous Monitoring	4,375	4,5	37,5
Continuous Deployment	3,875	4,5	0
Cloud Computing	3,625	3,5	0
Continuous Experimentation	3,375	3,5	0
Microservice Architecture	3,25	3	12,5
Test Driven Development	2,875	3	0
Behavior Driven Development	2	2	0

Table 6.4: DevOps practices ranked based on the expert feedback

6.1.2.1 Continuous Integration

There was a strong consensus about importance of continuous integration among the experts. For example, "Team lead 1" stated that without the possibility to add increments and perform automated tests, there is no way to set up an orderly DevOps stream, "Team lead 2" called continuous integration the basis for efficient work and "Developer 1" pointed out that the more often code is integrated into a common codebase, the lower the error rate. Regarding importance, "CEO" added that continuous integration is not immediately necessary when starting a new project, but the further along the project gets, the more important it becomes.

Practice Adaptations

As all experts agreed with the practice definition it will not be changed. However, the remark about its necessity at different stages of the project will be added.

6.1.2.2 Continuous Delivery

There was a strong consensus about importance of continuous delivery among the experts. Regarding its importance "Team lead 2" added that developers should not be given the power to veto a release, as this would not be scalable up to many developers and continuous delivery prevents this as the software is always in a release state. Furthermore, "CEO" pointed out that continuous delivery can only be done when having a continuous integration pipeline in place. Additionally, "Developer 3" stated that in practice, not every increment is also directly rolled out to an environment. Usually certain development packages are being built and delivered to a test environment on demand, with "CEO" adding, that this is usually done once per sprint.

Practice Adaptations

The remarks about the necessity of a continuous integration pipeline and the possibility to deliver deployment packages at defined points in time will be added to the practice description.

6.1.2.3 Continuous Deployment

Regarding continuous deployment the experts raised some concerns, with "Team lead 1" stating that experience shows that a fully automated deployment process without any human interaction is still not possible in many areas, however, it is still important to continue working towards this goal in order to reduce human errors and dependencies. Similarly, "Team lead 2", "Developer 2", "Developer 3" and "Agile coach" stated that always deploying directly to production only works in theory and is not scalable to big products. It might work in start-ups, however, the bigger the company and the more critical the product, the more requirements, such as company guidelines, certifications or regulatory requirements, exist, which prevent deployment without approval, but apart from that, everything that can be automated should also be automated in order to reduce manual tasks. "Developer 1" pointed out, the more environments a software needs to be

deployed to, the more important it is to automate these deployments, however, if there is only one environment, the benefit is relatively small.

Opposed to this, "CEO" stated that continuous deployment to production as soon as all previously defined acceptance criteria are met prevents stakeholders from blocking a release during sprint review. Furthermore, "Team lead 1" stated that using DevOps in combination with Scrum, the development of these automated deployment pipelines should be kept in a separate team, provisioning the service to the DevOps team, so that they can focus on the development of their continuous artifact.

Practice Adaptations

Due to the number of concerns that were raised about the usability of continuous deployment in practice, these concerns will be added to the practice description. However, the possibility of triggering a deployment to production every time an increment is integrated into a common code base and successfully evaluated, will still be kept as there might be use cases where this is usable. Also, the possibility of developing a base for automated deployment pipelines in a different team and providing it to several teams within the company will be listed.

6.1.2.4 Continuous Testing

There was a strong consensus about importance of continuous testing among the experts. For example, "Team lead 1" stated that test automation is absolutely necessary and safes a lot of time where it can be set up with reasonable overhead. "Agile coach" added that successful automated tests can be part of the DoD and significantly accelerate feedback loops when releasing code to a different stage. Furthermore, "Developer 1" pointed out the importance of performing automated tests as soon as possible at the integration stage, when code is added to a common codebase. Another advantage of continuous testing was added by "Developer 2", who stated that with increasing complexity of a product it gets harder to keep track of all changes. It is therefore important to build an automated test suite to prevent unwanted behavior and maintain quality. However, "CEO" also stated that it is not necessary to write an automated test for every small change as successful feature tests often also test smaller changes implicitly. "CEO" further added that continuous testing is not immediately necessary when setting up a new project using DevOps, especially when still working on a proof of concept.

However, the experts also raised some limitations, with "Team lead 1" stating that the time and cost required to build and maintain the tests rise with increasing complexity, which often makes it economically unfeasible. Furthermore, "CEO" stated that fully setting up continuous testing only works in combination with CI/CD.

Practice Adaptations

The acceleration of the feedback loop as well as the increased importance with increasing complexity of the product and the raised limitations of continuous testing will be added to the practice description.

6.1.2.5 Continuous Monitoring

Regarding continuous monitoring some additions were made by the experts. "Team lead 1" and "Developer 2" pointed out that the importance of continuous monitoring strongly depends on the importance of the application. While it might be very important for certain critical applications, this might not be the case every time. Furthermore, the experts "CEO", "Developer 1", "Developer 3" and "Agile coach" stated that continuous monitoring distinguishes itself from continuous testing by having a focus on the operations side of a business by monitoring servers, systems, workload, performances, memory etc. "IT-manager" also added that continuous monitoring is more important in a production than a test environment.

Regarding the practices integration with Scrum "Agile coach" had a different opinion and stated that, despite continuous monitoring and Scrum sharing the same principals, namely inspection and adaptation, continuous monitoring is not part of a Scrum process, but a separate process running independently, providing input to the Scrum team.

Practice Adaptations

Due to the number of experts pointing out a distinction from continuous testing by having a focus on the operations side of a business, this distinction will be added to the practice description. The same applies to its changing importance based on the environment and type of application.

6.1.2.6 Collaboration & Communication

There was a strong consensus about importance of collaboration and communication among the experts. For example "Team lead 1" stated that collaboration and communication are mission critical for any agile project, especially if the developed product is too big to be handled by a single team there are dependencies between teams which need to be tackled using transparency and communication. Similarly, "Developer 1" called communication with operations a must-have, however, pointed out that communication of every single team member outside the team should be kept to a minimum and it should be handled by the product owner.

Apart from its importance, "Developer 2" stated that collaboration and communication is often implemented wrong, as too many mandatory meetings are set up, adding that it works best when implementing it into the teams culture so that knowledge exchange takes place on demand. During the sprint daily buzzwords can be shared and follow-ups can be scheduled on demand.

Furthermore, "IT-manager" and "Agile coach" both stated that if DevOps is implemented according to the textbook, then there is no distinction between development and operations and an employee covers development as well as operational expertise, therefore making communication to an external operations team obsolete. However, "IT-manager", "Agile coach", "Developer 2" and "Developer 3" all pointed out that in practice operations is often still a separate entity, be it due to company guidelines, regulatory requirements

or because they provide services for several teams. If there is a separation of teams the experts had different opinions on how to handle that. On the one hand "IT-manager" and "Agile coach" were of the opinion that it is not necessary to permanently have an operations employee sit in on meetings of the DevOps team, as this communication can be set up on demand, while on the other hand "Developer 2" and "Developer 3" pointed out the benefits of having them take part in meetings like the sprint daily, which deepens collaboration.

Practice Adaptations

As there was no clear consensus among experts on how to integrate operations into the Scrum team, in case there needs to be a separate team, both listed option will be pointed out in the practice description. This is done as both options agree that communication and collaboration with the operational side of a business is important and the best solution might depend on the concrete situation and setting.

6.1.2.7 Cloud Computing

There was no clear consensus about the importance of cloud computing among the experts, but they tended to see the practice as less important. "Team lead 1" stated that on the one hand using cloud computing is a good idea if it reduces complexity and risk. It is a way to remove certain tasks and complexity away from the team, so that they can focus on product development. On the other hand it is a benefit knowing one's product front to back and being able to adjust operational requirements as needed.

Regarding setting it up, "Developer 2" was of the opinion that if cloud computing is used, then managing and providing the basic cloud infrastructure should be done in a separate team, as it can then be used by several other teams, with "Developer 1" and "Developer 3" adding that provisioning of resources should always be done within the DevOps team, as each dependency to the outside leads to delays and negatively effects agility. Opposed to this, "Team lead 2" pointed out that giving developers an infrastructure without the possibility to change anything and experiment, might prevent them from finding better and maybe cheaper solutions.

Several experts, namely "Developer 1", "Developer 2" and "IT-manager" pointed out that this definition misses the distinction between private and public cloud, with "IT-manager" and "CEO" both preferring private cloud implementations and "CEO" going as far as trying to avoid public cloud computing where possible.

Practice Adaptations

The experts opinions on how to split responsibilities of setting up a cloud infrastructure and provisioning resources will be used to refine the description on how to integrate this practice with Scrum. As the majority of experts agreed with setting it up in a different team, this option is chosen for this process. Furthermore, the practice definition will be extended by different deployment models for cloud computing to add the distinction of

private and public cloud, stating that both possibilities exist. This extension will also be taken from the NIST definition of cloud computing.

6.1.2.8 Continuous Feedback

All experts agreed with the importance of continuous feedback. "Team lead 1" stated that feedback is always needed in order to continuously improve or keep providing a continuously high quality product. Therefore, continuous feedback loops are needed, be it human or systemic feedback. Similarly, "Developer 2" stated that without continuous feedback there is a bigger risk that developments miss the point and do not meet expectations. Furthermore, "Developer 3" pointed out that continuous feedback, provided by sprint daily and sprint retrospective, are absolutely necessary, however, additional usage of CI/CD-technologies to trigger a feedback process are optional and less important.

Practice Adaptations

As all experts agreed with this practice description no changes will be made.

6.1.2.9 Code Versioning

While all experts categorized this practice as very important and a must-have for a successful DevOps implementation, the experts "Team lead 2", "Developer 2", "Developer 3" and "IT-manager" stated that this practice could be removed from this list, as nowadays it is a standard for software engineering and its usage is therefore implied. Similarly, "Team lead 1" sees code versioning as part of a continuous delivery process.

Opposed to that, "Agile coach" stated that despite code versioning being an industry standard, nothing should be taken for granted and the practice should therefore still be listed.

"Developer 1" made a distinction between code versioning in general software engineering projects, where code versioning is used as an industry standard, and code versioning while working with DevOps, as DevOps adds a focus to versioning of metadata.

Practice Adaptations

As all experts agreed with the importance of code versioning and there is no harm in listing it as a separate practice, it will not be removed. However, the distinction made by "Developer 1" regarding versioning of metadata while working with DevOps will be added to the practice description.

6.1.2.10 Continuous Experimentation

The importance of continuous experimentation was one of the most disputed points during the interviews. The experts "Team lead 1", "Team lead 2", "CEO", "Developer 3" and "IT-manager" all agreed that it is important that employees can be creative and introduce their own ideas, however, these innovation need to work towards a goal, be prioritized and time-boxed. "Team lead 1" further added that doing this, the freedom

to perform continuous experimentation can easily be given using Scrum, but giving this freedom constantly takes a huge financial investment and takes away from the actual product being developed. "Developer 3" brought up the point of encouraging every employee to create new backlog items about ideas to further investigate. Those items should then be treated like every other story, being evaluated and prioritized. Similarly, "Agile coach" stated that experimentation needs to be time-boxed, however, proposed designated sprints solely working on experimentation. "Agile coach" also stated that this practice must not be limited to developers, but rather be open to all employees.

Besides potentially finding better solutions to existing problems or coming up with new ideas helping the company, "Developer 2" pointed out that giving employees the freedom to try out new ideas also provides motivation.

However, there were also limitations being named. "Agile coach" stated that the benefit of continuous experimentation strongly depends on the type of organization and its products and "Team lead 2" added that the bigger the organization and the product, the more these innovations may collide and lead to problems.

Practice Adaptations

As there was such a strong consensus about the importance of continuous experimentation being time-boxed, this adaptation will be added to the practice description. Furthermore, the listed limitations will also be added.

6.1.2.11 Infrastructure Automation

There were different views regarding infrastructure automation among the experts. While "Team lead 1" and "Developer 1" saw infrastructure automation as part of cloud computing and recommended merging the two practices, the other experts saw infrastructure automation and cloud computing as different practices overlapping with each other. Furthermore, "Team lead 2" agreed with all the listed goals and benefits of infrastructure automation, however, listed IaC as only one possibility of implementing it and stated that its implementation needs to be handled with care, as experience shows that IaC can make a system sluggish and can have negative effects on other implementations like feedback loops.

"CEO" went as far as to say that infrastructure automation is the only mandatory practice for a successful DevOps implementation. Furthermore, "CEO" named dockerization of applications, so that developers can simply use these containers to test and run their software, as an important part of infrastructure automation.

Regarding its implementation, "Developer 2" and "Agile coach" were of the opinion that it would be a good idea to split responsibilities. Basis tools for infrastructure automation should be provided by an external team, but the DevOps team should still be able to make changes independently.

Practice Adaptations

As most experts saw infrastructure automation as an independent practice it will not be merged with cloud computing. Furthermore, its definition will be adjusted, listing IaC as one, but not the only, possibility of setting up infrastructure automation. IaC is therefore a subset of infrastructure automation.

6.1.2.12 Microservice Architecture

Microservice architecture was the most controversial practice among experts. While "IT-manager" stated that strategically a microservice architecture should be enforced as it enables agility, "Team lead 2", "Developer 2" and "Agile coach" only stated that depending on the product a microservice architecture might be useful, but not a necessity for a successful DevOps implementation. Opposed to that, "CEO" called microservice architecture a philosophy, which should be seen independently from DevOps, as DevOps can work with monolithic systems as well as with microservice architectures and it should therefore be removed as a DevOps practice.

Furthermore, "Developer 1" and "IT-manager" pointed out that the more granular a service is set up, the higher the resulting overhead of managing it. It is important to find the right balance in order to prevent drawbacks like performance loss.

Practice Adaptations

As the experts views regarding microservice architecture were so different, yet, the general practice description was not disputed and the majority of experts found microservice architecture to be a DevOps practice, the practice description will not be changed.

6.1.2.13 Continuous Planning

There was a strong consensus about importance of continuous planning among the experts, with almost all experts seeing it as an absolute necessity for a successful DevOps implementation. Some experts made some additions to the practice description with regards to DevOps. "Team lead 2" pointed out that when working with DevOps, the ongoing workflow is much more depend on external factors. Due to the implemented automations, more frequent releases and a faster feedback loop, the ongoing workflow is interrupted far more often by external factors, which need to be addressed immediately. Therefore, a bigger buffer for last-minute adjustments needs to be calculated when planning a sprint or a release. Furthermore, "Developer 1" pointed out that when working with agile software development and therefore more short-term planning and potential last-minute adjustments, it is important not to forget about mid- and long-term plans. Similarly, "Developer 2" and "Developer 3" stated that when working with related products and interfaces provided by other teams, more long-term planning is required.

Practice Adaptations

The additions made by the experts regarding bigger buffers for last-minute adjustments and the importance of keeping an eye on mid- and long-term planning will be added to the practice description.

6.1.2.14 Quality Assurance

There was a strong consensus about the importance of quality assurance among the experts, with "Agile coach" pointing out that quality assurance corresponds with the Scrum principle of technical excellence, which is an example of why Scrum and DevOps complement each other so well - Scrum says what to do and DevOps says how to do it. "Agile coach" also added that fully implementing quality assurance yields huge benefits, however, the implementation is very time and cost intensive.

However, "Team lead 1" and "Developer 3" stated that the importance of quality assurance strongly depends on the developed product - while for some products it might be very important, it might not be for others.

Furthermore, "CEO" pointed out a strong correlation between quality assurance and CI/CD and "Developer 1", "Developer 3" and "IT-manager" went as far as calling quality assurance an umbrella term for continuous testing, continuous feedback and continuous monitoring. Additionally, all experts agreed that it is not necessary to list static code analysis as an individual practice.

Practice Adaptations

Despite quality assurance being a very broad term and several experts even calling it an umbrella term for other practices that were already discussed, the practice will still be kept and listed separately, due to the assigned importance. Even if other practices associated to quality assurance are not being fully implemented, singular quality assurance practices can still yield a benefit. As all experts agreed that it is not necessary to list static code analysis as an individual practice, the corresponding sub-chapter will be removed. Apart from that the practice description will not be changed.

6.1.2.15 Test Driven Development

Several experts raised doubt about this practice. "Team lead 2", "Developer 1", "Developer 3" and "Agile coach" all stated that TDD is theoretically a good practice, however, it does not assert itself in practice, with "Developer 3" adding that it is more important to write automated tests at all than to necessarily write them before development. Furthermore, "Developer 1" stated that experience shows, that before development it is often not known what is actually wanted, so the previously written tests might be correct, but the implementation still needs to be changed afterwards.

Practice Adaptations

As the expert feedback is reflected by the experts assigned importance, the practice is kept and no changes are made. Despite, the experts stating that the practice has difficulties asserting itself in practice, they also agree that the practice works well in theory, so there might be DevOps implementations benefiting from implementing it.

6.1.2.16 Behavior Driven Development

There was general skepticism about the usability of BDD in practice. "Team lead 2" believes that BDD would require a functional specification document, which fully depends on external stakeholders and fully defining this document in sufficient quality to use BDD would take up too much time and be unfeasible in practice. Similarly, "Developer 1" and "IT-manager" stated that BDD sounds nice in theory, but is not feasible in practice, as specifications are often incomplete or faulty, assumptions about behavior are made and stakeholders change requirements over time. However, "Agile coach" pointed out future potential of this practice connected to advancements in artificial intelligence and artificially generated tests.

Practice Adaptations

Due to the general skepticism about this practice and its usability in practice it will be removed from this list. However, it will be added to the section about possible future work, so that it might be further investigated in conjunction with support provided by artificial intelligence.

CHAPTER 7

Results

This chapter recaps the research questions and outlines the final answer for each of them. Afterwards, the found results are put into context of related work and existing state-of-the-art research.

7.1 Most frequently used DevOps Practices

In order to help companies to adapt their working process towards enabling agile DevOps, by extending a common Scrum process using DevOps practices, firstly the most frequently used DevOps practices in state-of-the-art research needed to be identified. To this end the following research question was deducted:

RQ1: Which DevOps practices are most frequently used in state-of-the-art research in the context of agile software development?

Using a systematic literature review, accurately described in Chapter 4, state-of-the-art research was analyzed in order to identify the most frequently used DevOps practices in the context of agile software development.

The SLR showed that during the years after the introduction of DevOps in 2008 publications were scarce, followed by a continuous increase of publications over the last couple of years. Up until 2013 there were hardly any publications addressing DevOps practices in agile software development and beginning in 2016 empirical papers conducting validation or evaluation research were picking up. Simultaneously, there was a steady flow of new philosophical papers, trying to structure the research field, proving the point that the scientific community still sees the need to further classify DevOps.

While nearly all papers mention the umbrella term agile (software) development, nearly half the papers do so without mentioning any concrete agile methodology. The agile methodology mentioned namely most frequently is Scrum, which underpins this studies

selection of a Scrum process as a suitable representative of agile methodologies to extend with DevOps practices. However, despite nearly half of the selected studies mentioning Scrum, hardly any of them go into it in further detail, proving the point that there is still great potential for further research regarding the interplay between the two methodologies DevOps and Scrum.

Table 7.1 shows all DevOps practices meeting the defined inclusion and exclusion criteria, which were found in at least 5 different studies and therefore selected as most frequently used in state-of-the-art research in the context of agile software development, answering RQ1. Practice descriptions, as found during full-text analysis of the state-of-the-art research, can also be found in Chapter 4.

DevOps Practice	**Reference**	**Count**
Continuous Integration	[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [29], [31], [32], [33], [34], [35], [36], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [52], [53]	49
Continuous Delivery	[1], [2], [3], [4], [5], [6], [8], [9], [10], [12], [13], [14], [15], [16], [17], [18], [19], [21], [22], [23], [24], [26], [27], [28], [29], [31], [32], [33], [34], [36], [38], [39], [40], [41], [42], [43], [44], [46], [47], [48], [49], [50], [51], [52], [53]	45
Continuous Deployment	[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [15], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [31], [32], [33], [34], [35], [36], [38], [39], [40], [41], [42], [44], [45], [47], [49], [51], [52], [53]	45
Continuous Testing	[1], [2], [3], [4], [5], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [23], [24], [25], [26], [27], [29], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [42], [43], [44], [45], [46], [47], [48], [49], [52], [53]	45
Continuous Monitoring	[1], [2], [6], [8], [9], [11], [12], [13], [16], [17], [18], [19], [20], [21], [23], [24], [25], [26], [27], [28], [30], [31], [32], [33], [34], [35], [36], [37], [40], [42], [43], [45], [46], [47], [48], [49], [50], [53]	38
Collaboration and Communication	[3], [6], [7], [9], [10], [11], [14], [16], [17], [18], [19], [20], [21], [22], [23], [25], [26], [27], [29], [31], [32], [34], [35], [36], [37], [38], [40], [41], [42], [44], [45], [46], [47], [50], [51], [52]	36

Cloud Computing	[1], [2], [3], [5], [7], [10], [11], [12], [15], [16], [18], [19], [20], [23], [24], [25], [26], [27], [29], [32], [36], [39], [40], [42], [44], [45], [46], [47], [49], [50], [51], [52], [53]	33
Continuous Feedback	[1], [2], [6], [8], [11], [14], [16], [19], [20], [24], [26], [28], [30], [32], [33], [34], [35], [37], [40], [44], [46], [47], [49]	23
Code Versioning	[1], [2], [4], [7], [8], [9], [10], [14], [17], [19], [20], [25], [29], [32], [33], [35], [38], [39], [40], [41], [42], [45], [47]	23
Continuous Experimentation	[2], [6], [9], [10], [12], [13], [16], [18], [20], [21], [22], [23], [26], [30], [32], [33], [34], [35], [37], [44], [45]	21
Configuration Management Automation	[1], [2], [8], [16], [18], [19], [22], [23], [25], [27], [29], [33], [34], [36], [39], [40], [42], [45], [46], [52]	20
Infrastructure Automation	[2], [7], [8], [10], [11], [15], [16], [18], [21], [22], [25], [26], [27], [29], [32], [34], [43], [45], [52]	19
Sandboxing	[2], [5], [7], [9], [10], [11], [14], [17], [20], [25], [32], [33], [38], [44], [46], [47], [48]	17
Microservice Architecture	[2], [3], [5], [10], [14], [18], [23], [25], [26], [28], [29], [39], [43], [47]	14
Continuous Release	[9], [18], [20], [27], [31], [32], [33], [35], [44], [46]	10
Continuous Planning	[1], [11], [12], [13], [17], [20], [23], [44], [49]	9
Quality Assurance	[3], [5], [10], [16], [17], [18], [19], [40]	8
Behavior Driven Development (BDD)	[10], [15], [19], [23], [37], [40], [48]	7
Process Standardization	[11], [12], [19], [26], [27], [36]	6
Static Code Analysis	[2], [9], [35], [37], [48], [53]	6
Test Driven Development (TDD)	[14], [15], [19], [37], [48]	5

Table 7.1: Most frequently used DevOps practices in state-of-the-art research and their references (Appendix A)

7.2 DevOps Practices to extend Scrum

After identifying the most frequently used DevOps practices in state-of-the-art research, they were analyzed regarding there usability to extend a common Scrum process. To this end the following research question was deducted:

RQ2: Which DevOps practices can be used to extend agile software development, by example of Scrum, in order to reduce the risk of development and operations working towards unaligned goals?

Using the identified DevOps practices from the systematic literature review in Chapter 4, Chapter 5 examined them towards their suitability to be used in connection with Scrum. Therefore, valid practice definitions in the context of this thesis were chosen and it was described if and how these DevOps practices can extend Scrum in order to reduce the risk of a gap between development and operations within the software engineering process. Following this definition, the process was validated using expert interviews as outlined in Chapter 6 and the process was adapted according to their feedback. The following sections outline the finalized process, including practice definitions and a description on how these DevOps practices can extend Scrum in order to reduce the risk of a gap between development and operations within the software engineering process, answering RQ2.

7.2.1 Continuous Integration

Practice Definition

Within this thesis continuous integration is defined as a developer practice where developers need to integrate their work into a shared repository frequently. Usually each person integrates at least daily, leading to multiple integrations per day [DSM+22] [FS17] [FS14] [GBM19] [MMA+20] [Son15] [SMB17] [VZR+17] [Vir15]. When code is added to the code base, interconnected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking compliance with coding standards, and building deployment packages are automatically triggered, so teams can quickly collaborate on a single project [DSM+22] [FS17] [FS14] [GS21] [GGL21] [GH23] [MMA+20] [PEL21] [SDW18] [Son15] [VZR+17] [Vir15]. Furthermore, the integration process of developer's code needs to detect errors quickly and avoid failure of builds. Any continuous integration failure is also an important event which may have a number of ceremonies and highly visible artifacts to help ensure that problems leading to integration failures are solved as quickly as possible by those responsible [AR19] [FS17] [FS14] [GGL21] [VZR+17].

Integration with Scrum

Continuous integration can be integrated in a Scrum process very well. Using Scrum the project is already broken down into small and manageable tasks which are worked on by the developers producing thoroughly verified increments. By integrating these increments more frequently into a common code base and adding the previously described automations, continuous integration can be achieved. However, it needs to be stated that continuous integration is not immediately necessary when starting a new project, but the further along the project gets, the more important it becomes ("CEO", personal communication, Jan. 21, 2025).

7.2.2 Continuous Delivery

Practice Definition

Within this thesis continuous delivery is defined as a development practice where every change is treated as a potential release candidate to be frequently and rapidly evaluated through one's continuous delivery pipeline. This typically involves activities such as code analysis, documentation generation, acceptance testing, regulatory compliance assessments, license scanning and requirement verification [DSM+22] [GS21] [GBM19] [GH23] [SMB17] [TSM19b] [TSB21] [VZR+17]. Using continuous delivery, software always needs to be in a release state, so that it is always possible to deploy and/or release the latest working version [AR19] [Son15] [SMB17] [TSM19b] [TSB21]. Furthermore, software is automatically deployed and delivered to a production-like environment [DSM+22] [FS17] [SDW18] [TSM19b] [TSB21]. In order for continuous delivery to work a continuous integration pipeline must be in place ("CEO", personal communication, Jan. 21, 2025). As this production-like environment matches a definition for a sandbox [Tec24] [Ven05] and no specific definition for sandboxes in the context of DevOps was found during the systematic literature review, the two practices are merged within this thesis.

Integration with Scrum

As Scrum increments are completed developments additive to all prior increments they can always be deployed by themselves, thereby being ideal to work in conjunction with continuous delivery. By implementing a continuous delivery pipeline and evaluating all finished increments through it, each increment is treated as a potential release candidate and it is thereby always possible to deploy and/or release the latest working version. Furthermore, using continuous delivery each evaluated increment can automatically be deployed to a working test environment, allowing stakeholders to test the increment as soon as possible and give valuable feedback without needing to wait for the end of the sprint. However, usage of continuous delivery in practice shows, that it is not always necessary or possible to automatically deploy each increment to a working test environment. Often certain development packages are being built and delivered to a test environment on demand ("Developer 3", personal communication, Feb. 10, 2025)("CEO", personal communication, Jan. 21, 2025).

7.2.3 Continuous Deployment

Practice Definition

Within this thesis continuous deployment is used synonymously to continuous release and is defined as the process of deploying to production as soon as qualified changes in software code or architecture are ready, and without human intervention. Continuous deployment implies continuous delivery, therefore, release candidates evaluated and approved in continuous delivery are automatically placed in a production environment [AR19] [DSM+22] [FS17] [GGL21] [GBM19] [GH23] [MMA+20] [SDW18] [SMB17] [Vir15].

Integration with Scrum

There are several possibilities on how to use continuous deployment in connection with Scrum, depending on the use case of the delivered product.

As Scrum usually deploys code at the end of a sprint, the continuous deployment process can be triggered once the sprint review is done and the outcome of the sprint is inspected, thereby automatically deploying code to production once per sprint cycle. This might be a scaled-down approach to continuous deployment, as not every increment is automatically and instantly deployed to production, however, the manual overhead of deploying to production is still massively reduced and as continuous deployment implies continuous delivery, each evaluated increment can still automatically be deployed to a working test environment, allowing stakeholders to test the increment as soon as possible and give valuable feedback without needing to wait for the end of the sprint.

Alternatively, if the use case of the delivered product allows it, the continuous deployment process can also be triggered every time an increment is integrated into a common code base, directly pushing the code to production as soon as it is evaluated. The total progress of the sprint can still be discussed during the sprint review, despite the code already being deployed to production. In practice this approach might work in start-ups, however, the bigger the company and the more critical the product, the more requirements, such as company guidelines, certifications or regulatory requirements, exist, which prevent deployment without approval, but apart from that, everything that can be automated should also be automated in order to reduce manual tasks ("Team lead 2", personal communication, Jan. 27, 2025)("Developer 2", personal communication, Jan. 23, 2025)("Developer 3", personal communication, Feb. 10, 2025)("Agile coach", personal communication, Jan. 30, 2025).

Furthermore, it needs to be added that the benefit of continuous deployment strongly depends on the number of environments a software needs to be deployed to ("Developer 1", personal communication, Jan. 23, 2025).

7.2.4 Continuous Testing

Practice Definition

Within this thesis continuous testing is defined as a process of testing early, testing often, testing everywhere, and automating [AR19]. This is done by building a test suite which can automatically be executed on every software build generated without any user intervention [Vir15] in order to verify the correctness, quality and performance of developed software [AMB+16] and promptly assess the business risks linked to a software release candidate [GHM18]. This test suite can then be integrated into the software delivery pipeline [Vir15]. All this reduces manual testing, accelerates the testing process significantly [H+16], reduce the time between the introduction of errors and their detection, with the aim of eliminating root causes more effectively [BSMG12] and strongly affects deployment speed [SBO18b].

Integration with Scrum

Continuous testing and Scrum can work together well. One of the main duties of developers working in a Scrum team is to instill quality by adhering to a Definition

of Done (DoD). Continuous testing can help achieve this task. By writing a test suit covering the DoD and continuously testing the produced code against it, a better test coverage can be achieved and the overall stability of the product is improved. Similarly, continuous testing can be part of the DoD and significantly accelerate feedback loops when releasing code to a different stage ("Agile coach", personal communication, Jan. 30, 2025). Furthermore, as tests of previous increments are continuously run on the current code base, unwanted ramifications of new developments on preexisting functions can be detected and thereby avoided early. This gets more important as complexity of a product increases and it therefore gets harder to keep track of all changes ("Developer 2", personal communication, Jan. 23, 2025). Finally, as manual testing is reduced and the testing process is significantly accelerated, more time can be spent on other tasks.

However, it needs to be stated that the time and cost required to build and maintain the tests rise with increasing complexity, which often makes it economically unfeasible ("Team lead 1", personal communication, Jan. 20, 2025). Furthermore, fully setting up continuous testing only works in combination with CI/CD ("CEO", personal communication, Jan. 21, 2025).

7.2.5 Continuous Monitoring

Practice Definition

Within this thesis continuous monitoring is defined as the instrumentation of an application and the aggregation of monitored data into insights. This involves having a continuous feedback loop that runs from the production environment to the start of the development cycle, including a complete timeline of development and operations events. It involves proactive detection and awareness of events in critical environments, such as test and production, in order to expose (know the state of) issues before they cause failures [LKO16], enable early detection of quality-of-service problems, such as performance degradation, and also the fulfillment of service level agreements [vHHW+09]. Continuous monitoring distinguishes itself from continuous testing by having a focus on the operations side of a business by monitoring servers, systems, workload, performances, memory etc. ("CEO", personal communication, Jan. 21, 2025)("Developer 1", personal communication, Jan. 23, 2025)("Developer 3", personal communication, Feb. 10, 2025)("Agile coach", personal communication, Jan. 30, 2025). Furthermore, continuous improvement in DevOps is enabled by constantly monitoring processes and measuring them against metrics [dF17] [GH23].

Determining and implementing monitoring metrics of a system is done by developers and operations alike and can include activities like [LKO16]:

- Using a small subset of high-priority test cases executed in production to actively monitor the environment
- Effective instrumentation of software by development in collaboration with operations to give information about its health and performance

- The use of aids, such as feature flags, to quickly recover code failures in production

Integration with Scrum

Two of the three empirical Scrum pillars are inspection and adaptation. Where inspection means to frequently and diligently inspect all Scrum artifacts to detect potentially undesirable variances or problems and adaption means adjusting the process, if any aspects of it deviate outside acceptable limits or if the resulting product is unacceptable [SS20]. These Scrum pillars are congruent with the goals of continuous monitoring, as continuous monitoring implements monitoring metrics in order to control the system and detect deviations or abnormalities early (inspection) and uses these measurements in order to promote continuous improvement (adaptation). Continuous monitoring, and subsequently continuous improvement, are therefore a concrete practice to implement these vague ideas.

It needs to be added that the importance of continuous monitoring strongly depends on the importance of the application ("Team lead 1", personal communication, Jan. 20, 2025)("Developer 1", personal communication, Jan. 23, 2025). It is more important in a production than a test environment ("IT-manager", personal communication, Jan. 27, 2025).

7.2.6 Collaboration and Communication

As only one definition for collaboration and communication was found during the systematic literature review, this definition is also adopted by this thesis.

7.2.6.1 Collaboration

Practice Definition

Within this thesis collaboration is defined as a cultural success factor of DevOps which focuses on working together and the exchange of knowledge and experience in and between teams [GH23]. DevOps is all about connecting development and operations, which entails collaboration [SG18]. Advantages of a strong culture of collaboration include enhanced development and operations processes, as well as improved quality of software services and products [dFJT16]. Cross-functional collaboration can result in competitive advantages [Wie18]. Furthermore, collaboration leads to significant business improvements, enhanced operation software quality, and better-quality IT services [vBdKT19].

Integration with Scrum

As collaboration is a cultural success factor it does not change the flow of the Scrum process, but incorporating it still has an impact by extending boundaries of the team and the circle of involved stakeholders. As collaboration focuses on working together and the exchange of knowledge and experience in and between teams, it is already partially congruent with Scrum, as Scrum places a strong focus on transparency and

sharing information and knowledge within the team. Using Scrum, teams are already self-managing, self-organizing and cross-functional in order to maximize team performance, meaning the Scrum team is already a self-contained group of people with different functional expertise working toward a common goal [Sch04] [SS20], therefore enforcing collaboration within the team and reducing external dependencies. As DevOps is all about connecting development and operations, these two point can tie together by adding an employee with operational expertise to this cross-functional team, thereby adding an operations point of view to the ongoing software development process. This employee does not necessarily need to be a developer, as providing a basic infrastructure might still be done in a separate team, even when working with DevOps, but using this infrastructure can move inside the Scrum team by employing the right personnel. In case of a separate team being necessary due to company guidelines or regulatory requirements, further collaboration can be achieved in one of two ways. Either necessary meetings between the teams are set up on demand ("IT-manager", personal communication, Jan. 27, 2025)("Agile coach", personal communication, Jan. 30, 2025)or a member of the separate team sits in on all meetings of the DevOps team ("Developer 2", personal communication, Jan. 23, 2025)("Developer 3", personal communication, Feb. 10, 2025). The best solution might depend on the concrete situation and setting.

In order to fully implement a culture of collaboration, lessons learned within the Scrum team also need to be communicated to the rest of the organization, so that other teams can learn from the Scrum team and adapt their process improvements. Furthermore, by implementing a strong culture of collaboration the Scrum team itself can profit from lessons learned by other teams and can take feedback from other departments into account, which are directly affected by the teams output.

7.2.6.2 Communication

Practice Definition

Within this thesis communication is defined as a cultural success factor of DevOps which focuses on the communication between development and operations, which is essential for DevOps to succeed [AMS+23]. For the teams to work together effectively, information within and between the teams needs to be shared effectively and frequently. The presence of quality communication helps establish a collaborative culture which contributes to a successful DevOps environment [GH23]. Communication is crucial for DevOps to succeed because not sharing important information may have a significant negative impact, therefore, managers must impose regular communication [AMS+23].

Integration with Scrum

As communication is a cultural success factor it does not change the flow of the Scrum process, but incorporating it still has an impact by extending boundaries of the team and the circle of involved stakeholders. As communication focuses on effectively and frequently sharing information between development and operations, operations should be seen as a crucial stakeholder of a successful software project. Therefore, an employee

with operational expertise should be added to the existing cross-functional Scrum team, thereby adding an operations point of view to the ongoing software development process, giving valuable feedback during planning and development and thereby eliminating misalignment between development and operations before it occurs. This employee does not necessarily need to be a developer, as providing a basic infrastructure might still be done in a separate team, even when working with DevOps, but using this infrastructure can move inside the Scrum team by employing the right personnel. In case of a separate team being necessary due to company guidelines or regulatory requirements, further communication can be achieved in one of two ways. Either necessary meetings between the teams are set up on demand ("IT-manager", personal communication, Jan. 27, 2025)("Agile coach", personal communication, Jan. 30, 2025)or a member of the separate team sits in on all meetings of the DevOps team ("Developer 2", personal communication, Jan. 23, 2025)("Developer 3", personal communication, Feb. 10, 2025). The best solution might depend on the concrete situation and setting.

7.2.7 Cloud Computing

Practice Definition

Within this thesis the cloud computing definition provided by the NIST is used. It describes cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [Clo11]. Hence, although cloud computing is sometimes regarded as a type of outsourcing or contrasted with on-premises infrastructure, it is a model for organizing shared computing resources such that automatic, on-demand access to these resources is possible, irrespective of who owns them and whether they are located “on or off premises” [KLJ18].

The essential characteristics of cloud computing according to the NIST definition are [Clo11]:

- On-demand self-service
  A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- Broad network access
  Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- Resource pooling
  The provider’s computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

- Rapid elasticity
  Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
- Measured service
  Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource use can be monitored, controlled and reported.

Furthermore, cloud computing comprises three service models [Clo11] [KLJ18]:

- Software-as-a-Service (SaaS): cloud-based applications
- Platform-as-a-Service (PaaS): cloud-based application environments
- Infrastructure-as-a-Service (IaaS): cloud-based hardware

Finally, cloud computing comprises four deployment models [Clo11]:

- Private cloud
  The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
- Community cloud
  The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.
- Public cloud
  The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
- Hybrid cloud
  The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Integration with Scrum

Cloud computing is an operations practice. Using cloud computing in order to manage an application can be seen as a developers task in the context of Scrum and it can therefore work in conjunction with Scrum. As cloud computing still needs to be set up, managed and the computing resources need to be configured, its usage does not make operations obsolete, but rather changes the resources they are working with. Depending on the size of the company these operations tasks might either be done by an employee with operational expertise who is part of the cross-functional Scrum team, or by a separate team, tasked with providing a basic infrastructure. Managing and providing the basic cloud infrastructure by a separate team brings the benefit of several other teams being able to use it ("Developer 2", personal communication, Jan. 23, 2025), however, provisioning of resources should always be done within the DevOps team, as each dependency to the outside leads to delays and negatively effects agility ("Developer 1", personal communication, Jan. 23, 2025)("Developer 3", personal communication, Feb. 10, 2025).

7.2.8 Continuous Feedback

Practice Definition

As only one definition for continuous feedback was found during the systematic literature review, this definition is also adopted by this thesis. Therefore, within this thesis feedback denotes “information about actions returned to the source of the actions” [KLJ18] and includes internal feedback (i.e. feedback that actors directly perceive from monitoring the outcomes of their actions) and external feedback (i.e. feedback provided by another person) [BW95]. Feedback processes operate during many software development activities, such as compiling, deploying, testing, experimenting, and reviewing [KLJ18].

Continuous feedback tries to reduce the waiting time for feedback through time-boxed iterations [Vid09], which are seen as learning cycles or feedback loops [Sch04]. The shorter these cycles are, the more can be learned [PP04]. Using CI/CD technology allows small code changes to trigger the feedback process [Hum10]. Such technology automatically integrates changes into the shared code base and automatically tests and deploys them [KLJ18].

Integration with Scrum

A Scrum process by definition already uses continuous feedback. A key pillar of Scrum is transparency, where the emergent process and work must be visible to those performing the work as well as those receiving the work. Consequently Scrum employs several events promoting continuous feedback. The daily Scrum is about updating the team daily on what has been accomplished, what needs to be done next and what obstacles developers are currently presented with. The sprint review is about presenting the results of the teams work to key stakeholders, discussing progress toward the product goal and consequently collaborating on what to do next. Finally, the sprint retrospective is about

planning ways to increase quality and effectiveness. Therefore, the daily Scrum and the sprint retrospective present internal feedback, while the sprint review presents external feedback [SS20].

Continuous feedback is all about shortening learning cycles or feedback loops. Within a Scrum process continuous feedback can be further improved by using CI/CD technology, which allows small code changes to trigger the feedback process. Thereby, feedback on the current work can be received and incorporated much sooner.

7.2.9 Code Versioning

Practice Definition

Within this thesis version control, also known as source control, is the management of changes to any kind of document, but it is usually used to control and manage software source code or scripts [Atl24].

There are two types of version control systems: Centralized VCS and distributed VCS. A centralized VCS is a client-server approach. The client checks out a copy from the centralized repository. All these data and history is stored and retrieved from the central repository. On the other hand, we have the distributed VCS which is a peer to peer approach. Over copy is a repository and contains full project history [Git14]. Network connection is needed when it is to be shared.

Version controlling code provides several benefits:

1. The ability to work on the same project by more than one person at the same time. Each person will have a copy of the source code to do whatever changes required, then changes are merged with the main code, after the necessary approvals [SDW18].
2. The history stored by a version control system would allow a developer to examine changes to faulty code [SDW18].
3. The code can be branched. Developers can work on separate branches of the source code to create and test new features. In the end, a branch can be merged with the main branch (called master) [SDW18].
4. The number of code submissions of the entire team and each developer can be viewed on a periodic basis, thereby making the contribution data of team members clear at a glance, which facilitates convenient and intuitive team work assessment [GGL21]

The most used and popular version controlling system with CI/CD approaches is Git [ZND18].

Integration with Scrum

Code versioning is a practice enabling development teams to manage changes made to the source code or scripts, enabling more than one person to work on a project at the same time and tracking the codes history. Using code versioning does not impact the overall workflow of a Scrum team and can therefore easily be used as a development practice by developers of a Scrum team. However, there is one distinction of using code versioning in connection with DevOps than in general software engineering projects, as it adds a focus to versioning of metadata ("Developer 1", personal communication, Jan. 23, 2025).

7.2.10 Continuous Experimentation

Practice Definition

Within this thesis continuous experimentation is used synonymously to continuous innovation and is defined as the continual testing of propositions to determine the benefits for the customer and the software organization [RK16]. Furthermore, techniques such as A/B testing, where features such as text, layouts, images and colors are manipulated systematically and customer reaction is monitored, can be used to effectively identify value-adding features [OAB12]. Continuous experimentation consists of repeated Build-Measure-Learn cycles [FS17] and is one of the attributes DevOps teams must exhibit to support DevOps capabilities [SBQ18b]. Another aspect of continuous experimentation that positively impacts a DevOps environment is giving developers time to experiment with new ideas, driving the discovery process, and inspiring team members to share their discoveries with the rest of the organization [H+16]. In addition, an experimental culture involves more customer interactions [RK16].

Integration with Scrum

Continuous experimentation can easily be integrated into a standard Scrum process. In order for continuous experimentation to work developers need to be given the time to experiment with new ideas, driving the discovery process, and inspiring team members to share their discoveries with the rest of the organization. During sprint planning this time can be allocated to the developers. At the end of the sprint, during sprint retrospective, developers then have the time to reflect on the changes and their outcome. Consequently, the changes can be discarded or permanently added to the workflow.

When using continuous experimentation it is important that the targeted innovations work towards a goal, are prioritized and time-boxed ("Team lead 1", personal communication, Jan. 20, 2025)("Team lead 2", personal communication, Jan. 27, 2025)("CEO", personal communication, Jan. 21, 2025)("Developer 3", personal communication, Feb. 10, 2025)("IT-manager", personal communication, Jan. 27, 2025). Furthermore, there are limitations to the usefulness of continuous experimentation as this strongly depends on the type of organization and its products ("Agile coach", personal communication, Jan. 30, 2025). Additionally, the bigger the organization and the product, the more these innovations may collide and lead to problems ("Team lead 2", personal communication, Jan. 27, 2025).

7.2.11 Infrastructure Automation

Practice Definition

Within this thesis infrastructure automation is defined as a broad term that refers to automating infrastructure-related tasks ("Team lead 2", personal communication, Jan. 27, 2025), in order to accelerate build-deploy cycles [Mer21], improve the consistency of infrastructure, team productivity, and repeatability of activities [dFJT16]. As this matches the definition for configuration management automation [TV14], which was not found during the systematic literature review and therefore added from different sources, the two practices are merged within this thesis.

A subset of infrastructure automation is Infrastructure as Code (IaC) ("Team lead 2", personal communication, Jan. 27, 2025), which is defined as the provision of infrastructure elements as code and the configuration of application servers using pre-set automation logic [H+16]. Application code has a defined format and syntax that follows the rules of the programming language in order to be created. Typically, the code is stored in a version-management system that logs a history of code development, changes and fixes. When code is compiled (built) into an application, the expectation is that a consistent application is created. When the latter occurs, the build process is said to be repeatable and reliable. Practicing IaC means applying the same rigor of application code development to infrastructure provisioning, orchestration and deployment. All configurations should be defined in a declarative way and stored in a version management system, just like application code. With IaC, it becomes possible to quickly provision application environments and deploy application automatically and at scale as needed [LKO16].

Integration with Scrum

Infrastructure automation is an operations practice used to accelerate and stabilize application environment provisioning and automate application deployment. As infrastructure automation is done by applying the same rigor of application code development to infrastructure provisioning, orchestration and deployment, setting it up can be seen as a developers task in the context of Scrum and infrastructure automation can therefore work in conjunction with Scrum.

A set up infrastructure automation makes it much easier and faster to deploy an application, therefore the individual tasks of deploying a new version of an application can be redistributed. Depending on the size of the company setting up infrastructure automation can either be done by an employee with operational expertise who is part of the cross-functional Scrum team, or by a separate team, tasked with providing a basic infrastructure. Either way, usage of the automated application deployment can be done by the developers within the Scrum team. This allows developers to deploy their products faster, without needing to wait on another team to perform deployment tasks.

7.2.12 Microservice Architecture

Practice Definition

As only one description for microservice architecture was found during the systematic literature review and this description focuses more on benefits and downsides than describing what a microservice architecture is, an additional definition was chosen from literature in order to cover the basic ideas. Therefore, within this thesis a microservices architecture is defined as a cloud-native architecture that aims to realize software systems as a package of small services. Each service is independently deployable on a potentially different platform and technological stack. It can run in its own process while communicating through lightweight mechanisms such as RESTful or RPC-based APIs. In this setting, each service is a business capability that can utilize various programming languages and data stores and is developed by a small team [FL15].

From a DevOps point of view microservice architectures have many desirable properties, as they introduce high agility, resilience, scalability, maintainability, separation of concerns, and ease of deployment and operations [AAE16]. Furthermore, due to the greater amount of modularization that microservices provide, new features can be placed in the hands of users faster [New21].

Even though microservices release the rigid structure of monolithic systems due to the independent deployment, this style also introduces a high level of complexity. It is more and more demanded to deploy new code quickly, while guaranteeing reliability and effectiveness. As microservices have more integration points between systems, they suffer from a higher possibility of failure than monolithic systems. It is possible to partially mitigate this problem by making an effort to monitor the services and take appropriate action to malfunctions. Hence, many different monitoring approaches and service discovery mechanisms are developed that pull the status of services dynamically over the network [KM18]. This simplifies many aspects of tracking the system, but lacks in connecting their obtained monitoring data in order to achieve an integrated and holistic view of the behavior and status of the whole microservice infrastructure [BGP11]. In addition, monitoring systems are mostly used in the production environment in order to react quickly to production failures. It is hardly considered to apply monitoring already in the development and testing phase for documentation and in particular architecture comparison in order to receive valuable feedback through the delivery pipeline [KM18].

Integration with Scrum

Microservice architecture can be used in conjunction with Scrum, having no negative impact on the overall Scrum process. In order to use a microservice architecture the developed product needs to be adjusted accordingly, meaning the developed product needs to be set up as a package of small services. Within a Scrum process these small services can then be worked on individually and incrementally by developers, producing deployable increments to the overall product within a sprint.

7.2.13 Continuous Planning

Practice Definition

Within this thesis continuous planning is defined according to Fitzgerald and Stol, as their definition is the most detailed one found during the systematic literature review and it includes/implies other definitions which were found. Therefore, continuous planning is defined as a holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution [FS14] [FS17]. In addition to iteration and release planning, product and portfolio planning activities would also be conducted [Ruh10].

When addressing an ongoing planning problem, time is divided into a number of planning horizons, each lasting a significant period of time. The only form of continuous planning is that which emerges from agile development approaches and is related to sprint iterations or at best, software releases, and is not widespread throughout the organization. However, just as Agile seeks to enable software development to cope with frequent changes in the business environment, the nature of the business environment also requires that planning activities should be done more frequently to ensure alignment between the needs of the business context and software development [LKVK09], and also requires a tight integration between planning and execution [Cas01].

Integration with Scrum

As defined the only existing form of continuous planning is that which emerges from agile development approaches, ideally related to sprint iterations, which is the case when working with Scrum. Furthermore, continuous planning already is one of the basic ideas of working with Scrum, as the whole Scrum process revolves around continuous planning. Scrum uses sprints as short planning horizons, with sprint reviews and sprint plannings as interim checkpoints where the situation can be reassessed and plans can be modified/adjusted based on market feedback. Furthermore, Scrum uses a prioritized backlog, which is managed by the Product Owner (PO) and can be reordered continuously, following current requirements.

When using continuous planning in conjunction with Scrum and DevOps the ongoing workflow is much more depend on external factors. Due to the implemented automations, more frequent releases and a faster feedback loop, the ongoing workflow is interrupted far more often by external factors, which need to be addressed immediately. Therefore, a bigger buffer for last-minute adjustments needs to be calculated when planning a sprint or a release ("Team lead 2", personal communication, Jan. 27, 2025). Furthermore, when working with agile software development and therefore more short-term planning and potential last-minute adjustments, it is important not to forget about mid- and long-term plans ("Developer 1", personal communication, Jan. 23, 2025). Finally, as DevOps adds employees with operational expertise to cross-functional Scrum teams, the operational aspects of a release can be considered more diligently and new automations regarding operational tasks can be planned and implemented in an agile manner.

7.2.14 Quality Assurance

Practice Definition

As only one definition for quality assurance was found during the systematic literature review, this definition is also adopted by this thesis. Therefore, within this thesis quality assurance is defined as a practice that helps to ensure that software is delivered not only fast, but also the overall quality of the delivered products is guaranteed. Problems and defects should be found in time, before having a negative impact on production [GGL21].

The following actions can help to achieve that goal [GGL21]:

1. Local code analysis:
   Before submitting the code, each developer can perform a static analysis to analyze the code quality locally and find the code that contains errors or does not meet the defined code style standards and has potential problems.

2. Unit tests:
   Each developer can compile the corresponding unit test code and submit it after the local test is passed. These unit tests can also be used in the integration stage and be added the automated delivery pipeline.

3. Automation tests:
   An automation test can be carried out in the continuous integration stage. According to the definition of the CI/CD pipeline, the corresponding test script is executed to perform module function tests and interface tests. After the tests are passed, it enters the next stage. Before the code is integrated, if the system captures an error, an automatic rollback will be triggered.

4. Automated code quality analysis:
   Automated code quality analysis can be used to generate code quality reports. When the previously performed tests are passed, the quality of the startup code is comprehensively scanned and analyzed. The scan results will present data like the number of bugs, defects, coverage, repetition rate in the form of reports.

5. Code review:
   In addition to automation tests and automated code quality analysis, a code review link can be set up. After the code is submitted, a dedicated person reviews the submitted code to check whether there are logic or business problems, or other coding deficiencies. After problem submissions, the code is fed back to the corresponding developers for improvement.

6. System tests:
   System tests are mainly used to test whether the system meets the users' needs. After a series of previous tests, the user environment is simulated to conduct a complete test of the system. System tests can use a combination of manual and automated tests.

Integration with Scrum

Using Scrum, quality assurance can be integrated as a part of the tasks performed by developers. It does not contradict any parts of the Scrum process, rather it further promotes inspection, one of the key pillars of Scrum, which states that artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems [SS20].

This is particularly true for automated tests like unit tests, automation tests, automated code quality analysis and system tests, as those tests can be written independently from the developed code, describing and testing the desired outcome. As those tests are automated, they can be added to a deployment pipeline and be run frequently, thereby consistently testing whether the developed product deviates from the desired output.

Manual quality assurance tasks like local code analysis and code review can be integrated as a mandatory part of the software development process.

Using and integrating all those tasks into the software development process leads to better software quality and a more stable product. Problems within the product are detected earlier, thereby not negatively effecting the code run in production. Furthermore, it helps to keep the development progress on the right track towards the desired final product.

7.2.15 Test Driven Development (TDD)

Practice Definition

As no definition for Test Driven Development (TDD) could be found during the systematic literature review, one was added from different sources. Within this thesis TDD is defined as an iterative software development methodology that emphasizes writing automated tests before writing actual production code. In contrast to traditional software development where software is produced first and test cases are created later, this approach interlaces programming, the creation of unit tests, and refactoring on the source code. In TDD, developers write a failing test case first, then write the code that passes the test intending to create a suite of tests that ensure the code behaves correctly [Bec03]. When compared to traditional methods, TDD can deliver higher-quality projects in lesser time. TDD implementation needs developers and testers to precisely predict how the program and its features will be utilized in practice. An advantage of TDD is the generation of a regression-test suite as a byproduct; this reduces human manual testing while detecting errors earlier, resulting in faster repairs. Also, as tests are performed at the start of the design cycle in this approach, the time and money spent on debugging at later stages is reduced. This is also why TDD is referred to as test-first development. It needs to be noted that refactoring plays a key role in the TDD process since it allows the improvement in the internal structure of the code and design whilst preserving the external behavior of the code [BCF+21].

TDD is also famously known by the red-green-refactor cycle:

1. Add a test to the test suite
2. (Red) Run all the tests to ensure the new test fails
3. (Green) Write just enough code to get that single test to pass
4. Run all tests
5. (Refactor) Improve the initial code while keeping the tests green
6. Repeat

Integration with Scrum

TDD is a developer practice which alters the order in which developers produce and test software, therefore it can be used in conjunction with Scrum without altering the overall Scrum process. Using TDD developers start software development by first writing tests and developing the actual software afterwards. This also has the benefit of producing a test suit which can be used as a basis for continuous testing.

7.3 Discussion of Related Work

Samarawickrama and Perera [SP17] examined how to extend a Scrum process using DevOps practices, however, they limited their research to the practices continuous integration and continuous delivery. Similarly, Gong et al. [GGL21] introduced a workflow of DevOps which is based on continuous integration, continuous deployment and continuous monitoring, aimed to be used in connection with agile software development methods like Scrum. However, both these studies did not validate their processes, neither implementing their processes in practice, nor using expert interviews. As this thesis considered more practices and validated the defined process, it can be seen as an extension of these studies.

Stahl et al. [SMB17] performed a systematic mapping study on DevOps and continuous practices, focusing on continuous integration, continuous delivery and continuous deployment. They concluded that both, continuous practices and the term DevOps, are vaguely defined and loosely used in the software engineering community. They also stated that the software engineering community is indeed far from reaching a well established consensus regarding the definition of continuous practices and DevOps — particularly their relations to one another. They propose future research into individual methods, practices and tools, often associated with DevOps, and the interplay between them. This thesis addresses the vague definition of DevOps (see Section 2.2) and its practices and tries to find a consensus regarding practice definitions. Furthermore, the proposed research regarding the interplay between practices is addressed and a hierarchy between the practices is defined.

Gwangwadza and Hanslo [GH23] performed a systematic literature review identifying the factors that contribute to the success of DevOps environments within software

organizations, proposing a DevOps environment success factors model. This model consists of 4 themes, being:

1. Collaborative Culture, consisting of the factors communication, collaboration and sharing
2. Organizational Aspects, consisting of the factors team roles setting, organizational culture, training and customer involvement
3. Tooling and Technology, consisting of the factors automation, measurement and tooling
4. Continuous Practices, consisting of the factors continuous integration, delivery, deployment, experimentation and monitoring

As the developed model was not validated, they propose the use of the model in practice as well as confirming or disproving there success factors and themes in future research. Conducting this study many factors identified by Gwangwadza and Hanslo were supported. Those factors being communication, collaboration, sharing, customer involvement, automation, measurement and all the listed continuous practices. However, compared to Gwangwadza and Hanslo this thesis considered even more practices. The factors team roles setting, organizational culture, training and tooling can not be supported by this thesis, as they were not further investigated. However, non of the factors were disproved either. Overall, despite this thesis not implementing their developed model, overlapping factors could be supported due to the performed validation using expert interviews.

Munoz et al. [MNM19] performed a systematic literature review focused on getting information about the state of DevOps implementation in practice, regarding the use of methodologies, frameworks and standards. They identified challenges and barriers of companies trying to implement DevOps, the main problems being a lack of process definitions within the companies, a fear to take risks and change processes, and a lack of expertise when searching for process improvements and best practices. They propose the development of a guideline to implement DevOps. Such a guideline is provided by this thesis (see Section 7.2).

Toh et al. performed two studies regarding DevOps. A systematic mapping study on problems and issues in DevOps practice adoption using a continuous delivery process [TSB21] and a qualitative web survey investigating the advantages and limitation of DevOps adoption to improve the continuous delivery process [TSM19a]. Both studies concluded that further research needs to be done to develop a standardized practice that can be followed by practitioners. Furthermore, following their conducted web survey they concluded that future research needs to perform direct interviews with practitioners. By combining a common Scrum process with DevOps practices and describing their specific interplay, this thesis provides a standardized approach on how to use DevOps in connection with Scrum. Furthermore, as this defined process was initially validated

using expert interviews, both suggestions for future work, proposed by Toh et al., were addressed within this thesis.

Several studies were found where DevOps was examined using expert interviews, who reported their experiences with DevOps in practice. Azad et al. [AHM23] explored the factors that contribute to the success of software development undertaken with DevOps practices, performing an open-ended survey among 72 DevOps professionals, analyzing their feedback and experiences. Similarly, Vassallo et al. [VZR+17] examined the use of continuous delivery practices in a large financial organization, surveying 152 developers. Souza et al. [SSRM19] investigated the use of agile practices in software startups, conducting 14 in-depth semi-structured interviews with the CEO and CTO from early-stage software startups. Furthermore, Macarthy and Bass [MB20] examined DevOps implementation in practice through an exploratory case study, based on interviews with 11 industry practitioners across 9 organizations. Finally, Lwakatare et al. [LKK+19] explored the implementation of DevOps in practice, performing interviews with 26 practitioners from 5 different small and medium sized companies. As all these studies collected their data using only interviews, therefore their data was limited and their results could not be directly generalized. Furthermore, they focused on DevOps in a general sense, disregarding specific adaptations of DevOps. As this thesis combined a systematic literature review with a validation conducting expert interviews, this thesis further examined DevOps, adding a different methodology and addressing these research gaps. This can also be seen as more practices were found during this study than within the listed individual studies above. Furthermore, by combining a common Scrum process with DevOps practices, a specific application of DevOps was examined, as proposed by Azad et al. [AHM23].

In summary, it can be said that this thesis provides value in several points. The problem with vague definitions of DevOps and its practices is addressed by conducting a systematic literature review and providing comprising descriptions, which existing individual studies are lacking. Furthermore, by combining a common Scrum process with DevOps practices, a specific application of DevOps is examined and a guideline on how to implement DevOps in practice is given. Finally, by validating the defined process using expert interviews, the defined process is more likely to work in practice and help companies to adapt their working process towards enabling agile DevOps and achieving a competitive advantage in the marketplace.

CHAPTER 8

Conclusion

During the initial state-of-the-art literature analysis it was established that DevOps is subject to continuous evolution ever since its introduction based on a discussion between Andrew Clay and Patrick Debois at the Agile Conference in Toronto in 2008 and that no common understanding of what it entails has been achieved yet [Hü12] [JbAPT16]. This realization led to the perceived necessity to further classify existing DevOps research in order to get a better understanding of what has been done already, the necessity for a clear overview of which practices DevOps entails and a clear description of what these practices actually mean, as most existing research seems to name practices without any context or explanation. All these points were confirmed while executing the systematic literature review and the subsequent process validation using expert interviews, emphasizing the necessity of this thesis. The systematic literature review showed, that despite there being many papers focused on characterizing and clarifying DevOps, the listed DevOps practices within these papers were vastly different and in many cases did not contain any further description, definition or context regarding the mentioned DevOps practices. The different DevOps descriptions, practices and tools listed within these papers made it clear, that the scientific community still has not settled on a clear DevOps definition and implementation approach. This was also substantiated during the expert interviews. Despite all experts having DevOps experience ranging from 3 to 20+ years, there was no clear consensus among them regarding the importance/necessity of individual DevOps practices, however, tendencies could still be seen, which made it possible to establish a hierarchy among the practices within this thesis.

Another gap in existing scientific research that was identified during the initial state-of-the-art literature analysis was, that DevOps is often practiced without a proper framework for software process management [SP17], complicating the adoption of DevOps for companies working with widespread agile software development processes like Scrum. It was established that most references in existing literature between DevOps and concrete agile methodologies are due to them being agile processes, however, information on how

to practically combine them with DevOps are limited, leading to the perceived necessity for further research regarding the interplay between concrete agile methodologies and DevOps. This gap was illustrated by the systematic literature review. While nearly all selected papers mentioned the umbrella term agile (software) development, nearly half the papers did so without mentioning any concrete agile methodology. The agile methodology mentioned namely most frequently was Scrum, underpinning this studies selection of a Scrum process as a suitable representative of agile methodologies to extend with DevOps practices. However, despite nearly half of the selected studies mentioning Scrum, hardly any of them went into it in further detail, proving the point that there is still great potential for further research regarding the interplay between the two methodologies DevOps and Scrum.

Finally, initial state-of-the-art literature analysis claimed, that while Agile and DevOps complement each other, they should not be considered as replacements for each other [SRA22]. DevOps — the approach, the practices, even the tools — all build on and extend Agile. DevOps and Agile require the same kind of commitment and energy, the same approach to cultural and organizational change, the same focus on iterative and incremental change and on feedback and learning [Bir17]. As Agile and DevOps were said to complement each other, this should also hold for Scrum, as it is the most prominent agile methodology. However, as existing research in the context of Scrum is limited, there was still a perceived necessity to fill this gap. Performing the systematic literature review, selecting and describing the most frequently used DevOps practices, putting them in the context of Scrum and validating the defined process using expert interviews, it became clear that these statements about Agile and DevOps complementing each other are supported. Not only was it possible to link all selected DevOps practices with a common Scrum process, but also could many synergies between the methodologies be observed. While in many cases DevOps practices did match the basic ideas, values and ideas of the Scrum framework, in others the general Scrum framework enabled DevOps practices. Additionally, there were DevOps practices that could simply be integrated into a Scrum process as e.g. a developer practice, which did not affect the overall process.

8.1 Limitations and Future Work

While this thesis provides a theoretical concept on how to extend a common Scrum process using DevOps practices, several limitations need to be acknowledged.

Firstly, the proposed process was developed purely in theory and evaluated using expert interviews rather than being implemented in practice. Therefore, the real-world applicability and effectiveness of the approach remains untested. Future research could address this limitation by conducting case studies implementing the defined process in practice.

Secondly, due to time constraints, the process validation was limited to eight expert interviews. While these experts had a lot of personal experience working with Scrum and DevOps and provided valuable insights, a larger and more diverse group could have enhanced the reliability and breadth of the findings. Further research could expand

the expert pool and incorporate perspectives from more practitioners across different industries and roles.

Additionally, the systematic literature review was conducted using only one research database, namely Scopus, and papers needed to be available via the university library service (Vienna University of Technology) during the time of search or be freely available on the web, which may have limited the scope of relevant studies included in the analysis. This restriction could have led to the exclusion of valuable insights available in other databases. Expanding the search to multiple sources would provide a more comprehensive understanding of the topic.

Moreover, the study only considered rigorously peer-reviewed literature written in English, so books, theses, workshop papers and gray literature as well as publications in other languages than English were excluded. These constraints may have excluded important contributions from different sources as well as non-English research communities, potentially limiting the global perspective of the findings. Future research could incorporate studies from different sources and in multiple languages to achieve a broader representation of existing knowledge.

Furthermore, papers covering the extension of DevOps beyond development and operations towards other departments, such as security (DevSecOps), privacy (DevPrivOps) or business (BizDevOps), were excluded. While this decision was necessary to maintain a focused scope, future work could explore these extensions.

Finally, future research could dig deeper into the DevOps practice Behavior Driven Development (BDD), which was disregarded in this study due to expert feedback. As this practice was found in several studies, there seems to be interest within the scientific community, however, experts "Developer 1" and "IT-manager" stated that BDD sounds nice in theory, but is not feasible in practice, as specifications are often incomplete or faulty, assumptions about behavior are made and stakeholders change requirements over time. Therefore, future research could examine how these gaps could be closed and how this practice could be used in practice. Additionally, as "Agile coach" pointed out the future potential of this practice connected to advancements in artificial intelligence and artificially generated tests, this connection could be investigated further.

List of Figures

List of Tables

Glossary

DevOps The term DevOps is a blend of development (representing software developers, including programmers, testers, and quality assurance personnel) and operations (representing the experts who put software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians).. xiii, xv, 1–5, 14–17, 19–21, 23–31, 33, 34, 38, 39, 41, 43, 44, 47, 49, 50, 52–54, 57–61, 65–67, 71–85, 87–91, 93–96, 98, 100, 102, 103, 106–111, 113, 115

Acronyms

A&I abstracting and indexing. 25

ASDM agile software development method. 5, 6

BDD Behavior Driven Development. 64, 65, 67, 86, 111

CI/CD Continuous integration and Continuous delivery/deployment. 41, 46, 56, 57, 61, 67, 79, 82, 85, 93, 98, 99, 104

Dev Development. 3, 14

DoD Definition of Done. 10, 12, 13, 52, 79, 93

IaC Infrastructure as Code. 43, 44, 58, 83, 84, 101

Infosec information security. 2

IS information systems. 4

NIST National Institute of Standards and Technology. 40, 54, 82, 96

Ops Operations. 3, 14

PaaS Platform-as-a-Service. 67

PO Product Owner. 9–13, 65

QA quality assurance. 2

ROI Return on Investment. 10, 13

RQ research question. 19, 24

SaaS Software-as-a-Service. 37, 67

SM Scrum Master. 9–11

TDD Test Driven Development. 63–65, 67, 85, 105, 106

VCS Version Control System. 42, 56, 57, 99

Bibliography

[AAE16] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 44–51, 2016.

[ACM24] https://authors.acm.org/journals/journals-indexing-list, 5 2024.

[AHM23] N. Azad, S. Hyrynsalmi, and M. Mäntymäki. *Understanding DevOps Critical Success Factors: Insights from Professionals*, volume 14316 LNCS. 2023.

[AI13] A. Aitken and V. Ilango. A comparative analysis of traditional software engineering and agile software development. pages 4751–4760. IEEE, 2013.

[AMB+16] Cesar Javier Gil Arrieta, Jorge Luis Diaz Martinez, Mario Orozco Bohórquez, Alexis Kevin De La Hoz Manotas, Eduardo Miguel De La Hoz Correa, and Roberto Cesar Morales Ortega. Agile testing practices in software quality: State of the art review. 2016.

[AMS+23] Muhammad Azeem Akbar, Sajjad Mahmood, Muhammad Shafiq, Ahmed Alsanad, Abeer Abdul-Aziz Alsanad, and Abdu Gumaei. Identification and prioritization of devops success factors using fuzzy-ahp approach. *Soft computing (Berlin, Germany)*, 27:1907–1931, 2023.

[AR19] P. Agrawal and N. Rawat. Devops, a new approach to cloud development testing. 2019.

[Atl24] https://www.atlassian.com/git/tutorials/what-is-version-control, 2024.

[B+10] Alexandru G Bardas et al. Static code analysis. *Journal of Information Systems Operations Management*, 4:99–107, 2010.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. 2001.

[BCF+21] Maria Teresa Baldassarre, Danilo Caivano, Davide Fucci, Natalia Juristo, Simone Romano, Giuseppe Scanniello, and Burak Turhan. Studying test-driven development and its retainment over a six-month time span. *The Journal of systems and software*, 176:110937, 2021.

[Bec03] Kent Beck. *Test-driven development : by example*. Addison-Wesley, 1st edition edition, 2003.

[BEK20] Leonard Peter Binamungu, Suzanne M. Embury, and Nikolaos Konstantinou. Characterising the quality of behaviour driven development specifications. volume 383, pages 87–102, 2020.

[BGP11] Tobias Brückmann, Volker Gruhn, and Max Pfeiffer. Towards real-time monitoring and controlling of enterprise architectures using business software control centers. In *Software Architecture: 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings 5*, pages 287–294, 2011.

[Bir17] Jim Bird. *Understanding Agile DevOps*. O'Reilly Media, Inc., 1st edition. edition, 2017.

[Bla09] Paul E Black. Static analyzers in software engineering. *CrossTalk, The Journal of Defense Software Engineering*, pages 16–17, 2009.

[BSMG12] Mario Bernhart, Stefan Strobl, Andreas Mauczka, and Thomas Grechenig. Applying continuous code reviews in airport operations software. pages 214–219, 2012.

[BT03] B Boehm and R Turner. Using risk to balance agile and plan-driven methods. *Computer (Long Beach, Calif.)*, 36(6):57–66, 2003.

[BW95] Deborah L Butler and Philip H Winne. Feedback and self-regulated learning: A theoretical synthesis. *Review of educational research*, 65:245–281, 1995.

[Cas01] Knight S Casper. space exploration through continuous planning [j]. *IEEE Intelligent Systems*, 16:70–75, 2001.

[Cen22] IEEE Author Center. Abstracting and indexing databases, 4 2022. https://journals.ieeeauthorcenter.ieee.org/when-your-article-is-published/abstracting-indexing-ai-databases/.

[Cho08] Juyun Cho. ISSUES AND CHALLENGES OF AGILE SOFTWARE DEVELOPMENT WITH SCRUM. *Issues In Information Systems*, 9(2):188–195, 2008.

[Clo11] Hybrid Cloud. The nist definition of cloud computing. *National institute of science and technology, special publication*, 800:145, 2011.

[Col01] Robert E Cole. From continuous improvement to continuous innovation. *Quality Management Journal*, 8:7–21, 2001.

[dF17] R de Feijter. Towards the adoption of devops in software product organizations: A maturity model approach. 2017.

[dFJT16] B.B. Nicolau de França, H. Jeronimo, and G.H. Travassos. Characterizing devops by hearing multiple voices. pages 53–62, 2016.

[Dig24] The 17th state of agile report, 1 2024.

[DKSD14] Melanie Diepenbeck, Ulrich Kühne, Mathias Soeken, and Rolf Drechsler. *Behaviour Driven Development for Tests and Verification*, volume 8570, pages 61–77. Springer International Publishing, 2014.

[DSM+22] I.-C. Donca, O.P. Stan, M. Misaros, D. Gota, and L. Miclea. Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors*, 22, 2022.

[DWAJ+05] Mary Dixon-Woods, Shona Agarwal, David Jones, Bridget Young, and Alex Sutton. Synthesising qualitative and quantitative evidence: a review of possible methods. *Journal of health services research policy*, 10:45–53, 2005.

[EAD17] F.M.A Erich, Chintan Amrit, and Maia Daneva. A qualitative study of devops usage in practice. *Journal of software : evolution and process*, 29:e1885–n/a, 2017.

[Els24] https://www.elsevier.com/, 2024.

[FL15] Martin Fowler and James Lewis. Microservices: Nur ein weiteres konzept in der softwarearchitektur oder mehr. *Objektspektrum*, 1:14–20, 2015.

[FOR+23] Muhammad Shoaib Farooq, Uzma Omer, Amna Ramzan, Mansoor Ahmad Rasheed, and Zabihullah Atal. Behavior driven development: A systematic literature review. *IEEE access*, 11:1–1, 2023.

[FS14] Brian Fitzgerald and KlaasJan Stol. Continuous software engineering and beyond: Trends and challenges. page 1 – 9. Association for Computing Machinery, 2014. Cited by: 114; All Open Access, Bronze Open Access, Green Open Access.

[FS17] B. Fitzgerald and K.-J. Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[GBM19] L.G. Guseila, D.-V. Bratu, and S.-A. Moraru. Continuous testing in the development of iot applications. 2019.

[GG17] Georges Bou Ghantous and Asif Gill. Devops: Concepts, practices, tools, benefits and challenges. page 16, 2017.

[GGL21] L. Gong, G. Gong, and X. Ling. Practice of software development based on devops. *Forest Chemicals Review*, 2021:1287–1302, 2021.

[GH23] A. Gwangwadza and R. Hanslo. *Towards the Success of DevOps Environments in Software Organizations: A Conceptual Model Approach*, volume 1774 CCIS. 2023.

[GHM18] Davrondzhon Gafurov, Arne Erik Hurum, and Martin Markman. Achieving test automation with testers without coding skills: an industrial report. pages 749–756, 2018.

[Git14] https://git-scm.com/book/en/v2, 2014.

[GS21] M. Gokarna and R. Singh. Devops: A historical review and future works. pages 366–371, 2021.

[GZCP20] Jonathan Guerrero, Karen Zúñiga, Camilo Certuche, and César Pardo. A systematic mapping study about devops. *Journal de Ciencia e Ingeniería*, 12, 2020.

[H+16] Joonas Hamunen et al. Challenges in adopting a devops approach to software development and operations. 2016.

[Hum10] Humble. *Continuous delivery: reliable software releases through build, test, and deployment automation*. 2010.

[Hü12] Michael Hüttermann. *DevOps for Developers*. Apress : Imprint: Apress, 1st ed. 2012.. edition, 2012.

[JbAPT16] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops?: A systematic mapping study on definitions and practices. page 11. Association for Computing Machinery, 2016.

[KBD19] Avita Katal, Vinayak Bajoria, and Susheela Dahiya. Devops: Bridging the gap between development and operations. pages 1–7, 2019.

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[KHD+16] By Gene Kim, Jez Humble, Patrick Debois, , and John Willis. *The DevOps handbook*, volume First Edit. IT Revolution Press, 2016.

[KLJ18] O. Krancher, P. Luther, and M. Jost. Key affordances of platform-as-a-service: Self-organization and continuous feedback. *Journal of Management Information Systems*, 35:776–812, 2018.

[KM18] Martin Kleehaus, Ömer Uludag, and Florian Matthes. Towards a continuous feedback loop for service-oriented environments. In *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, pages 126–134, 2018.

[Lib24] https://nuim.libguides.com/az/databases, 2024.

[LKK+19] L.E. Lwakatare, T. Kilamo, T. Karvonen, T. Sauvola, V. Heikkilä, J. Itkonen, P. Kuvaja, T. Mikkonen, M. Oivo, and C. Lassenius. Devops in practice: A multiple case study of five companies. *Information and Software Technology*, 114:217–230, 2019.

[LKO16] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. An exploratory study of devops: Extending the dimensions of devops with practices. 6 2016.

[LKVK09] Laura Lehtola, Marjo Kauppinen, Jarno Vähäniitty, and Marko Komssi. Linking business and requirements engineering: is solution planning a missing activity in software product companies? *Requirements engineering*, 14:113–128, 2009.

[Lwa17] Lucy Ellen Lwakatare. Devops adoption and implementation in software development practice: concept, practices, benefits and challenges. 2017.

[May15] Philipp Mayring. *Qualitative Inhaltsanalyse : Grundlagen und Techniken*. Beltz, 12., überarbeitet... edition, 2015.

[MB20] R.W. Macarthy and J.M. Bass. An empirical taxonomy of devops in practice. pages 221–228, 2020.

[MD21] V. P. Munteanu and P. Dragos. The case for agile methodologies against traditional ones in financial software projects. *European Journal of Business and Management Research*, 6, 2021.

[Mer21] P Merson. Principles for microservice design: Think ideals, rather than solid. *The InfoQ eMag*, pages 6–13, 2021.

[MG20] Krikor Maroukian and Stephen R. Gulliver. Leading devops practice and principle adoption. 8 2020.

[MHKB15] Matthew A. McCarthy, Lorraine M. Herger, Shakil M. Khan, and Brian M. Belgodere. Composable devops: Automated ontology based devops maturity analysis. 2015.

[Mis17] Abhishek Mishra. Introduction to behavior-driven development. *iOS Code Testing: Test-Driven Development and Behavior-Driven Development with Swift*, pages 317–327, 2017.

[ML05] K.W Miller and D.K Larson. Agile software development: human values and culture. *IEEE technology society magazine*, 24(4):36–42, 2005.

[MMA+20] L.D. Aguiar Monteiro, D.S.M. Pessoa Monteiro, W.H. Carvalho Almeida, A. Cavalcanti De Lima, and I.S. Sette. Methods of implementation, maturity models and definition of roles in devops frameworks: A systematic mapping. pages 1766–1773, 2020.

[MNM19] M. Muñoz, M. Negrete, and J. Mejía. *Proposal to Avoid Issues in the DevOps Implementation: A Systematic Literature Review*, volume 930. 2019.

[New21] Sam Newman. *Building microservices.* " O'Reilly Media, Inc.", 2021.

[NS13] Steve Neely and Steve Stolt. Continuous delivery? easy! just change everything (well, maybe it is not that easy). pages 121–128, 2013.

[OAB12] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *2012 38th euromicro conference on software engineering and advanced applications*, pages 392–399, 2012.

[PEL21] M.A.P. Pardo, H.A.O. Erazo, and C.A.C. Lozada. Documenting and implementing devops good practices with test automation and continuous deployment tools through software refinement. *Periodicals of Engineering and Natural Sciences*, 9:854–863, 2021.

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, page 10, 2008.

[PMSS03] Prashant Palvia, En Mao, A.F. Salam, and Khalid S. Soliman. Management Information Systems Research: What's There in a Methodology? *Communications of the Association for Information Systems*, 11, 2003.

[PP04] Mary Poppendieck and Tom Poppendieck. *Lean software development : an agile toolkit.* Addison-Wesley, 4. print. edition, 2004.

[PSHH04] A Parrish, R Smith, D Hale, and J Hale. A field study of developer pairs: productivity impacts and implications. *IEEE software*, 21(5):76–79, 2004.

[RK16] Riungu-Kalliosaari. *DevOps adoption benefits and challenges in practice: A case study.* 2016.

[Ruh10] Günther Ruhe. *Product release planning: methods, tools and applications.* CRC Press, 2010.

[RYA20] S. Rafi, W. Yu, and M.A. Akbar. Rmdevops: A road map for improvement in devops activities in context of software organizations. pages 413–418, 2020.

[SB01] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Pearson, 10 2001.

[SB02] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*. Series in agile software development. Prentice Hall, Upper Saddle River, NJ, 2002.

[SBO18a] Mali Senapathi, Jim Buchan, and Hady Osman. Devops capabilities, practices, and challenges: Insights from a case study. page 11, 2018.

[SBO18b] Mali Senapathi, Jim Buchan, and Hady Osman. Devops capabilities, practices, and challenges: Insights from a case study. volume 137700, pages 57–67. ACM, 2018.

[Sch04] Ken Schwaber. *Agile project management with Scrum*. Mcirosoft Press, 2004.

[Scr16] ScrumAlliance. Scrum Values, 2016. https://www.scrum.org/.

[Scr17] Scrum Inc. Scrum Process, 2017. https://ichi.pro/de/9-schritte-zum-starten-von-scrum-von-grund-auf-neu-51523235504000.

[Scr24a] https://www.scrumalliance.org/, 2024.

[Scr24b] https://www.scruminc.com/, 2024.

[Scr24c] https://www.scrum.org/, 2024.

[SDW18] J. Shah, D. Dubaria, and J. Widhalm. A survey of devops tools for networking. pages 185–188, 2018.

[SG18] Sánchez-Gordón. *Characterizing DevOps culture: a systematic literature review*. 2018.

[Sim98] Herbert Alexander Simon. *The sciences of the artificial*. MIT Press, Cambridge, Mass. [u.a.], 3. ed., 2. print.. edition, 1998.

[SLC+19] Clauirton Siebra, Rosberg Lacerda, Italo Cerqueira, Jonysberg P Quintino, Fabiana Florentin, Fabio B Q da Silva, and Andre L M Santos. Empowering continuous delivery in software development: the devops strategy. In *Software Technologies: 13th International Conference, ICSOFT 2018, Porto, Portugal, July 26-28, 2018, Revised Selected Papers 13*, pages 247–265, 2019.

[SMB17] D. Ståhl, T. Mårtensson, and J. Bosch. Continuous practices and devops: Beyond the buzz, what does it all mean? volume 2017-Janua, pages 440–448, 2017.

[Son15] Mitesh Soni. End to end automation on cloud with build pipeline: The case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery. page 85 – 89. Institute of Electrical and Electronics Engineers Inc., 2015. Cited by: 44.

[SP17] Saliya Sajith Samarawickrama and Indika Perera. Continuous scrum: A framework to enhance scrum with DevOps. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, volume 2018-, pages 1–7. IEEE, 2017.

[Spr24] https://www.springer.com/gp/computer-science/lncs/information-on-abstracting-and-indexing/799288, 5 2024.

[SRA22] Navin Sabharwal, Raminder Rathore, and Udita Agrawal. *Hands-On Guide to AgileOps : A Guide to Implementing Agile, DevOps, and SRE for Cloud Operations.* Apress L. P., 2022.

[SS17] P. Schaefer and D. Söllner. Devops by scrumban. *HMD*, 54:251–260, 2017.

[SS20] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 21(19), 2020.

[SSRM19] R. Souza, F. Silva, L. Rocha, and I. Machado. Investigating agile practices in software startups. pages 317–321, 2019.

[Tec24] TechTarget. Sandbox, 2024.

[TSB21] M.Z. Toh, S. Sahibuddin, and R.A. Bakar. A review on devops adoption in continuous delivery process. pages 98–103, 2021.

[TSM19a] M. Zulfahmi Toh, S. Sahibuddin, and M.N. Mahrin. Adoption issues in devops from the perspective of continuous delivery pipeline. volume Part F1479, pages 173–177, 2019.

[TSM19b] M. Zulfahmi Toh, Shamsul Sahibuddin, and Mohn Naz'ri Mahrin. Adoption issues in devops from the perspective of continuous delivery pipeline. page 5, 2019.

[TV14] Mischa Taylor and Seth Vargo. *Learning chef: a guide to configuration management and automation.* " O'Reilly Media, Inc.", 2014.

[vBdKT19] M. van Belzen, D. de Kruijff, and J. Trienekens. Success factors of collaboration in the context of devops. pages 26–34, 2019.

[Ven05] Vivek Venugopalan. Cvs best practices. 2005.

[vHHW+09] André van Hoorn, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey, and Dennis Kieselhorst. Continuous monitoring of software services: Design and application of the kieker framework. 2009.

[Vid09] Vidgen. Coevolving systems and the organization of agile software development. *Information systems research : ISR.*, 20:355, 2009.

[Vir15] Manish Virmani. Understanding devops bridging the gap from continuous integration to continuous delivery. page 78 – 82. Institute of Electrical and Electronics Engineers Inc., 2015. Cited by: 66.

[VZR+17] C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. Di Penta, and A. Zaidman. Continuous delivery practices in a large financial organization. pages 519–528, 2017.

[WFW+19] A. Wiedemann, N. Forsgren, M. Wiesche, H. Gewald, and H. Krcmar. Research for practice: The devops phenomenon. *Communications of the ACM*, 62:44–49, 2019.

[WH06] Thomas Wilde and Thomas Hess. Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung. 2006.

[Wie18] Anna Wiedemann. It governance mechanisms for devops teams – how incumbent companies achieve competitive advantages. volume 2018-, pages 4931–4940, 2018.

[Wie21] TU Wien. Business informatics, 7 2021. https://informatics.tuwien.ac.at/master/business-informatics/.

[WMMR06] Roelf J Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements engineering*, 11:102–107, 2006.

[ZND18] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. Version control system: A review. *Procedia Computer Science*, 135:408–415, 2018.

Appendix A: Included studies for 2008-2023

[1] P. Agrawal and N. Rawat. Devops, a new approach to cloud development testing. 2019.

[2] I. Alves and C. Rocha. Qualifying software engineers undergraduates in devops - challenges of introducing technical and non-technical concepts in a project-oriented course. pages 144–153, 2021.

[3] N. Azad, S. Hyrynsalmi, and M. Mäntymäki. *Understanding DevOps Critical Success Factors: Insights from Professionals*, volume 14316 LNCS. 2023.

[4] P. Batra and A. Jatain. Measurement based performance evaluation of devops. pages 757–760, 2020.

[5] S. Bobrovskis and A. Jurenoks. A survey of continuous integration, continuous delivery and continuous deployment. volume 2218, pages 314–322, 2018.

[6] K. Byrne and A. Cevenini. *Aligning DevOps Concepts with Agile Models of the Software Development Life Cycle (SLDC) in Pursuit of Continuous Regulatory Compliance*, volume 1029 LNEE. 2023.

[7] Maximilien de Bayser, Leonardo G Azevedo, and Renato Cerqueira. Researchops: The case for devops in scientific applications. pages 1398–1404. IEEE, 2015.

[8] I.-C. Donca, O.P. Stan, M. Misaros, D. Gota, and L. Miclea. Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors*, 22, 2022.

[9] K. Duan, J.M. Caballero, and X. Jing. Scrum-based development model:improve the engineering quality and testing method of blockchain projects. pages 807–811, 2023.

[10] F.M.A. Erich, C. Amrit, and M. Daneva. A qualitative study of devops usage in practice. *Journal of Software: Evolution and Process*, 29, 2017.

[11] J. Faustino, R. Pereira, B. Alturas, and M.M. da Silva. Agile information technology service management with devops: An incident management case study. *International Journal of Agile Systems and Management*, 13:339–389, 2020.

[12] B. Fitzgerald and K.-J. Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[13] Brian Fitzgerald and KlaasJan Stol. Continuous software engineering and beyond: Trends and challenges. page 1 – 9. Association for Computing Machinery, 2014. Cited by: 114; All Open Access, Bronze Open Access, Green Open Access.

[14] R. Gniadek, M. O'Neil Ellis, G. Back, and K. Cameron. Integrating devops to enhance student experience in an undergraduate research project. 2022.

[15] K Gohil, N Alapati, and S Joglekar. Towards behavior driven operations (bdops). volume 2011, pages 262–264. IET, 2011.

[16] M. Gokarna and R. Singh. Devops: A historical review and future works. pages 366–371, 2021.

[17] L. Gong, G. Gong, and X. Ling. Practice of software development based on devops. *Forest Chemicals Review*, 2021:1287–1302, 2021.

[18] R.K. Gupta, M. Venkatachalapathy, and F.K. Jeberla. Challenges in adopting continuous delivery and devops in a globally distributed product team: A case study of a healthcare organization. pages 30–34, 2019.

[19] L.G. Guseila, D.-V. Bratu, and S.-A. Moraru. Continuous testing in the development of iot applications. 2019.

[20] L.G. Guseila, D.-V. Bratu, and S.-A. Moraru. Devops transformation for multi-cloud iot applications. 2019.

[21] A. Gwangwadza and R. Hanslo. *Towards the Success of DevOps Environments in Software Organizations: A Conceptual Model Approach*, volume 1774 CCIS. 2023.

[22] A. Hemon, B. Fitzgerald, B. Lyonnet, and F. Rowe. Innovative practices for knowledge sharing in large-scale devops. *IEEE Software*, 37:30–37, 2020.

[23] R. Hernández, B. Moros, and J. Nicolás. Requirements management in devops environments: a multivocal mapping study. *Requirements Engineering*, 28:317–346, 2023.

[24] Martin Kleehaus, Ömer Uludag, and Florian Matthes. Towards a continuous feedback loop for service-oriented environments. In *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, pages 126–134, 2018.

[25] O. Krancher, P. Luther, and M. Jost. Key affordances of platform-as-a-service: Self-organization and continuous feedback. *Journal of Management Information Systems*, 35:776–812, 2018.

[26] L.E. Lwakatare, T. Kilamo, T. Karvonen, T. Sauvola, V. Heikkilä, J. Itkonen, P. Kuvaja, T. Mikkonen, M. Oivo, and C. Lassenius. Devops in practice: A multiple case study of five companies. *Information and Software Technology*, 114:217–230, 2019.

[27] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. An exploratory study of devops: Extending the dimensions of devops with practices. 6 2016.

[28] R.W. Macarthy and J.M. Bass. An empirical taxonomy of devops in practice. pages 221–228, 2020.

[29] L.D. Aguiar Monteiro, D.S.M. Pessoa Monteiro, W.H. Carvalho Almeida, A. Cavalcanti De Lima, and I.S. Sette. Methods of implementation, maturity models and definition of roles in devops frameworks: A systematic mapping. pages 1766–1773, 2020.

[30] S. Mosser, C. Pulgar, and V. Reinharz. Modelling agile backlogs as composable artifacts to support developers and product owners. *Journal of Object Technology*, 21, 2022.

[31] M. Muñoz, M. Negrete, and J. Mejía. *Proposal to Avoid Issues in the DevOps Implementation: A Systematic Literature Review*, volume 930. 2019.

[32] P.A. Nielsen, T.J. Winkler, and J. Nørbjerg. Closing the it development-operations gap: The devops knowledge sharing framework. volume 1898, 2017.

[33] T. Pandiyavathi and B. Sivakumar. *DevOps Challenges and Practices in Software Engineering*, volume 665 LNNS. 2023.

[34] C. Pardo, J. Guerrero, and E. Suescún. Devops model in practice: Applying a novel reference model to support and encourage the adoption of devops in a software development company as case study. *Periodicals of Engineering and Natural Sciences*, 10:221–235, 2022.

[35] M.A.P. Pardo, H.A.O. Erazo, and C.A.C. Lozada. Documenting and implementing devops good practices with test automation and continuous deployment tools through software refinement. *Periodicals of Engineering and Natural Sciences*, 9:854–863, 2021.

[36] M Rajkumar, Anil Kumar Pole, Vittalraya Shenoy Adige, and Prabal Mahanta. Devops culture and its impact on cloud delivery and software development. Institute of Electrical and Electronics Engineers Inc., 2016. Cited by: 22.

[37] V.S. Rani, A.R. Babu, K. Deepthi, and V.R. Reddy. Shift-left testing in devops: A study of benefits, challenges, and best practices. pages 1675–1680, 2023.

[38] S.S. Samarawickrama and I. Perera. Continuous scrum: A framework to enhance scrum with devops. volume 2018-Janua, pages 19–25, 2017.

[39] J. Shah, D. Dubaria, and J. Widhalm. A survey of devops tools for networking. pages 185–188, 2018.

[40] Mitesh Soni. End to end automation on cloud with build pipeline: The case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery. page 85 – 89. Institute of Electrical and Electronics Engineers Inc., 2015. Cited by: 44.

[41] R. Souza, F. Silva, L. Rocha, and I. Machado. Investigating agile practices in software startups. pages 317–321, 2019.

[42] S.S. Sravan, C. Sai Ganesh, K.V.D. Kiran, T. Aakash Chandra, K. Aparna, and T. Vignesh. Significant challenges to espouse devops culture in software organisations by aws : A methodical review. pages 395–401, 2023.

[43] A. Steffens, H. Lichter, and J.S. Döring. Designing a next-generation continuous software delivery system: Concepts and architecture. pages 1–7, 2018.

[44] D. Ståhl, T. Mårtensson, and J. Bosch. Continuous practices and devops: Beyond the buzz, what does it all mean? volume 2017-Janua, pages 440–448, 2017.

[45] C.A. Supit, A.A. Pangeran, K.O.C. Laban, I.I. Gutandjala, and A. Ramadhan. Incorporating cloud native architecture and devops culture to improve company agility. pages 290–294, 2023.

[46] M. Zulfahmi Toh, S. Sahibuddin, and M.N. Mahrin. Adoption issues in devops from the perspective of continuous delivery pipeline. volume Part F1479, pages 173–177, 2019.

[47] M.Z. Toh, S. Sahibuddin, and R.A. Bakar. A review on devops adoption in continuous delivery process. pages 98–103, 2021.

[48] C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. Di Penta, and A. Zaidman. Continuous delivery practices in a large financial organization. pages 519–528, 2017.

[49] Manish Virmani. Understanding devops bridging the gap from continuous integration to continuous delivery. page 78 – 82. Institute of Electrical and Electronics Engineers Inc., 2015. Cited by: 66.

[50] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. Automated capturing and systematic usage of devops knowledge for cloud applications. page 60 – 65. Institute of Electrical and Electronics Engineers Inc., 2015. Cited by: 32; All Open Access, Green Open Access.

[51] Johannes Wettinger, Uwe Breitenbücher, and Frank Leymann. Devopslang - bridging the gap between development and operations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8745 LNCS:108 – 122, 2014. Cited by: 22; All Open Access, Bronze Open Access, Green Open Access.

[52] P. Wolfschwenger, B. Sabitzer, and Z. Lavicza. Design and evaluation of an agile framework for continuous education in software engineering. volume 2022-Octob, 2022.

[53] S. Zhang, D. Fan, J. He, and P. Zhang. A new approach for end to end automation testing platform with cloud computing for 5g product. pages 322–326, 2021.