TURUN YLIOPISTO

Release and Deployment Management in a Scrum Team: A Case Study

Software Engineering
Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Saku Ailasmaa

Supervisors:
Lauri Koivunen (UTU)
Tuomas Mäkilä (UTU)
Petri Karjalainen (If Vahinkovakuutus Oyj)

April 2024

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Software Engineering
Programme: Information and Communication Technology Degree Program
Author: Ailasmaa Saku
Title: Release and Deployment Management in a Scrum Team: A Case Study
Number of pages: 109 pages, 11 appendix pages
Date: April 2024

Release and Deployment Management is a process for managing the delivery of new or updated software. On a practical level, Release and Deployment Management mean all the activities that happen to the software after a new version of it has been created by the Developer and before the software is used by the end customer. This includes such activities as building and packaging the software, running automated tests, deployments to test environments and finally deployment of the release to the live environment. On top of this, all the created release candidates should be easy to keep track of and there should be adequate logging of the aforementioned actions in place.

This thesis uses the migration process of Release and Deployment Management tools as a case study to study how Release and Deployment Management affects the daily work in a Scrum team developing software. The study starts with a literature review on IT literature on Release and Deployment Management to build a foundation on which the conducted research is based. After the literature review the pre-migration process, description of the migration, and the post-migration processes are presented. The remaining part of the study is dedicated to collecting data from the Scrum teams about Release and Deployment Management. Qualitative data is collected by interviewing the members of the Scrum and mixed data is collected by a longitudinal survey that is used to rate both the pre- and post-migration systems.

The thesis concludes that there is no single way to do Release and Deployment Management, instead the process should be tailored for the software that is developed and it has to be iteratively improved with the requirements of the project. Therefore, even though modern CI/CD solutions offer the possibility for extensive automation it still may not be the best choice if that does not fit the project. Nevertheless, a well-implemented Release and Deployment promotes agility and makes it easier to follow predefined Scrum workflows. Moreover, Release and Deployment Management supports software development by increasing the quality of the end product, and the amount of manual work needed is lowered through automation. Furthermore, it makes delivering software quicker, more transparent, predictable, replicable, flexible, smoother, safer and more stable. Additionally, it helps in recovering from failures, tracking errors and changes as well as increasing system integrity. All these factors increase software development teams' productivity, and it saves time. However, Release and Deployment Management is perceived to support the technically oriented roles more than others.

Keywords: Release and Deployment Management, CI/CD, DevOps, Scrum

Diplomityö
Tietotekniikan laitos, Teknillinen tiedekunta
Turun yliopisto

Oppiaine: Ohjelmistotekniikka
Tutkinto-ohjelma: Tieto- ja viestintätekniikan tutkinto-ohjelma (DI)
Tekijä: Ailasmaa Saku
Otsikko: Release and Deployment Management in a Scrum Team: A Case Study
Sivumäärä: 109 sivua, 11 liitesivua
Päivämäärä: Huhtikuu 2024

Julkaisu- ja asennustenhallinta on prosessi päivitetyn tai uuden ohjelmiston toimituksen hallitsemiseksi. Käytännön tasolla julkaisu- ja asennustenhallinta tarkoittaa vaiheita, jotka tapahtuvat ohjelmistolle sen luomisen jälkeen ja ennen kuin se on loppukäyttäjän käytössä. Tähän sisältyy vaiheita, kuten ohjelmiston rakentaminen ja paketointi, automaattisten testien suorittaminen, asennus testiympäristöihin ja viimeisenä asennus tuotantoympäristöön. Prosessiin kuuluu myös mahdollisuus julkaisukandidaattien muutosten seurantaan ja virheiden lokitus edeltävistä vaiheista.

Diplomityössä tehdään tapaustutkimus julkaisu- ja asennustenhallintatyökalujen vaihtamisprosessista, jonka avulla selvitetään, miten julkaisu- ja asennushallinta työkalut vaikuttavat ohjelmistoa kehittävän Scrum-tiimin päivittäiseen työhön. Tutkimus alkaa kirjallisuuskatsauksesta ajankohtaiseen tietoteknilliseen kirjallisuuteen, jota käytetään perustana työssä toteutettavalle tutkimukselle. Tämän jälkeen käydään läpi julkaisu- ja asennustenhallintaprosessi ennen uusien työkalujen käyttöönottoa, käyttöönottoprosessi, ja vielä lopuksi prosessi uusien työkalujen käyttöönoton jälkeen. Työssä kerätään laadullista tietoa haastattelemalla Scrum-tiimin jäseniä, sekä määrällistä ja laadullista tietoa kyselyillä, joita käytetään vanhojen ja uusien työkalujen arvioimiseksi.

Opinnäytetyön johtopäätös on, että ei ole yhtä ainoaa tapaa hallita julkaisuja ja asennuksia, vaan prosessi pitää aina räätälöidä kehitettävän ohjelmiston perusteella ja sitä pitää kehittää iteratiivisesti projektin vaatimusten mukaisesti. Täten vaikka moderneilla CI/CD ratkaisuilla voidaankin automatisoida laajamittaisesti se ei ole välttämättä paras vaihtoehto, jos se ei sovi projektin luonteeseen. Siitä huolimatta hyvin toteutetulla julkaisu- ja asennustenhallinnalla voidaan edistää ketterää kehitystä ja helpottaa ennalta määrättyjen työnkulkujen sujuvuutta Scrum-tiimissä. Sen lisäksi sillä voidaan tukea ohjelmistokehitystä parantamalla lopputuotteen laatua, samalla vähentäen manuaalisen työn määrää automatisaatiolla. Julkaisun- ja asennustenhallinnalla voidaan myös toimittaa ohjelmistoja nopeammin, läpinäkyvimmin, ennalta arvattavammin, toistettavammin, joustavammin, sujuvammin, turvallisemmin ja vakaammin. Lisäksi se auttaa virhetilanteista elpymisessä, virheiden seurannassa ja se parantaa järjestelmien eheyttä. Edellä mainitut tekijät kasvattavat ohjelmistokehitystiimin tuottavuutta ja säästävät aikaa, mutta julkaisu- ja asennustenhallinnan koetaan tukevan teknisesti suuntautuvia rooleja enemmän kuin muita.

Keywords: Julkaisu- ja asennustenhallinta, CI/CD, DevOps, Scrum

Table of contents

1 Introduction

1.1 Motivation

Mature Scrum teams may find themselves in a situation where the tools and practices they have been following ever since they started are becoming outdated. In a situation like this where technical debt has been accumulated over time migrating to newer tools and practices may seem like a daunting task. On the other hand, the team needs to be delivering value but improving the process may enable the team to deliver more value in the future. One such process is the continuous integration and continuous delivery/deployment (CI/CD) cycle.

CI/CD forms the backbone of every scrum team by enabling the team to continuously deliver value to the stakeholders. The CI/CD cycle is crucial for the team because it allows the team to deliver value in small increments and to get constant feedback from the customers. Nowadays there are a myriad of different tools and services to help a scrum team to follow the CI/CD process and to do deployment management, which all have different features and things to offer. Therefore, it might be difficult for a team to decide which tool or service is going to be acquired to assist the team in integrating a CI/CD pipeline. Not only that but different members of the team might prefer certain features over others; Developers may prefer a tool that has a high level of automation to make integration and delivery as smooth as possible, but it may be a business requirement that the used tool has mandatory manual steps to ensure that the delivered items are always verified properly before being deployed to production use. Some tools may offer only certain features from the CI/CD cycle, which means multiple different tools may have to be paired for the team to have proper Release and Deployment Management.

The motivation for this thesis from a practical point of view is to migrate the Release and Deployment Management system from TeamCity and Octopus Deploy to Azure DevOps pipelines. The role of TeamCity in the pre-migration setup is to run through the test coverage and to create a build artifact that can then be deployed using Octopus Deploy. The Release and Deployment Management solution is used by three different Scrum teams that are working on a shared solution. The migration would enable the teams to complete the full continuous integration and continuous delivery/deployment (CI/CD) cycle in Azure DevOps.

The teams are already using Azure DevOps for everything else in the CI/CD cycle and it is also used for Scrum artifacts and ceremonies. Moving to the Azure DevOps pipelines would also allow the process to be automated more, for example, by having stories automatically put into the next state or by updating the release iteration. Thus, the migration would also eliminate waste in terms of manual

work that could be automated. This presents the opportunity to test the limits on how much automation can be done with modern Release and Deployment Management solutions.

1.2 Research Questions and Research Methods

All the mentioned factors make it difficult to choose the right tool for the team and most teams most likely do not have the resources to go around testing different tools and services. Thus, this thesis helps to resolve the problem of choosing between different deployment management tools and services by condensing the aforementioned problems into these four research questions.

RQ1. How does Release and Deployment Management support software development?
RQ2. Do Release and Deployment Management tools make it easier for a Scrum team to follow defined workflows?
RQ3. What aspects of Release and Deployment Management tools are most valued by a Scrum team?
RQ4. How much of the deployment can and should be automated by using modern CI/CD solutions?

Research questions are crucial for the success of this study as it is conducted as a case study and every case study needs a solid foundation that is built on research questions. This is because a case study marries the practical elements of real life with an empirical inquiry. Therefore, the purpose of the predefined research questions is to act as a guide, which helps to find out why and how certain phenomena occurs. [1] [2]

The first research question is answered by first going through the current IT literature on what Release and Deployment Management is, what is achieved by it and what are the claimed benefits. These are put under scrutiny in the interviews and surveys, which try to collect information on how these are seen by a software development team. Research questions two, three and four provide insight into the migration process and the points of views that the different members of the Scrum teams had about the migration. To answer all the research questions, a qualitative research method is combined with a quantitative one. Interviews form the qualitative approach of the study. They are conducted with people from each role present in the Scrum teams. The quantitative approach is a personal opinion survey, which uses the information gathered in the interviews.

Therefore, the aim of the interviews is twofold. Firstly, the interviewing process gathers rich and in-depth qualitative information on the topics of this thesis from multiple points of view, which may have otherwise been overlooked or missed. Secondly, the information gathered in the interviews is used to create a better survey. Additionally, the interviewing process verifies that the terminology used in the

survey is understood by everyone. To achieve these goals the interview is a semi-structured interview. Therefore, the questions in the interviews are going to be as open-ended as possible to allow for as much manoeuvring in the topic as possible, while still all the interviewees answer the same general questions. [2] Chapter 7 elaborates more on the information on the interview.

The survey on the other hand expands on the knowledge gathered during the interviews and more accurate data can be gathered on people's opinions. The survey is a mixed survey, meaning that it will contain both open-ended as well as closed questions. This is due to the nature of personal opinion surveys; the main goal of the survey is to gather quantitative data, but in the case that some of the respondents want to elaborate their answers more there needs to be a possibility for it. The survey is a longitudinal survey by its design, this means that the survey collects data from the same group twice with the same survey. The reason behind this is the unique opportunity to rate the pre- and post-migration systems by using the same exact questions for a fair assessment. [2] More information on the survey is in Chapter 8.

In the end, it can be said that the interviews and the survey both complement each other. If there are only interviews, deviant opinions may have too much weight, which is mitigated by more people answering the survey. On the other hand, if there is only a survey it may not be understandable and precise enough to collect the wanted data. This problem is fixed by having interviews function as a basis for the surveys.

Based on previous research, using interviews as a qualitative research method in the field of Release and Deployment Management is something new and interviews have been used in studies such as [3], [4], [5], [6]. In addition, three of the mentioned studies utilized semi-structured interviews as well. Surveys have been conducted in the field of Release and Deployment Management for example by Puppet Labs [7]. Therefore, as both qualitative and quantitative methods have been successfully used in the field, a possibility for combining them opens up. Combining both qualitative and quantitative research methods means that the research is mixed methods research. Mixed methods research is said to be of superior quality compared to research that is done with only one method [8]. This is because the collected qualitative and quantitative data can be used to verify each other.

Furthermore, help on future migration processes is provided by analysing how the migration process was handled and what was learned from it. The thesis also sheds light on how much should and could be automated by CI/CD pipelines.

1.3 Thesis Structure

The thesis has a linear flow with each consecutive chapter building on the established knowledge from the previous chapter. The starting point for the research is a literature review that is conducted in Chapter 2, the purpose of which is to set the foundation for the following sections. The chapter starts by going through current literature on deployments and releases and finally ties them together to form the term Release and Deployment Management. Section 2.2 is about Scrum and what its relationship with Release and Deployment Management is. This chapter is used as a base for answering RQ2. The next sections 2.3, 2.4, and 2.5 cover Release and Deployment Management from a more detailed view by going through commonplace Release and Deployment Management features, flows and frameworks, which are used to establish theory that is used to answer RQ3 and RQ4. All the chapters in Section 2 collect data that can be used to answer RQ1.

Chapters 3, 4, 5, and 6, are used to describe the research from the case study perspective. Chapter 3 presents the background of the case study. Chapter 4 is the case study's case description, which means that it describes the pre-migration situation and what the goals and reasons for the migration are. Chapter 5 is the case implementation section, which is about the way migration is done and what the technical implementation of the new solution is. Chapter 6 is the case results section, which describes the post-migration process and how it differs from the pre-migration process.

Chapter 7 covers the interviews and Chapter 8 is about the surveys. Both of the chapters have a similar structure, and both are for collecting data to answer the research questions. The sections start by describing how the research methods are used, followed by the reasoning behind the questions. Lastly, the backgrounds of the interviewees and respondents are presented, and the rest of the sections are dedicated to analysing the data.

Chapter 9 discusses the limitations and other factors, which might have affected the outcome of the research. Moreover, the section compares this thesis to previous work done in the same field and how it generates new value in the field of Release and Deployment Management. Chapter 10 concludes the thesis by tying up the established theory from the literature review together with the results of the interviews and surveys to answer the research questions.

2 Background

The purpose of the background chapter is to conduct a literature review on Release and Deployment Management. The chapter begins by defining what Release and Deployment Management is in the scope of this thesis and what it includes. After the term has been established, the chapter continues to other Release and Deployment Management related topics that are relevant to the thesis. The relevant topics are first introduced and then reflected on what is their relation to Release and Deployment Management. The relevant topics include Scrum, DevOps, Continuous Integration / Continuous Deployment and Content Management Systems.

2.1 Release and Deployment Management

The terms deployment management and release management are often times used together to describe the process of managing the delivery of new or updated software. The same process is sometimes referred to by different terms including software distribution and patch management [5]. Therefore, to clarify what is meant by Release and Deployment Management in this thesis both terms are first described individually and then together in the context of this thesis.

2.1.1 Current IT Literature on Release and Deployment Management

Deployment management as a term is rarely used by itself and it is rarely seen alone in the current landscape of IT literature, therefore the meaning of the term is derived from what is Software deployment. Dearle's [9] definition of software deployment consists of multiple interwoven activities including the releasing of software, which includes the required configuration and installation to a production environment. Dearle [9] specifies the software deployment lifecycle is as follows:

1. The software is prepared for delivery from the software producer to its deployer by packaging it with the needed metadata.
2. Receiving and configuring the software to suit the environment that is used by the deployed. This is to be done before any decision about a deployment is made.
3. Describing the prerequisites of the execution environment.
4. Planning the deployment of the software to the targeted distributed execution infrastructure.
5. Preparing the software for execution.
6. Launching, monitoring and terminating the software.

The presented lifecycle makes clear that software deployment is a process that lasts from until the piece of software is developed and tested till the software reaches the end of its life cycle. Software deployment can include some customization and services, for instance, updating, reconfiguration and redeployment. This makes software deployment a post-production activity that can have post-installation activities, which may render software deployment a near-infinite and continuous process. In terms of deployment management, this means that deployment management can be thought of as the management and oversight of this process [9].

Release management, on the other hand, is more established as a term and it was dubbed "a poorly understood and underdeveloped process" by van der Hoek, et al. in 1997 [10]. Albeit a long time has passed since that article was released and release management is better understood now, its complexity has also increased due to customers demanding more challenging and sophisticated systems [5]. The article from van der Hoek, et al. suggested ways to make the complex process easier to manage most of which still hold true today. That includes the five different requirements listed in [10] for a release management system are also laid down in the same article.

1. *Dependencies should be explicit and easily recorded.*
2. *A system should be available through multiple channels.*
3. *The release should involve minimal effort on the part of the Developer.*
4. *The scope of a release should be controllable.*
5. *A history of retrievals should be kept.*

The first requirement is about keeping documentation of dependencies that were present in the released software. The second requirement ensures that the released system is automatically available through the channels the software is distributed to. For example, after the release, the web page should be accessible on the World Wide Web. The third one is in place to make the process as simple as possible for the Developer by not treating the newly released version as a completely new piece of software, rather the Developer should only have to specify what is different from the last release. Thus, by default, the Developer should be able to trust that only the changed parts of the system are affected and that is easy to choose what versions to release. The fourth requirement emphasises that the Developer should be allowed to choose in what circumstances and to whom is the piece of software released. The fifth one ensures that historical records of releases are in place for the Developers and that changes can be communicated to the users. This enables the Developers to keep track of the changes and they can also keep the users in track of the changes [10]. The objective of release management is to make sure that only approved and tested software is distributed to the customers. This is done by controlling releases, patches and updates done to the production environment [11].

2.1.2 ITIL Definitions for Release and Deployment Management

ITIL or Information Technology Infrastructure Library can be seen as one of the single most influential IT service management frameworks. ITIL provides insight into managing, designing and even supporting IT services and it touches on both deployment management and release management [4]. Most of the time in ITIL both of these terms are used together as well, which emphasizes their inherent inseparability.

Deployment management in ITIL is described as the movement of a modified or a new IT service-related entity to another environment. The entity itself may be anything from hardware, software, documentation, processes, or a component. The environment into which the entity is moved may be a live environment or even a test environment. The most important aspects in deployment management to make it successful in practice according to ITIL are firstly the creation and maintenance of efficient ways to deploy IT services to all parts of the organisation. Secondly, ensuring that the deployment of IT service-related entities is done efficiently [12].

Release management according to ITIL is about “Planning, scheduling and controlling the movement of Releases to Test and Live Environments”. The main goal of the Release management process is to secure the live environment's integrity and to ensure that the right components are the ones that are released [13].

Hence, deployment management is the movement and how the moving itself happens and release management is about the greater picture of the whole process, which includes planning scheduling and controlling the movement, or as explained, deployment management. The difference between the two can be made clearer when the definitions for release and deployment are explored. Release is the collection of changed or new IT service-related entities that are tested, developed, and treated as a whole. In simpler terms, it could be said that the release is the new version of the software packaged. Deployment on the other hand is about moving the release between environments. Release and Deployment Management is described as responsible for both the deployments and release management [13].

2.1.3 Release Management Strategies

Release management strategies are described to be based on three different factors: time, features, and quality. In feature-based release management the releases are planned around features, meaning that whenever a feature is deemed complete, it is released. In time-based release management, the releases

are planned around release dates, which means that when the release date comes, all features that are deemed complete are released. The features that were not ready for the release will be then released on the upcoming release dates. In a quality-based release management strategy, once the features have reached a certain quality they are released. It is said that only two of the three aforementioned factors can be fulfilled at once. If features are deployed in time, it may come at the expense of quality. On the other hand, if the features are to be of good quality it requires more time, which means that the deadlines may not be met. To deliver quality features on time means that there will be fewer features deployed, which means that the features are sacrificed. [14]

The most frequently used software release management strategies in open-source projects are feature and time-based strategies. Additionally, there is a hybrid model, which is a mix of feature and time-based strategies. The aim of the hybrid model is to have regular releases, but they should contain a collection of features that are of good quality. A hybrid model could, for instance, work in such a fashion that there are meetings regularly between certain periods to discuss when the next release should be and what features should define it. The model aims to avoid having unfinished features postponing the release, while still providing some frequency in the releases [15]. Texteira [16] has observed that projects tend to initially use a feature-based release management strategy, but then as the project becomes larger and more complex a time-based strategy is seen as the more favourable model.

Michlmayr [17] hypothesised that the triumph of the time-based strategy of the feature-based one in open-source projects is due to the more predictable and steady releases. The predictability and steadiness of the releases seem to contribute positively to the motivation of Developers. Motivation has been said to play such a prominent role, because of the nature of the open-source projects; the work is done by volunteers, which means that they cannot be told to work, rather they work on the projects if they deem it to be of value. According to some Developers, the regular releases make better quality code, because in the feature-based strategy, the Developers do not know when the releases come. Consequently, this causes the Developers to rush their code into the next announced release. This problem is said not to exist in the time-based strategy since the Developers know that if they miss one release they can get their code into the next already known release date [18].

Most of the factors mentioned in why the time-based release management strategy seems to be more popular and better than the feature-based in open-source projects seem to be due to them being open-source projects. In closed-source projects the work done is not done by volunteers, so the people doing the work are motivated by other factors as well. It also assumes to a degree that the feature-based model can only be used to release bigger features. This problem could potentially be alleviated by having a feature-based release management strategy, where even the smallest of features could be considered important enough to warrant a release. The latest advances in the feature-based release

management strategies, and namely DevOps and the Continuous Integration and Continuous Delivery (CI/CD) model are based around this very idea, and it could be summed up in the undying words of Eric Raymond: “Release early. Release Often” [19].

2.1.4 Release and Deployment Management in the Scope of This Thesis

Deployment management in the scope of this thesis is a bit more akin to what can be derived from the IT literature rather than the used ITIL explanation. The reason is that for instance the monitor and other so-called post-installation activities are included in the scope.
The release management part on the other hand still includes the same requirements that were laid down by van der Hoek, et al. Although, the ITIL definition for release management does not mention anything that would be out of scope, but it also does not include all of the intricate details mentioned in the requirements for a release management system. In the end, Release and Deployment Management are used together in this thesis, because as seen in the chapters before Release and Deployment Management are hard to separate from one another since they essentially rely on each other to function. The goal of Release and Deployment Management is to make the deployment of releases easy, controllable, transparent, and to protect the live environment's integrity.

On a more practical level, Release and Deployment Management simply mean all the activities that happen to the software after the new version of it has been created by the Developer and before the software is used by the end customer. This would include such activities as building and packaging the software, running automated tests, deployments to test environments and finally deployment of the release to the live environment. On top of this, all the created release candidates should be easy to keep track of and there should be adequate logging of the aforementioned actions in place.

2.2 Scrum

2.2.1 Basic Information on Scrum

A common anecdote used to explain why Scrum or other agile project management systems are great goes as follows: if someone is driving from point A to B, they cannot program the route onto their vehicle and just press the go button because then the vehicle would just crash the instant other vehicles or obstacles would come their way. Instead, the vehicle needs to be able to control its speed, watch out for other vehicles, and possibly even change the route if some roads are not usable after all. In some instances, even the endpoint might not be where it was initially thought, and thus the vehicle needs to be able to recalibrate its priorities. The anecdote is an adapted version of the one presented in Agile

Project Management With Scrum [20] on why Scrum works, where the power of Scrum is in its inherent ability to allow for independent agency in unexpected situations.

To achieve this agility at the heart of the Scrum is the system of breaking development down into iterative units, which are known as Sprints. Sprints last from one to four weeks during which the development, testing, and business development on the upcoming work all happen concurrently. The smallest units of work in Scrum are called User Stories, which are meant to divide a bigger feature into smaller increments, the Scrum cycle is presented in Figure 1. Consequently, all the team members are going to be working on something all the time and the flow of the process is almost from hand to mouth meaning that after something has been business developed then it is developed and after that, it is tested and then released for the customer. Therefore, the amount of work that is not going to be utilized in the final product is thought to be rather low. This is known as reducing waste, which is at the core of agile methodologies. [21]

Another core principle in Scrum and Agile is to have the product in the hands of the customers as fast as possible to make sure that the customer's original vision was correct and if it was not then the end goal or priorities can be adjusted accordingly [22]. In Scrum, this adjusting can be done whenever the need for it is detected and because the units of work are broken down into smaller pieces it's easier for the customer to see if it was indeed what they wanted. The plan in scrum is not to have a set plan but rather to have a continuously evolving process that improves itself with every passing iteration.

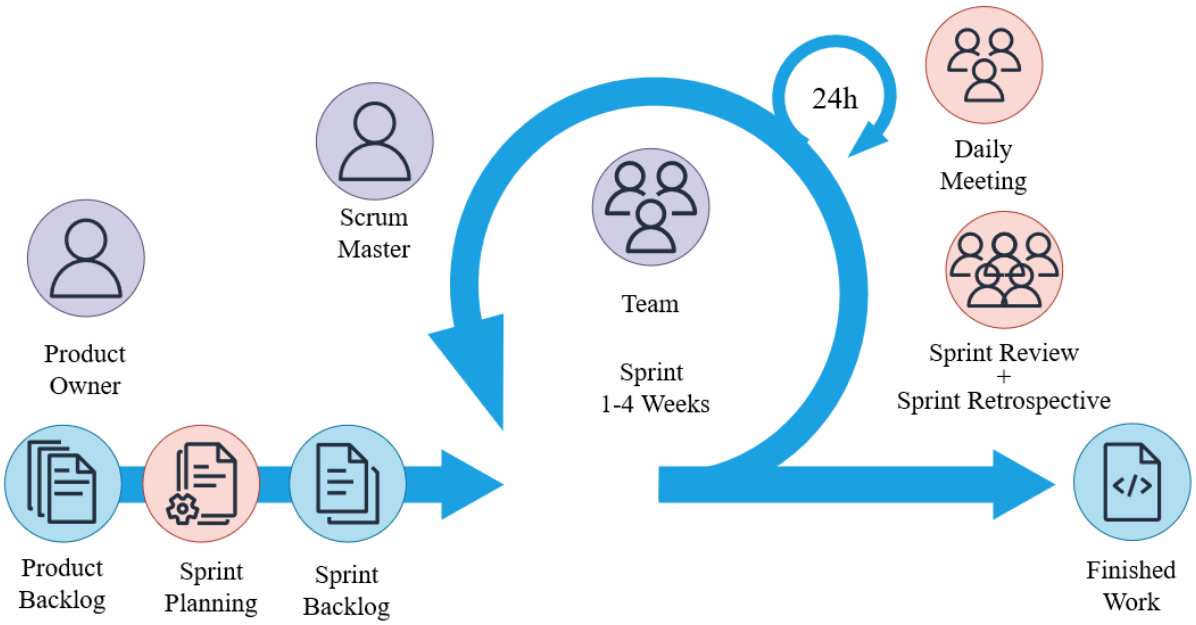

Figure 1. Structure of Scrum. Adapted from [23]

In the anecdote, the vehicle that is not able to adjust to its environment, change the route or recalibrate its priorities represents the old non-agile way of doing project management for example the waterfall method. The waterfall is a so-called linear software development management method, meaning that one step is done at a time and after that the process moves on to the next step and so on [24]. The process is defined in Figure 2. This means that by the design of waterfall, the release of the software is done after all the steps have been completed.

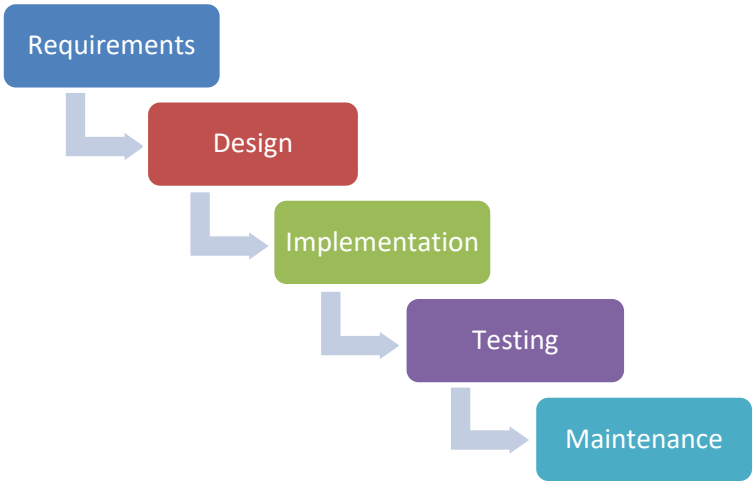

Figure 2. Flow of the waterfall method. Adapted from [24]

Petersen, et al. [25] suggested that many of the negative criticism against the waterfall model is based on beliefs and not concrete studies. These beliefs include the inadaptability described in the anecdote and the amount of rework that is caused by working in the waterfall model. However, the empirical study concluded by Petersen, et al. concluded that the criticism against the waterfall model is justified and that the standout issues were indeed with the requirements and verification. Similar conclusions were drawn in [24], where it is reported that most people prefer working in Agile, because of the increased efficiency.

2.2.2 Team Roles in Scrum

In scrum there are three different roles, Product Owner, Scrum Master, and the Team and their functions are described as follows [26]:

- **Product Owner** is responsible for managing the communication between the Team and the Stakeholders, which means that all communication should ideally go through the product

owner. The Product owner is also in charge of the Product Backlog and the prioritization of it. The prioritization of the Product Backlog i.e., the order in which the User Stories are done, makes sure that maximum value is produced for the Stakeholders.

- **Scrum Master** is the embodiment of Scrum inside the team and their job is to make certain that the Scrum Team follows the Scrum practices. In the event that the team is failing to follow the Scrum practices the Scrum Master provides teaching and guidance on how the practices should be followed. The Scrum master also functions as the facilitator in the different meetings that take place in Scrum.
- **The Team** is the engine of the Scrum Team i.e., their job is to generate value by participating in the design of the User Stories and by completing them. The Team consists of people who are participating in the development process in one way or another. This means that the Team can consist of any member the Scrum Team deems important for doing their job. This might include Developers, designers, or even business experts.

Even though by just reading the role descriptions it may sound like either the Product Owner or the Scrum Master would be functioning as the project manager that is not the case. In scrum, each team member is on the same level in terms of hierarchy and the team must together work as a self-governing organism that is able to generate value without a de facto leader. [27]

2.2.3 Scrum Artifacts

Scrum artifacts are in place to make following the Scrum Process more tangible and they are described as follows [26]:

- **Product Backlog** contains all the User Stories related to that specific product i.e., it includes all the specifications and features the product has.
- **Sprint Backlog** contains tasks that are taken from the Product Backlog for that specific sprint. The tasks are User stories broken down into smaller units.
- **Release Plan** is as the name suggests the release that is planned to take place in the future, it lists the goal of the release, the functionality it should contain in priority order, possible risks, and the date for the release.
- **Burn-down charts** are the visualisation of how much work is still left to do for the Sprint.
- **Task board** is the visualisation of the User stories and tasks for a specific Sprint.

2.2.4 Scrum Ceremonies

The Scrum ceremonies are like meetings, and they are described as follows [26]:

- **Sprint Planning** is a meeting to plan what is going to be done during the next sprint and how is that goal going to be achieved.
- **Spring Review** takes place at the end of the Sprint and its purpose is to present the new functionality to the Stakeholders that was achieved during that Sprint.
- **Sprint Retrospective** is used to improve the ways of working by critically inspecting the last sprint. The key takeaways of the meeting are to recognize what went well in the last sprint and what did not go well, and the goal is to continue doing things that went well and to stop doing this that did not go well.
- **Daily Scrum meeting** takes place daily, during which the Team explains what they have been doing after the last Daily Scrum meeting, and what they are planning to do before the next daily meeting, and they may ask for help from others.

2.2.5 Release and Deployment Management in Scrum

The release management strategy defined by the artifact Release Plan seems to be most similar to the Hybrid release management strategy explained in Chapter 2.1.4 because it has the following characteristics: a meeting is conducted between time intervals in which, the features released are listed and the release date is decided. The landscape of today's software development is said to revolve around the cost of projects and that the return on investment time needs to be short [28]. Therefore, the Stakeholders want to see results even faster than before. This is the same reason that initially caused waterfall to be replaced by the agile methodologies is now causing the agile methodologies to adapt [29]. The adaption comes in the form of introducing DevOps to the agile processes, which enables the Scrum teams to continuously deliver value to their Stakeholders.

The change from following the traditional Release Plan to continuous delivery makes sense if the demands of Scrum on the Release and Deployment Management strategy and the core value of Agile are inspected. This is because the release and deployment strategy should support the Scrum process and values as much as possible. Hence, the used release and deployment strategy should allow for a smooth and easy transition of the User Stories from the development environment to the testing environment because this is done a lot in Scrum. Additionally, it should allow the User Stories to reach the customer as soon as possible once they have been deemed done. This makes time-based release management strategies suboptimal for Scrum because it means that the User Stories cannot be released when they are actually finished, rather they may only be released on the set release dates. Therefore, based on the core principles of Scrum and Agile it could be said that the only optimal release management strategy when using an Agile methodology is to use a feature-based release management strategy, that is based on releasing features to the customer whenever they are finished.

This means that more emphasis is put on frequent releases when working with Agile methodologies than when working with older project management systems such as waterfall.

2.3 DevOps

DevOps is a word that combines “development” and “operations”, which makes it a portmanteau and so does DevOps also aim to combine these two traditionally separated silos into one seamless process. Combining these two separated entities embodies the Agile values as it increases collaboration during the development process, for it brings the operational processes closer to the Developers and the development process closer to the operational side. As a result, the team becomes more cross-functional and each team member is more aware of the whole process happening around them. [30]

As DevOps embodies the Agile values, some of the DevOps practices may also be seen in Agile. However, many DevOps practices have been reported only to be found in DevOps, “such as automating the analysis of applications, i.e. continuously monitoring the performance of the system, presenting the results in automated dashboards” [31]. The similarities between DevOps and Agile are still clearly visible as DevOps has been seen as a supporting element for Agile or as something that extends Agile. [31]

In 2013 Puppet Labs and IT Revolution Press conducted a survey [32] that was taken by more than 4000 IT professionals and the reported results in favour of using DevOps were quite clear. According to the survey, professionals from teams that use DevOps answered that they have seen a rise in software deployment quality and the amount of software releases. Additionally, “High-performing organizations” [32] are reported to ship code thirty times faster and those deployments are also 8000 times faster than the ones from their peers. Those organizations also have fifty per cent fewer failures, and they recover from failures twelve times as fast as their peers do. An organization is reported to be more likely a performing organization if they have implemented DevOps practices. Moreover, DevOps maturity is said to have a positive effect on the performance of the organization.

2.3.1 DevOps Practices

The goal of DevOps is to speed up the delivery of the software, while still keeping the code quality and system integrity high. This is achieved by shortening the time it takes for changes done to the software to reach the production environment all the while integrating mutually reinforcing testing

practices [33]. To ensure that these goals are achieved when using DevOps, Bass, et al. [33] suggested the following practices:

1. The operations side of DevOps ought to be treated with the utmost importance when it comes to their requirements. This is due to there being a myriad of different requirements for logging and monitoring. An example of such a requirement is that the logging messages should be clear and usable by the persons who are affected by them. Therefore, it is crucial that the operations are involved when developing the requirements for them.
2. The development side of DevOps should be more involved in incident handling. The purpose of this is to speed up the process of error solving by decreasing the time between error discovery and the repairment of it.
3. The deployment process must be enforced by all including both Dev and Ops personnel. If these practices are followed correctly the quality of the deployments is higher because there are fewer misconfigurations and errors in ad-hoc deployments. Additionally, these practices entail the time it takes to observe and solve an error as well. Furthermore, the deployment process should support the process of keeping track of the history of deployed artifacts and which components are included in the artifacts.
4. Continuous deployment is to be used, in order to deploy software changes quicker and to have adequate automated test coverage to ensure the quality of the changed software.
5. Infrastructure code should be developed with the same practices as application code is developed. Therefore, infrastructure code like application scripts is to be developed with the same quality control practices that would be used in normal development processes. Low-quality infrastructure code may result in errors in the deployment process and the application and its environment. Thus, by using proper quality control practices these errors and misconfigurations in the infrastructure code may be rectified before they can cause problems.

2.3.2 DevOps Cycle

The DevOps cycle consists of eight steps of which the development part contains four of them and the operational part consists of four. The development part entails the following steps: Plan, Code, Build and Test. While the operations part has the following steps: Release, Deploy, Operate and Monitor. The loop is visualised in Figure 3. The DevOps process is an infinite loop that starts with the development part and ends with the operations part. Albeit, since it is an infinite loop, it means that the different steps often happen simultaneously. For instance, a Developer may have already started coding something new while the Developer's previous work is currently being deployed and while the Developer is monitoring their old work. [34]

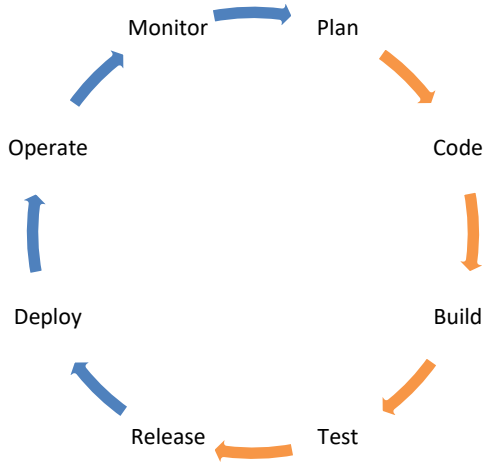

Figure 3. The DevOps cycle. Adapted from [34]

2.3.3 DevOps Tools

DevOps offers a lot of features and procedures, and to execute these, support may be needed in terms of tools. Although every single aspect of DevOps does not need tools to actualise, they can still provide valuable help and they may also make it easier to follow DevOps. Katal Avita, et al. listed [35] nine different categories in which tools may be applied to help, which are as follows:

1. *Collaboration tools* are important because DevOps requires communication with multiple different parties. Communication can help in sharing knowledge, avoiding misunderstandings and in general making the parties more involved in the flow of information. Therefore, the tools in this category are said to offer real-time chats, groups for discussion and repositories for storing knowledge. Example collaboration tools include Skype for the communication aspects and GitHub Wiki for information storage.
2. *Planning tools* aim to make the workflow in DevOps more transparent and communicative while making it easier for teams to be engaged in planning as well. Planning tools include such tools as the Kanban Board and JIRA. Both of these examples visualize the progress made and what is being worked on transparently.
3. *Issue tracking tools* enable the team to respond to issues quickly and they are also used for storing, sorting, and sharing information about issues. Issues may come from customers or from inside the team or organisation. For customer-related issues, a tool called Zendesk is listed as an example. For issues found inside the organisation, a JIRA-like tool may be used to track the issue and the progress of the issues.
4. *Monitoring tools* are valuable for a team using DevOps because the releases are more commonplace, and the systems are advised to be more closely monitored after releases. After a release, a monitoring tool can prove helpful in diagnosing if the system is running without any

problems. For instance, if the activity is lower than usual or if there are more errors than normal in the logs, it may be a tell-tale sign that something is not right. For monitoring systems, Microsoft System Center is used as an example and for logging Logstash is used.

5. *Configuration Management tools* are used to combat configuration drift by making the changes to the configuration file easily deployable systematically and automatically instead of manually. Configuration drift describes a situation in which the configuration slowly changes to be inconsistent with its or the organization's requirements. Systematically and automatically updating the configuration file can be used to track who made the changes and it makes it easier to update multiple servers at once to all the right places. Examples of configuration management tools are Chef and Puppet.
6. *Source Control Tools* may be used to keep track of changes in the codebase while controlling who may commit changes. Therefore, it is known what, when and by whom something was changed. This may be useful in an event where something breaks after a change has been made, because then the previous version can be restored, and it is easy to see the changes that broke the application. GitHub has been said to be the most common source control tool available.
7. *Development Environment tools* are used to create an environment for the Developer that mimics the production environment. To create an environment like this configuration management tools can be used to make sure that the configurations are the same in the development environment as in the production environment. Using tools like this can solve the issue where the made changes only work locally on one computer.
8. *Continuous integration tools* are capable of building, automated code-inspections, automated testing and of automated security check-ups. The goal of these tools is to make sure that the changed software can be built and that it is up to standard by inspecting and testing it autonomously. Examples of continuous integration tools are TeamCity and Jenkins.
9. *Deployment tools* are in charge of reliably deploying the newest software package to the different environments autonomously. The deployment tools rely on the other tools to ensure that the package is good to go. Examples of tools in this category are Docker and Octopus.

2.3.4 Current State of DevOps

The 2021 State of DevOps [7] report by Puppet Labs highlighted that most DevOps teams that have implemented DevOps practices are stuck in the middle level of DevOps evolution, which causes the teams not to be able to achieve the full potential of DevOps. Being stuck in the middle has been reported to have varying effects depending on whether the organisation is in the top part of the middle or the bottom part. The impediments reported by the high-middle teams are inadequate feedback loops, vague responsibilities, and failures in sharing best practices. For the lower part of the middle, there are skills shortages, legacy architecture, organisational resistance to the DevOps changes, and

insufficient amounts of automation. Of these mid-level organisations only three per cent report that the executive level resists the changes needed to enable DevOps.

To remedy these issues the report suggests that the leadership in these teams should actively promote DevOps and its values. This is to be done to dispel the illusion that only talking about DevOps culture would be enough to foster a thriving DevOps environment. The report highlights that to reach the highest evolution of DevOps both bottom-up and top-down initiatives are needed, these help the organisation in creating patterns and knowledge-sharing practices through which the team is enabled to optimize and adapt quicker.

2.3.5 DevOps in Relation to Release and Deployment Management

As mentioned before DevOps aims to integrate the development and operational parts of creating software, which often before have been thought of as separate silos [30]. DevOps itself can be thought of as a Release and Deployment Management framework, for its operations part incorporates all the parts software cycle listed by Dearle [9]. Additionally, the requirements for release management listed by Van der Hoek, et al. [10] can also be found in somewhat different words in the Bass' book on DevOps [33]. It has also been stated that DevOps is the answer to the IT industry's need to have a more frequent and stable release management strategy [36] [37]. Thus, it could be argued that Release and Deployment Management have been made more visible through the rise of DevOps. Even though, it could be said that the Release and Deployment Management strategy provided by DevOps is based on CI/CD, some have stated that DevOps is vital if an organization wants to enable Continuous Delivery [38]. Nevertheless, in the next Chapter, the meaning of CI/CD is explored.

2.4 Continuous Integration / Continuous Deployment

CI/CD combines the two terms: Continuous Integration (CI) and Continuous Deployment (CD), but Continuous Delivery (CDE) is often added to the mix. The term CI is said [3] to have reached its popularity thanks to Kent Beck's work [39] on extreme programming (XP), and the term CDE has been coined by Jez Humble & David Farley [40]. The origins of the term CD are a bit more unclear, but it may have originated from Timothy Fitz's blog [41]. As mentioned in Chapter 2.1.4 CI/CD is a feature-based Release Management Strategy.

2.4.1 Continuous Integration

Continuous Integration means that the building and testing of the software are done automatically [42], which aims to make the process of developing and deploying the software more seamless [43]. By having a more seamless process companies can have more frequent releases with improved software quality while increasing the team's productivity [43].
Setting up a CI process may not be as easy as it sounds, because the process requires a lot of automation and tools to realize [44]. The goal of automation is to lower the amount of manual work needed, thus it potentially makes the process of building and testing the software easier and faster. The functions of CI tools and example tools of that category are covered in Chapter 2.3.3.

Continuous Integration has been reported as being a net positive to software development, but at the same time, it does have some room for improvement. CI has been stated to do well in improving cooperation but it also increases the amount of process and technical challenges. [45]

2.4.2 Continuous Delivery

At its best Continuous Delivery means that the new version of the software may be released with just a press of a button [46]. The process should result in quick and safe deployments, while the process itself should be sustainable [40]. CDE is also described as something that gives the team the power to deploy at will, thus the team can deploy at any time with any frequency [47]. Steve Neely & Steve Stolt [47] argue that deploying at will does not make the code any less safe than if there are longer periods between deployments. Their argument is based on CDE’s ability to have large test coverages, quick feedback cycles, monitoring systems, and swift rollback mechanisms. All these features are said to make the deploying process safer, but they also make it clear that it comes with a cost.
The cost is that implementing a CDE process takes resources and time, and it may be made more difficult if the systems and solutions are considered legacy.

In Continuous Delivery the code does not necessarily have to reach the end customer, the deployments may be done to any environment it does mean that the deployment would be done to the production environment [40], [47]. In order to have a CDE process it is indicated that a Continuous Integration process should be in place already [48]. As CDE is about deploying as continuous delivery, tools that can be used for CDE can also be used for CD. CD tool functionality and example tools are described in Chapter 2.3.3.

2.4.3 Continuous Deployment

Tony Savor, et al. [49] describes Continuous Deployment as a process in which there are smaller iterative software deployments into a production environment as a continuous stream. This means that as soon as a piece of software has passed automated validation steps, it is autonomously deployed into a production environment. CD has been widely accepted by some of the biggest internet companies including Facebook, Flickr, Etsy, Amazon, Google and Netflix. In the same paper, Tony Savor, et al. [49] list the key features of a Continuous Deployment process as the following:

1. The changes done to the software are as small and isolated as they can be.
2. After the software changes have been developed and tested, they are released right away for deployment.
3. The Developers are mostly in charge of making the decision when to deploy.
4. The process of deploying is completely automated.

There is academic debate on the differences between Continuous Delivery and Continuous Deployment. Sometimes the difference is thought to be that in CD the software is deployed to the production environment, and it is done completely automatically [48]. Thus, in CDE the process leading up to the release of the software is automatic, but the release is not. In contrast in CD, the whole process is automated [50]. Additionally, Jez Humble [51] emphasises that the key difference between CD and CDE is that in CD every good build is released to the customers, which is why Humble would also prefer the name ‘Continuous Release’. Humble also states that the CDE process is included in CD, but the CD process has parts that the CDE process does not have. Additionally, the CD process has been described as being a push-based solution and the CDE a pull-based one [48]. All in all, it could be said that the CD process is the final evolution of continuous software development.

2.4.4 CI/CD Pipeline

The Continuous Integration and Continuous Delivery pipeline is in charge of automating the previously mentioned processes, this includes building, testing and deploying the software [52] [53]. The CI/CD pipeline has been studied to increase productivity and to enable faster releases when it has been used in an Agile environment [54] [6]. Additionally, a CI/CD has been said to enable value to be delivered to customers more often, and that the automation reduces the team’s workload [55]. Another reported benefit is that it removes the dread of having to deploy multiple features at once, something Neely and Stolt called the ‘Scary Saturdays’ [47]. This has been said to make delivering software easier since if only one feature is deployed at once, a potential production bug is easier to locate and solve. In comparison, if ten features are deployed at once and something goes wrong the amount of

code that needs to be inspected is going to be a lot more. Additionally, in a ‘Scary Saturday’ system if one of the ten features is faulty that will possibly mean that none of the ten features can be released, and the release may have to be postponed. However, this problem does not exist when using a CI/CD pipeline because releases can be made at will, thus a faulty feature does not prevent the releasement of other features.

2.5 Content Management Systems

Content Management Systems (CMS) are systems that automate tasks for content creation. A CMS enables a Content Manager to create and edit content for end users to consume, deletion of content is allowed as well. A CMS is composed of many different parts, which typically include an editorial interface, a repository for the data, and a mechanism for publishing. There are multiple types of Content Management Systems, but this thesis focuses on Web Content Management Systems. Some examples of Web Content Management Systems are WordPress and EpiServer. [56]

A CMS allows the Developers of the website to develop reusable blocks that a Content Manager may use to design the website [57]. If a Content Management System is not used it means that the content on the website is static and changes to the website require code changes that need to be released. Therefore, a website that is powered by a CMS is more fluid by its nature, because when using a CMS, the content of the website can be modified without changing the underlying code.

2.5.1 Content Management Systems in Release and Deployment Management

Content Management Systems allow the creation and editing of blocks on the website, which makes the website more reusable, but it also means that the content must be set after the code for the website has already been made. Thus, a webpage that relies on CMS blocks is incomplete before a Content Manager has added the content to the page. This means that in Release and Deployment Management for software that uses CMS, there must be an additional step in which the Content Managers set the Content into the software before it is shown to the end users, or there is a risk that the software is released incomplete to the end user.

In a Release and Deployment Management system that is not fully automated, the content can be set as the last step before the software is released. The same approach may not be as efficient in a fully automated Release and Deployment Management system, because it may not be certain that the Content Manager is able to set the content before it already reaches the production environment.

The solution to the problem that arises from managing a software product that uses a content management system may be solved using feature toggles. A feature toggle is essentially an on/off switch for a feature that allows the new features to be turned on and off [58]. Consequently, this enables a fully automated Release and Deployment Management process for features that are reliant on content that a Content Manager needs to set. Therefore, it could be argued that an automated Release and Deployment Management process needs feature toggles to combat the lost control over features reaching the production environment automatically. This may also be the reason why big software companies like Google, Facebook and Netflix are reported [58] to use feature toggles as part of their CICD deployment processes. Although feature toggles allow more flexibility when releasing new features, it has also been reported to increase the amount of technical debt and maintenance needed [58].

2.6 Release and Deployment Management Attributes

The purpose of this section is to condense the attributes that were found to be associated with Release and Deployment Management in the literature review. The found attributes are summarized in Table 1 together with their sources. The attributes in the table are used in the interviews and in the surveys in two different ways. In the interviews the attributes are compared to the opinions of the interviewees. In the surveys the attributes are used to create a scoring system that can be used to rate the Release and Deployment Management systems. This also serves as a way to verify whether the terms are relevant to Release and Deployment Management by measuring the results.

Table 1. Attributes attached to Release and Deployment Management and their sources.

Attribute	Source
Quality	One of the goals of DevOps is to increase the quality of the end product [33].
Less manual work through automation	Automation of CI/CD reduces the workload of the team [55]
Trackability	Tracking of changes is a requirement for release management [10].
Traceability	Keeping historical records is a requirement for release management [10]
Flexibility	CI/CD can be used to deploy at will [55]
Quickness	One of the goals of DevOps is to speed up delivery [33].
Smooth flow of work	Agile methodologies require a smooth flow of work in order to continuously deliver value [29].
Transparency	Planning tools are used in DevOps to promote transparency [35].

Replicability	Configuration Management tools are used in DevOps to make sure that the deployment process can be done with the same configurations across multiple servers [35].
Failure recovery	Mature DevOps organizations recover faster from failures [32].
Productivity	Seamless processes increase productivity [42]
Predictability	A time-based Release Management Strategy increases the predictability of releases in open-source projects [17].
Seamlessness	Continuous Integration aims to make the development process more seamless [42]
Error tracking	Issue-tracking tools are used in DevOps to keep track of errors [35].
System integrity	The main goal of Release Management is to have high system integrity [13]
Safety and stableness	Continuous deployments are made safe by having large test coverages, quick feedback cycles, monitoring systems and swift rollback mechanisms [47]

3 Migration Case Study Description

The thesis uses a migration case study to conduct research on Release and Deployment Management. Therefore, the thesis revolves around the case study, which provides a practical foundation on which the empirical inquiry is carried out. This chapter's objective is to explain the context, goal and approach of the case study.

3.1 Case Study Context

The case study is conducted in If Vahinkovakuutus Oyj, which is a Nordic insurance company. The project in question is a public-facing website that is used in Finland, Sweden, Denmark and Norway. There are in total three teams working with the solution out of which two are working with a single country and one is working with two. Even though, the website is used in four countries they share a common solution, code repository and a Release and Deployment Management process. Additionally, the teams share a Sprint Review but other Scrum Ceremonies are done on a team basis. Furthermore, the teams have their own Scrum Artifacts, but there is also an additional common Backlog for common technical improvements and bugs.

Each team consists of a Scrum Master, Product Owner, UX Designer, Lead Developer, Quality Engineers, Developers and Business Specialists. In addition, the teams share an Architect. There are 39 people working in the teams. This includes three Product Owners, three Scrum Masters, three UX Designers, three Lead Developers, four Quality Engineers, 14 Developers, 8 Business Specialists and one Architect. The pre-migration Release and Deployment Management tools have been in use since 2016. There are only a few people in the teams who have been working on the solution when the pre-migration process was adopted. As the teams share a common process it requires coordination between the teams. One of the Scrum Masters is also working as a Release Manager.

3.2 Case Study Goals

The goals of the case study are explained in more detail in Chapter 1 and they are briefly summarized in this section. The main goal of the case study from a practical point of view is to migrate the Release and Deployment Management tools to Azure DevOps. The reason for this is that everything else used by the teams is already in the Microsoft ecosystem. Another goal is to remove the necessity of maintenance pages for the duration of production releases. This could give more flexibility for the releases and make the customer experience better by increasing the availability of the website. Additional goals related to the migration are presented in Chapter 4.3.

The goal of the migration from an academic point of view is to offer a practical foundation on which the empirical inquiry can be conducted. The empirical inquiry is carried out by using four research questions, which can be found in Chapter 1. The chapter gives more background regarding the academic goals. In short, the academic interests are related to three different viewpoints: (1) Release and Deployment Management in relation to software development, (2) The relation between Scrum and Release and Deployment Managemnt tools, (3) Automation that can be achieved with CI/CD solutions.

3.3 Case Study Approach

The case study is divided into four different parts. Firstly, the pre-migration chapter represents the case study's case description, which means that it is used to explain the situation before the migration. Additionally, it is about the reasons behind the migration and it contributes to the research by explaining the background for the study. Secondly, the migration chapter is the implementation chapter for the case study. The implementation chapter is about the process of building the new Release and Deployment Management tools. The tools are built by using the knowledge from the literature review. The chapter is mostly practical, but it also provides contextual insight into the topic and it helps to understand the results of the study. Thirdly, the post-migration chapter is for the case results, which means that it is about the post-migration process and it compares the tools and processes to the pre-migration situation. It contributes to the research by giving details on the differences between the pre-and post-migration systems. The differences are used to interpret the collected data from the interviews and surveys. In addition, the chapter condenses the information from the practical aspects of the case study, which increases readability and verifies the case study's results empirically. Fourthly, data collection for the case study is done by conducting interviews and surveys. The data collection's main objective is to use the previous three parts as a foundation to gather data. The gathered data is analyzed to verify the theories found in the literature review and to asses the practical findings.

4 Pre-migration Release and Deployment Management Process

This Chapter serves as the case description for the case study. This makes it a starting point for the case study part of the thesis. The chapter's purpose is to present the pre-migration situation and the reasons for it. Therefore, the chapter starts by going through the pre-migration tools and after that the pre-migration process. Lastly, the chapter describes the reasons and ambitions for the migration.

4.1 Pre-migration Tools

In order to understand the process, the tools must be presented first because they form the backbone of the process. Presenting the tools is important because it helps in constructing a holistic image of the whole Release and Deployment Management process, as the process is built with these tools. Presenting the tools helps in understanding what the interviewees and respondents refer to with their answers. The tools are categorized in the same manner as in the paper by Katal Avita, et al. [35].

1. Collaboration tools, for real-time chatting and communicating with other team members Microsoft Teams is used and for storing knowledge Azure DevOps is used.
2. Planning tools, the team uses Azure DevOps for planning
3. Issue Tracking tools, Issues are tracked by using an in-house ticket system and with Azure DevOps.
4. Monitoring tools, the production environment's availability and functionality are monitored by using Apica, which runs automated tests on the production environment and verifies that everything can be accessed. Splunk is used for logging errors in the software code. For monitoring the health of the application Azure Monitor is used
5. Configuration Management tools, configurations are managed by using Azure Resource Manager templates.
6. *Source Control Tools,* for source control a Git repository is used together with Azure DevOps.
7. *Development Environment tools,* the Developers use a localhost environment that mimics the production environment. Additionally, there are three different testing environments: one for integration testing, one for system testing and one for acceptance testing. The integration testing environment is not used except for testing new environment configurations that may break the System Testing environment. The reasoning for this is to have an environment, which is hosted and accessed similarly to the other environments, but since it's only used for integration testing, System Testing will not be interrupted if the environment breaks.
8. *Continuous integration tools,* for CI the team uses TeamCity.
9. *Deployment tools,* Deployments are done using Octopus Deploy.

4.1.1 TeamCity

As mentioned in the list above the teams are using TeamCity as their Continuous Integration tool. TeamCity listens to commits on a branch and after a commit is made TeamCity automatically builds the project and runs the automated test coverage on the updated software. The tests can be run in parallel, which allows for faster test results. After the software has been built successfully TeamCity creates a build artifact out of the software. TeamCity allows running tests on other branches than just what it is listening to this is useful in the cases where there is a need to run a lot of tests before committing to the branch which is shared between all the Developers. [59]

The TeamCity user interface offers an overview of the test runs and their statuses as can be seen in Figure 4. The view is offered for all branches of the project.

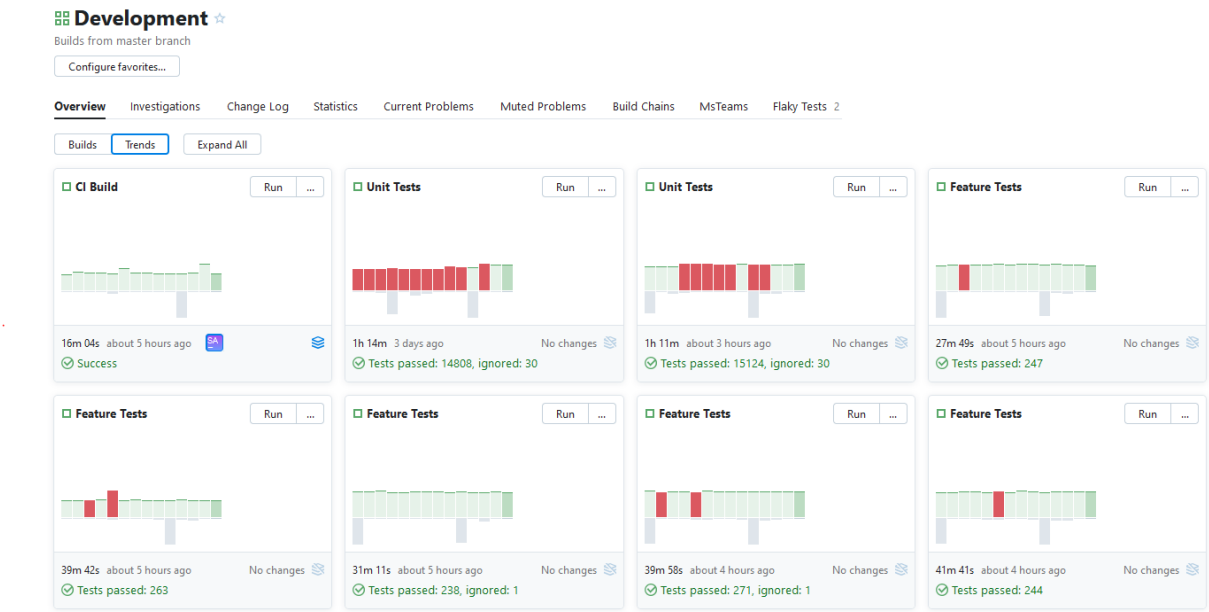

Figure 4. TeamCity User Interface.

4.1.2 Octopus Deploy

Octopus Deploy is linked together with TeamCity so that it can fetch the created build artifacts from it. Deploying to environments is simply done by selecting the release and pressing the deploy button. Deploying can also be done automatically, for instance in the pre-migration setup if all automatic tests pass TeamCity tells Octopus Deploy to automatically deploy to the System testing environment and the Acceptance testing environment. Octopus Deploy allows the deployment of ARM templates as well. [60]

Octopus offers a user interface that shows all projects divided into project groups as can be seen in Figure 5.

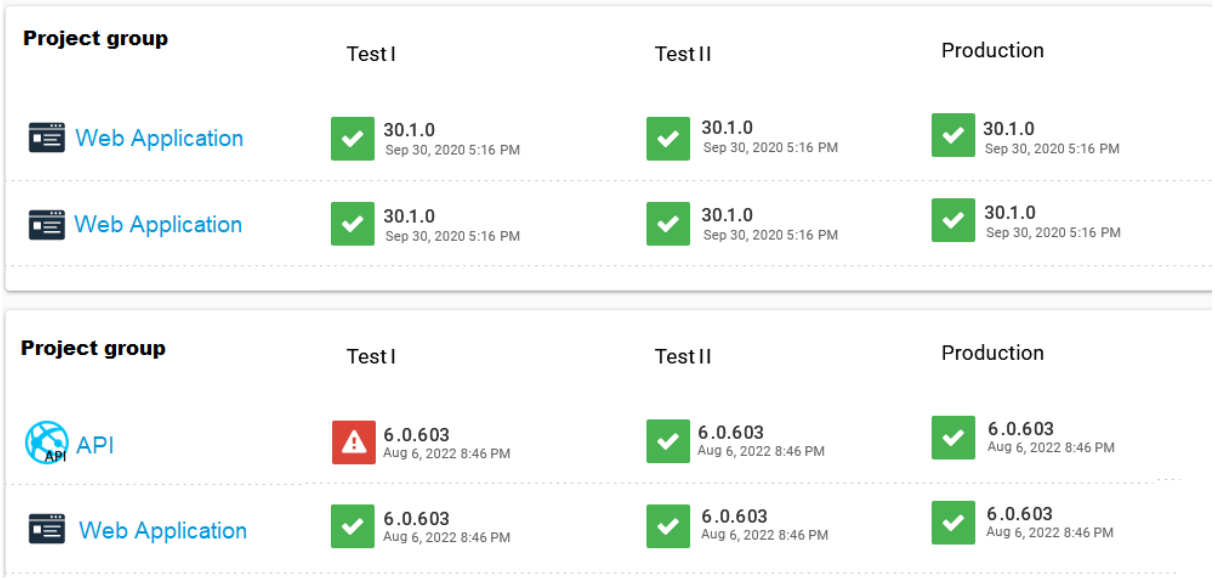

Figure 5. Octopus Deploy User Interface.

4.2 Pre-migration Process

The pre-migration Release and Deployment Management process could be divided into four parts: The first part starts when a Developer creates a Pull Request to commit new changes on the master branch. This moves the previously local changes to the System Testing environment, which is based on the master branch. After the changes have been committed on the branch the Continuous Integration process is triggered in TeamCity. First TeamCity builds the newest code, then it runs an automated test coverage against the code after which if none of the tests fail TeamCity triggers a deployment to the System Testing environment with Octopus Deploy. In the event that some tests fail the newest version of the code can be manually deployed with Octopus Deploy as well. This may be the case when for instance a flaky test failed. After the changes have reached the System Testing environment the Developer notifies the testers by marking the story as ‘ready for Sprint Testing’ in Azure DevOps.

In the second part, the testers are testing the new code in the System Testing environment. After the changes have been tested and accepted, the tester marks the story as ‘Done’ in Azure DevOps. The deployment of the story to the acceptance testing environment is done only after the latest deployment to production has been deemed stable and after all the Developers have confirmed that their stories that are in the System Testing environment are good to go for the next release. The deployment of the stories from the System Testing environment to the Acceptance Testing environment is done by

merging the master branch with the release branch, which is done by any of the Developers. Commits into the release branch trigger a similar process in TeamCity and Octopus Deploy that happens for the master branch except the destination of the changes is the Acceptance Testing environment.

The third part consists of acceptance testing done by the team's Business Specialists, essentially by the persons who designed the story for the changes in the first place. After the story has been acceptance tested, the Business Specialist changes the story's state to 'Accepted'. To track which stories are going to be in the next release, a release query is used, which uses a date that can be set for a specific task in Azure DevOps. The Developer who developed the story is supposed to check what stories went into the release and mark the release date for those stories.

The fourth part of the process is the deployment to the production environment. The deployments take place after every Sprint and as the Sprint length is three weeks, releases are done in three-week intervals. Before the release is done all the teams confirm that the changes, they currently have in the Acceptance testing environment can be released. The release itself takes place at 7 a.m. Eastern European Time, which means that for the other Nordic countries, it starts at 6 a.m. The release starts with the Developer turning on maintenance pages for the website after which the Developer promotes the latest Acceptance Testing version to production. Then the Developer goes into the CMS to content lift the latest Acceptance Testing content to the production environment. After the latest Acceptance Testing version and the content of it has reached the production environment the Developer notifies the Business Specialists that they can start verifying the content and the main flows of the website using the internal user version of the website. After the verification has been completed the Developer turns off the maintenance pages. Then after the main part of the deployment, the Developer monitors the environment using Splunk to see that everything is going smoothly. Additionally, the Developer is on standby in case there are Apica alerts, and the Developer is ready to roll back or react to possible problems however required. The whole process is visualized in Figure 6.

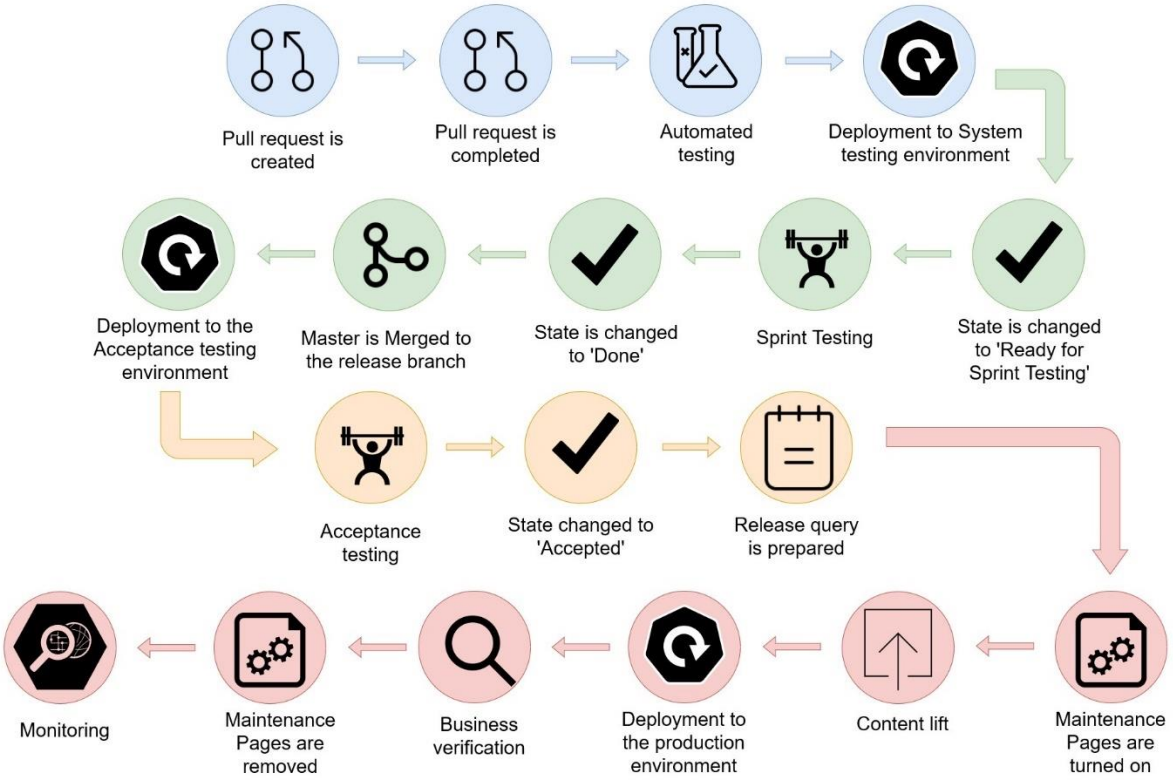

Figure 6. Pre-migration Release and Deployment Management process.

4.3 Reasons for Migration

As described the pre-migration process is a time-based Release and Deployment Management strategy and therefore it encompasses all the problems that are accompanied by a time-based strategy. Such problems are features are not delivered to the customer when they are ready, Sprint-based releases make sense on paper, but stories are not always ready in the sprint they were planned, so Sprint's released content is not actually that Sprint's content. Additionally, as the releases are done in the same manner as Scary Saturdays [47], it makes the releases seem more dreadful than they could be. Doing a release with this system requires the effort of multiple people in the early hours of the morning and if the release goes wrong somehow all of the effort was done in vain. As there are more than ten Developers working on the solution and only one of them is doing the release at any given time this means that if something goes wrong the Developer in charge of the release is most likely not the person whose changes caused the issue. Therefore, it could be more beneficial if the Developer whose changes are released would be present and the Developer could also be in charge of the release as well. However, this is often not the case, since the releases happen so early that usually it happens before anyone else is online.

Some additional reasons why a change to Azure DevOps pipelines would be beneficial is that the webpage consists of multiple solutions, which could all have their own pipelines instead of depending on the one TeamCity Octopus Deploy combination. This would enable us to deploy features to the smaller components much quicker because that would mean that parts that were not affected would not have to be built and tested as well. Another big reason is that the teams are already using Azure DevOps for everything else and since the applications are hosted in Azure it makes sense to have everything in the same place. This would essentially cut down costs and there would be no need to go around three different websites to find out what is going on. Additionally, there are agents that everyone can use in the company so this would mean that we would not have to manage and configure our own agents anymore, this would also lower the costs.

To sum it all up the goal is that deployments could be done without turning on the maintenance pages because this would enable us to deploy more frequently and whenever, thus, a time-based model would not have to be used anymore. The maintenance pages tie the deployments to a time-based model because the website should be available to the customers as much as possible. This means that deployments need to be done during times when there is not much traffic, which is during the early hours of the day. The maintenance page free deployments could be achieved using the pre-migration tools, but since we are already using Microsoft services for everything else, it makes sense to unify the services completely. More reasons for and against the migration will be offered in the interview and survey results.

5 Migration of Release and Deployment Management Tools

This chapter covers the technical implementation of the migration and it is the implementation chapter of the case study. The chapter starts by introducing the Continuous Integration pipeline and then continues to the Continuous Deployment pipeline. After that, the User Interface testing is briefly covered and the chapter ends by presenting problems that arose during the migration. The end goal of the migration is to move the CI/CD tools to Azure DevOps pipelines and to enable deployments to be done without turning on maintenance pages for the customers.

5.1 Description of the Process

The process is divided into three distinct steps, building the projects, running tests and deploying the projects. The first two pieces are both happening in the CI pipeline and the last one is in the CD pipeline.

5.1.1 Continuous Integration Pipeline

The process starts by creating a Continuous Integration Pipeline for the projects. The CI pipeline's function is to automatically detect changes and then build and test the code. CI pipeline in Azure DevOps is in three pieces compared to the one in TeamCity. By splitting the pipeline into smaller pieces only the affected parts need to be rebuilt and tested and for the unaffected ones, the old build artifacts can be used.

The first CI pipeline that is created is for the API project, and before it can be done the dependencies between the API project and the User Interface (UI) project must be severed so that when the API project is built the parts of the UI project must not be built. Severing dependencies can take a lot of time depending on how tightly the two are intertwined and most likely the code needs to be restructured to get rid of the excess dependencies. After the dependencies are severed from the projects and their test projects the building of the pipeline may commence.

After the project has been built the testing will take place. The testing is done in parallel, and it is done to a greater degree than is done with the TeamCity setup, this is because there are more agents available to be used. The unit tests are going to be sliced into smaller pieces using a slicing script and the different test slices are executed by different agents. In the TeamCity setup, the unit tests would

take more than an hour to run, but by slicing the tests it takes 15 minutes or less to execute the same number of tests.

The deployable build artifact is always published, given that the project can be built. This is to ensure that the artifact could still be deployed to the test environment, even though tests might fail. This is because the failed tests might be unrelated to the made changes or they might be flaky. A flaky test means that the test results are inconsistent between different test runs. Even though the described process above is for the API project, the same steps are taken for the UI project. After the UI project CI pipeline has been made, the similarities between the API and UI CI pipelines become apparent and a common CI pipeline template can be made for both of the solutions. This means that in the template the only things that need to be specified are what projects are going to be built and what are the test projects. The pipeline is made even more clean by introducing parameter files that contain all the project-specific information.

The CI pipeline template has the following steps:

1. The build job starts with restoring Nugets for the projects, if some Nugets need some extra authentication Nuget authentication step must be run first.
2. The project and the accompanied test projects are built.
3. The project and the accompanied test projects are published in the locations in which they are going to be uploaded as artifacts.
4. The test slicing script is copied to a location from which it can be uploaded as an artifact.
5. The project, the test projects and the test slicing script are all published as their own artifacts. This ends the completion of the build job.
6. The test job starts on as many agents as is specified after the build job has been completed. The first step is to download the test projects' artifact and the test slicing script artifact.
7. The test project artifact is unzipped in a manner that keeps the file structure the same for the projects. This means that every test project is unzipped into its own subfolder.
8. The paths to the test projects are assigned to a variable.
9. The test slicing script is run on the test project paths and the filter is assigned to a variable.
10. The tests are run on the filter and the test results are automatically published after.

There is also a third CI pipeline that is for the project infrastructure and its job is to create an artifact that contains the ARM templates and Bicep files that are used to create the required Azure infrastructure to run the application

5.1.2 Continuous Delivery/Deployment Pipeline

The implemented pipeline for deploying the software is not used for automatically deploying every good build into production, which means that it is a CDE pipeline. However, the pipeline itself has the capability to be used as a CD pipeline and the pipeline is referred to as a CD pipeline by the teams. Therefore, for simplicity's sake, the pipeline is going to be referred to hereafter as a CD pipeline. After the CI pipelines it is the CD pipeline's turn, which is triggered by running one of the CI pipelines. Therefore, again the CD pipeline only needs to get the new artifact from the projects that have changed. For example, if there are only UI changes the CD pipeline uses the previously created artifacts for the API and the infrastructure. The CD pipeline is a multistage pipeline [61], meaning that the same pipeline is used to deploy to different environments and the environments themselves are specified as parameters. The environments are specified in Azure DevOps as well, which means that each environment can have its own approvals and checks, Therefore, different people can control what can be deployed and where. To make the pipeline tidier, an if statement is used together with parameters files. The if statement has all the possible environments in it, and it will select the correct parameter file based on the environment it is going to deployed to. The parameter file has information like application names and other information needed to deploy the projects.

After selecting the corrected parameter file based on the if statement the CD pipeline deploys the infrastructure first and after that is completed it deploys the API and the UI projects. The reason for this is that the applications depend on the infrastructure to work and therefore they need to wait before that is deployed. The infrastructure deployment only deploys and modifies the things that have been modified in the ARM and Bicep templates.

The deployment of the applications is done by using a common application deployment template. The first step in the deployment is to get the environment-dependent transformation of the web configurations. This is done so that the System testing version of the application uses the System testing specific endpoints etc. After that, the application is deployed to an Azure staging slot, which allows the application to be warmed up or even tested in the staging slot that is not used by actual customers. After the deployment to the staging slot, a slot swap is done, this means that the production slot version of the application goes to the staging slot and vice-versa. Thus, if the new version starts to malfunction the old working version can be swapped back in a matter of minutes. The staging slots are turned on and off during the deployment process, which saves money as the slots are not running when they are not in use.

The possibility to utilize the staging slot in testing and verification as mentioned before required some changes. This is done by overwriting the application's app configurations in Azure by using

Application Service configurations in the staging slots. Before this change the staging slot and production slot applications had the same connection strings to resources, meaning that the staging slot application used production slot APIs etc. The Application Service configurations are set as slot-specific configurations to ensure that the configurations are not transferred over to the production slot when the swap is done. This change enables the verification and testing of the newest build in the staging slot before the swap is done, thus the integrity of the production slot is better protected as the faulty build should be caught in the staging slot before the swap is done.

To summarize the CD pipeline, it has the following steps:

1. The pipeline starts by downloading the infrastructure artifacts.
2. The bicep files are verified and then deployed.
3. Artifact for deployable Application Service is downloaded.
4. Configuration transformations are done for the Application Service software package.
5. The Staging slot is turned on.
6. The software package is deployed to the staging slot.
7. The application settings are updated on the staging slot.
8. The staging slot is swapped with the production slot.
9. The staging slot is turned off.

Steps 4-7 are run in parallel for all the Application Services that are deployed, the same applies for steps 8-9, which are for swapping the slots. To further improve modularity when the pipeline is being run it will ask for what steps are to be run and in which environments. The possible jobs include “deploy infrastructure”, “deploy applications” and “swap applications” and these jobs are all available for all the environments. Therefore, it can be controlled using the same pipeline into which environments the code is deployed and whether the infrastructure is deployed, the applications are deployed or whether the swaps are done. This is especially useful as it allows deployments without always deploying the infrastructure as it takes the longest to deploy, but it has the fewest changes. The whole process including the CI pipelines and the CD pipeline is illustrated in Figure 7.

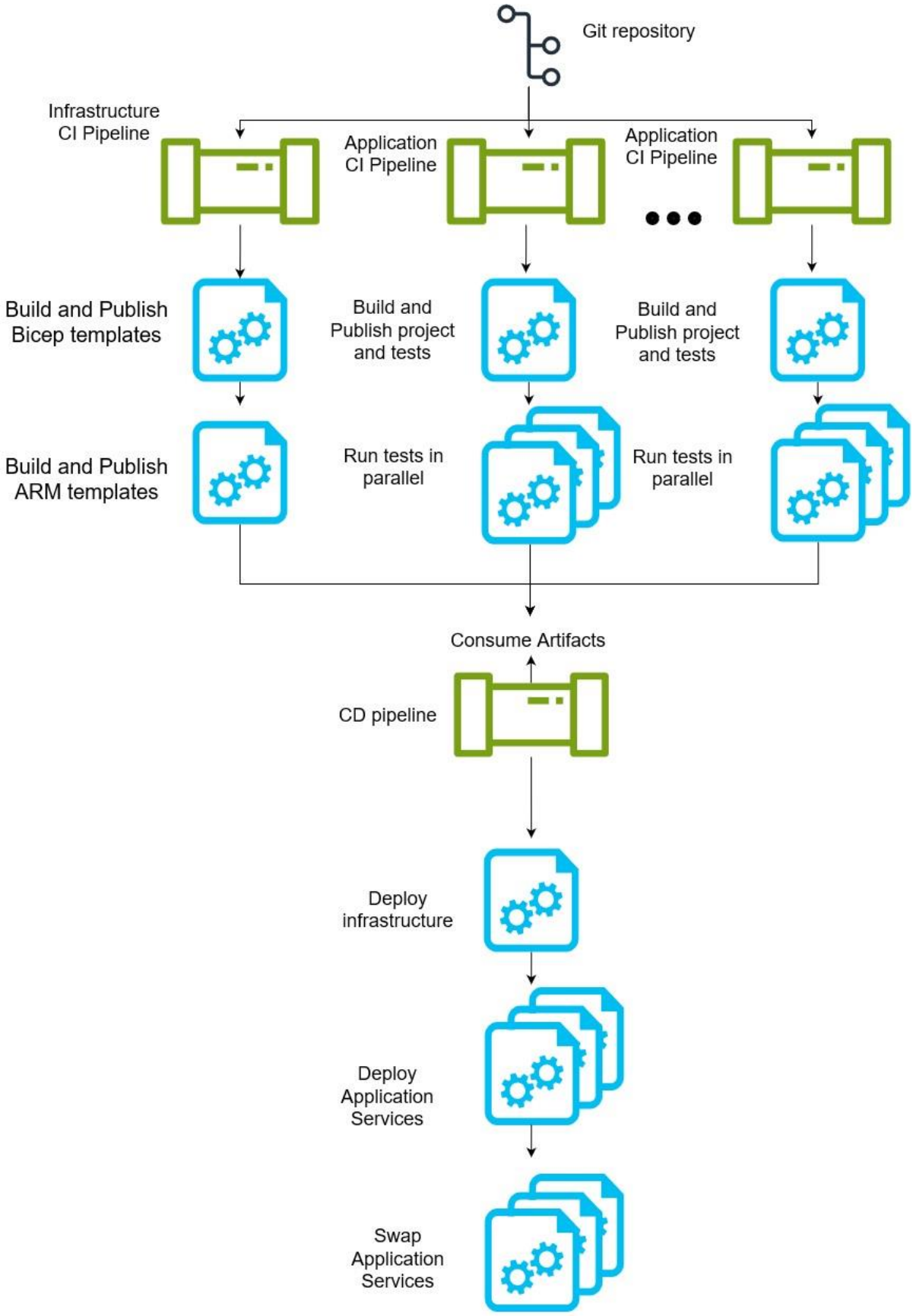

Figure 7. CI and CD pipelines combined. The dots next to the Application CI pipeline symbolize that there can be any amount of CI pipelines producing artifacts for the CD pipeline. Three files with cogs symbolize that multiple jobs are running in parallel in that stage.

5.1.3 User Interface Testing

Feature testing or User Interface testing is done with Azure-hosted agents in TeamCity in the pre-migration setup. To speed up the process the User Interface (UI) testing that is done with Azure pipelines utilizes containers, which enables improved parallelity. A further time-saving improvement is to run the tests without loading the web browser's user interface, which is also known as running tests headless. Effort is also placed into solving problems with flaky tests so that the tests can be trusted in the future.

5.1.4 Problems During the Migration

A problem with a time-based unit test occurred when running tests on Azure pipelines, a problem that does not exist with TeamCity. The problem was that the test was failing because the date was supposed to be 2021-01-01 00:00, but now it suddenly was 2020-01-01 02:00. The problem was solved by changing the assert in the test from using localized time to just parsing directly to DateTime. The problem is most likely caused by the agents in Azure using Greenwich Mean Time and the test using Eastern European Time and this caused the time to be two hours more than it should have been.

A problem that occurred during removing dependencies caused a package not to exist for the project anymore, even though Nuget Package Manager showed that it would still exist. Perhaps due to the package being referred by another project which this project referred to. This resulted in the error: "System.IO.FileNotFoundException: Could not load file or assembly 'Microsoft.AspNetCore, Version=2.2.0.0 " The error was solved by removing the package with NuGet Package Manager and simply by re-downloading it.

A problem that occurred when testing deployments to the staging slot caused the slots to be swapped even though auto-swap was disabled in the slot configuration. The solution was to turn the swap on and to deploy and then turn it off and deploy again. The solution could suggest that the value was actually turned on, but the Azure Portal user interface just did not display it, which could perhaps happen if the auto-swap configuration name had been changed at some point.

Migration from Octopus Deploy requires changes to the project files as Octopus Deploy copies and moves the files when it deploys the software package. On a practical level, this meant that a new target element needed to be added to the project files, which moved all the needed files to the publish directory's bin folder. Additionally, for the web application the static files and the Roslyn files needed

to be copied over to the publish directory itself. Moreover, Some nuisances were also caused by the ongoing effort to update the .net version of the project.

6 Outcome of Migration

The objective of this chapter is to present the results of the case study's practical implementation. The chapter wraps up the practical part of the case study so that it can be used to compare the pre-and post-migration processes. Furthermore, the chapter is used to summarize the changes, which can be used as context to understand the data collected in the interviews and surveys. The chapter also provides empirical data on whether the migration was successful.

6.1 Post-migration process

6.1.1 Tools

Of the tools mentioned in Chapter 4.1, the CI and deployment tools were migrated. The migration process and the technical implementation is mentioned in the previous chapter and the purpose of this chapter is to go over the post-migration process on a more practical level.

As mentioned in the previous Chapter the Continuous integration tool and the Deployment tool were migrated to Azure DevOps and both of them are now Azure DevOps Pipelines based on YAML templates. As they are both in Azure DevOps, they are both accessed through it as well and they can be found in the pipelines tab, as illustrated in Figure 7. The pipeline tab provides an overview of all the pipelines associated with the specific Azure DevOps project. The default tab on the page is about the most recently run pipelines, but when looking for a specific pipeline it can sometimes be easier to go through the 'All' tab, which has the pipelines in a folder format.

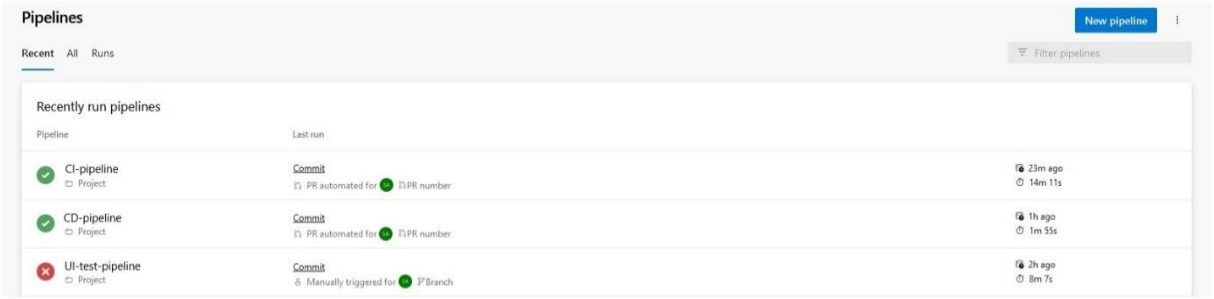

Figure 8. Pipelines overview in Azure DevOps

Pipelines that have tests in them have a test tab, which has information on the tests that were run for the pipeline. This information includes such things as pass rate, duration, number of tests and the specific tests, which have failed in the run. Through the tab the Test failure report for that pipeline can be accessed, which offers a historical overview of the test performance, the test failure report can be

seen in Figure 9. To get more TeamCity-like information the ‘Branches’ tab of the pipeline can be utilized. Figure 10 is a screenshot of the branches tab. The branches tab shows all the historical runs on red if they have had failed pipeline jobs and on green if not. As it is Azure DevOps does not offer a tab through which all the tests for the whole project can be seen, rather the tests are divided per pipeline. As in the Post-migration setup, there are multiple pipelines that have their own tests all these have to be accessed separately. However, the Test failure report shows a finer grain way of showing which tests have been shown recently than TeamCity did. Albeit the history is only limited to the last 30 days.

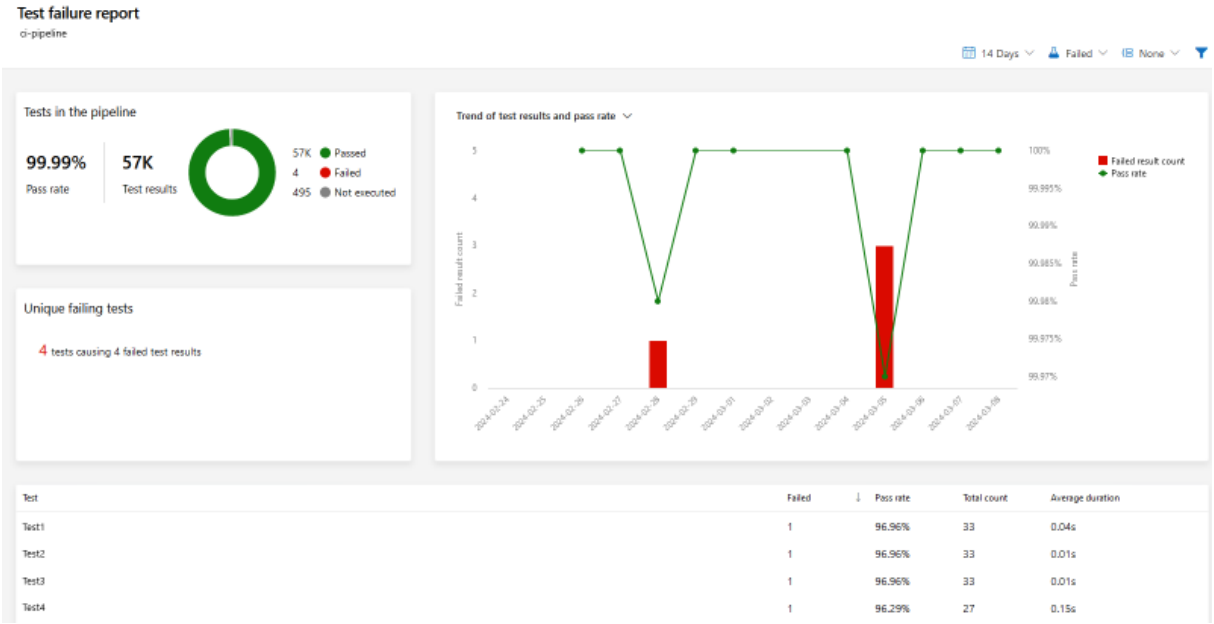

Figure 9. Test failure report.

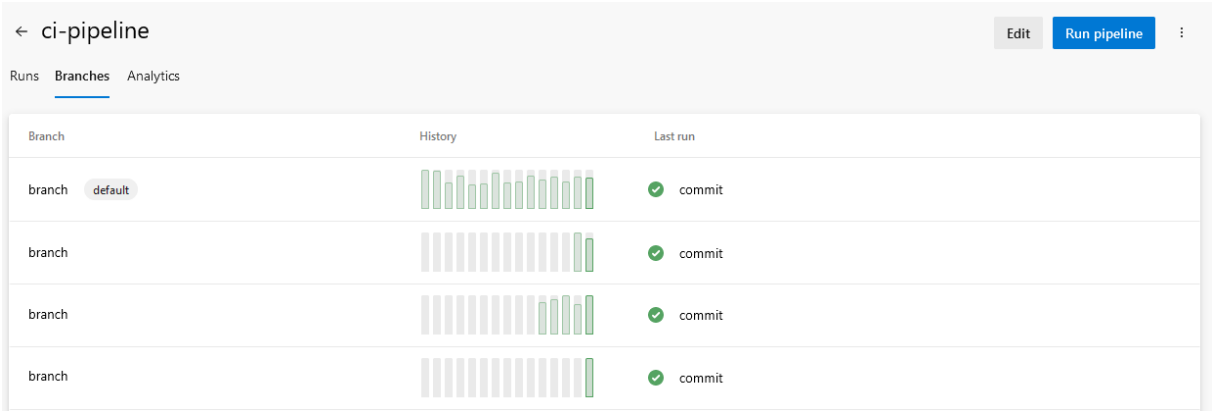

Figure 10. 'Branches' tab of a pipeline

To deploy the applications the CD pipeline is selected from the pipeline tab, then the run button is clicked and the environments into which the deployment is to be done are selected. After starting the

pipeline, the view that is seen in Figure 11 is presented to the user. In the pre-migration process, Octopus Deploy provided an overview of what version of the software was used for each application in each environment. In Azure DevOps pipelines the environment tabs can be used to see what version of the code is in each environment. Other options exist as well such as the Repos tab. It shows the git history of a repository, which has the commit names, Pull Requests IDs and their Statuses, as seen in Figure 11. The history can sometimes be a bit tricky to read as the status is shown as failed even if only one of the pipelines that have been run on the commit has failed. So, for example, if the feature test pipeline would fail on a commit, but the CD pipeline would have been run anyway the status of the commit would be failed even though that would be the latest version of the code. Therefore, to see what commit actually is in the environment, which is based on the branch the status needs to be clicked to see what pipelines have passed for that commit. Another way to see what version of the code is running in an environment is to find from the cd pipelines tab, the last commit that was successfully deployed into that environment.

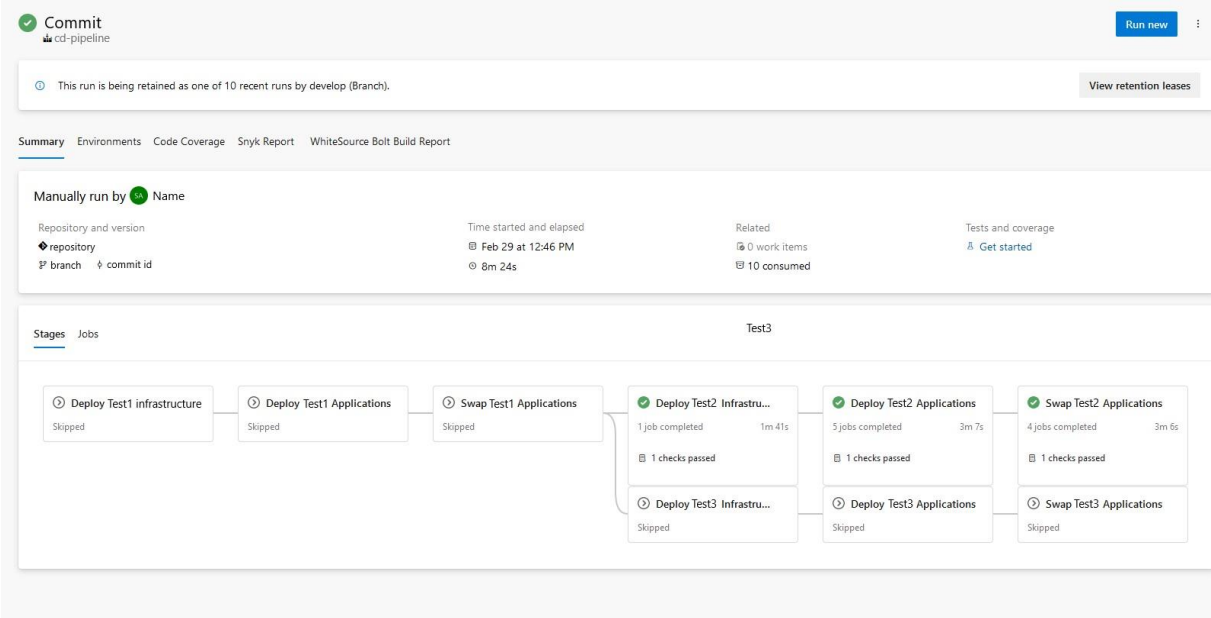

Figure 11. Result of running the CD pipeline.

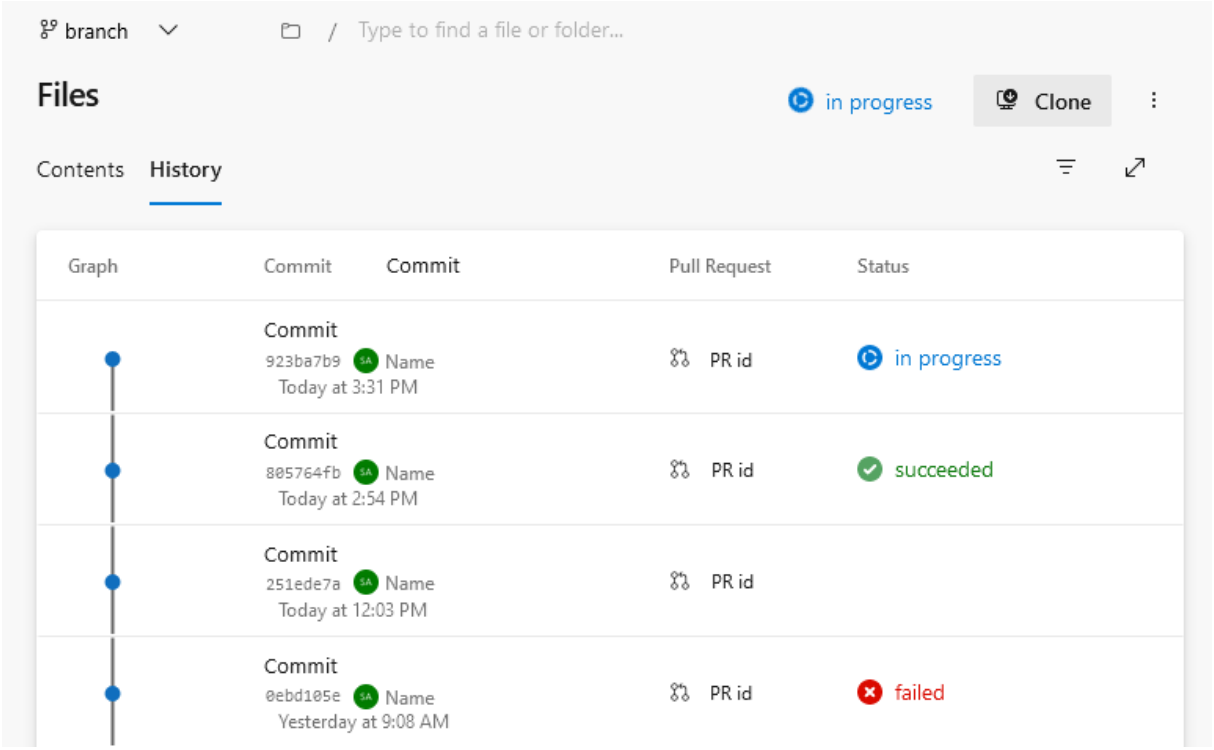

Figure 12. Branch history in Azure DevOps

6.1.2 Process

Before the Post-migration process can be adequately explained it must first be mentioned that the branches have been renamed. The old master branch, which is used for System Testing is now called develop. The release branch does not exist anymore instead a release-candidate branch is created, and the branch on which the Production branch is based is called master.

The Post-migration process starts when a Pull Request is created because the Continuous Integration pipelines of the projects that were changed have to be run before the Pull Request is completed. This is done to protect the develop branch's integrity because all code that ends up in the develop branch must now pass all the unit tests and it must also be buildable. This is important because develop is the branch on which System testing is done and from which all the Developers create their branches and if this branch is broken it makes it hard to create new branches. At the moment of writing this, the UI tests are not automatically run whenever something is merged or is going to be merged into develop, instead, it is a best practice to run all the UI tests that the changes affect. After the Pull Request is completed the Continuous Delivery pipeline automatically deploys the changes into the System Testing environment. After the new version has reached the System Testing environment the Developer changes the state of the story to 'ready for sprint testing in Azure DevOps.

The System Testing part of the process has remained quite the same as in the pre-migration process, the stories are still marked as ‘Done’, but instead of merging the develop branch into the release branch, a new release candidate branch is created from the develop branch. Then a Pull Request is created from this branch towards the master branch and in the Pull Request all the CI pipelines are run. The artifacts of these CI pipelines are then used to run the CD pipeline to deploy the changes into the Acceptance testing environment and then later into the Production environment. The Pull Request can also be used to find all the stories for the release query. The Acceptance testing of the tasks is similar to the pre-migration process; once the tasks are tested and accepted, they are marked ‘Accepted’ in Azure DevOps.

As mentioned in the previous paragraph the same Continuous Delivery pipeline run is used to deploy the release candidate to the Acceptance Testing and the Production environment. This is simply done just by approving the deployments to the Acceptance Testing environment and waiting for the release date to approve the deployment to the Production environment. The release branch, the Pull request, and the deployments are all managed by the Developer whose turn it is to be in charge of releasing. The deployment itself is done on a specific date, but there are now two different ways of deploying. The first way is essentially the Azure DevOps version of the pre-migration process, which is as follows:

1. Maintenance pages are turned on.
2. Deployment is approved to the production environment in the CD pipeline.
3. Content lifting is done.
4. Business verification and smoke testing is done.
5. Maintenance pages are turned off.

The whole process of deploying requires the maintenance pages to be turned on for at least an hour and sometimes up to two hours. The second way minimizes the amount of time the maintenance pages must be turned on for, by taking advantage of the staging slots. The idea behind the business verification is that the new software needs to be verified by the business before it can be presented to the customer so that the customers don’t have a bad experience. In the pre-migration system, the staging slots were not used at all for the deployments, which meant that the business verification must be done in the production slot, thus maintenance pages had to be turned for that period. Additionally, deploying straight on top of the old version causes a break in the services during, which the website cannot be accessed. However, this does not happen during swapping, because of the traffic redirection [62]. All these factors combined mean that there is a possibility to deploy without using maintenance pages at least on the theoretical level. However, in practice, it might not be entirely possible to deploy without using maintenance pages, because running a content lift task slows down the performance of the website, but the bigger problem is that the content could change during the time a customer uses

the website, which could lead to a bad customer experience. This is not all bad news though, because it would still mean that maintenance pages do not have to be turned on. The second process is as follows:

1. Deployment is approved to the production environment's staging slot in the CD pipeline.
2. Content lifting is done.
3. Business verification and smoke testing is done.
4. Swap slots whenever business verification is done.

The staging slots could also be utilized so that the steps from one to four would be done during the day before the release and the swap would be run on the next day, which could mean that the deployment process could be started later than at 7 a.m. Eastern European Time. Additionally, fewer people might be needed if the new software was already verified the day before.

The downside of using the staging slots to do verification is that there is no separate database for the staging slot to access, which means that the staging slot might change the database when a separate version of the code is deployed to it. The solution to this problem is to introduce a staging database that can be used for the staging slot, and it would be swapped at the same time with the application services. However, at the time of writing this, it is not ready, and it will be worked on outside of this thesis. The current workaround for this problem is to check what changes are going to be released and if there are database-breaking changes the staging slots should not be utilized. The whole post-migration process is illustrated in Figure 13.

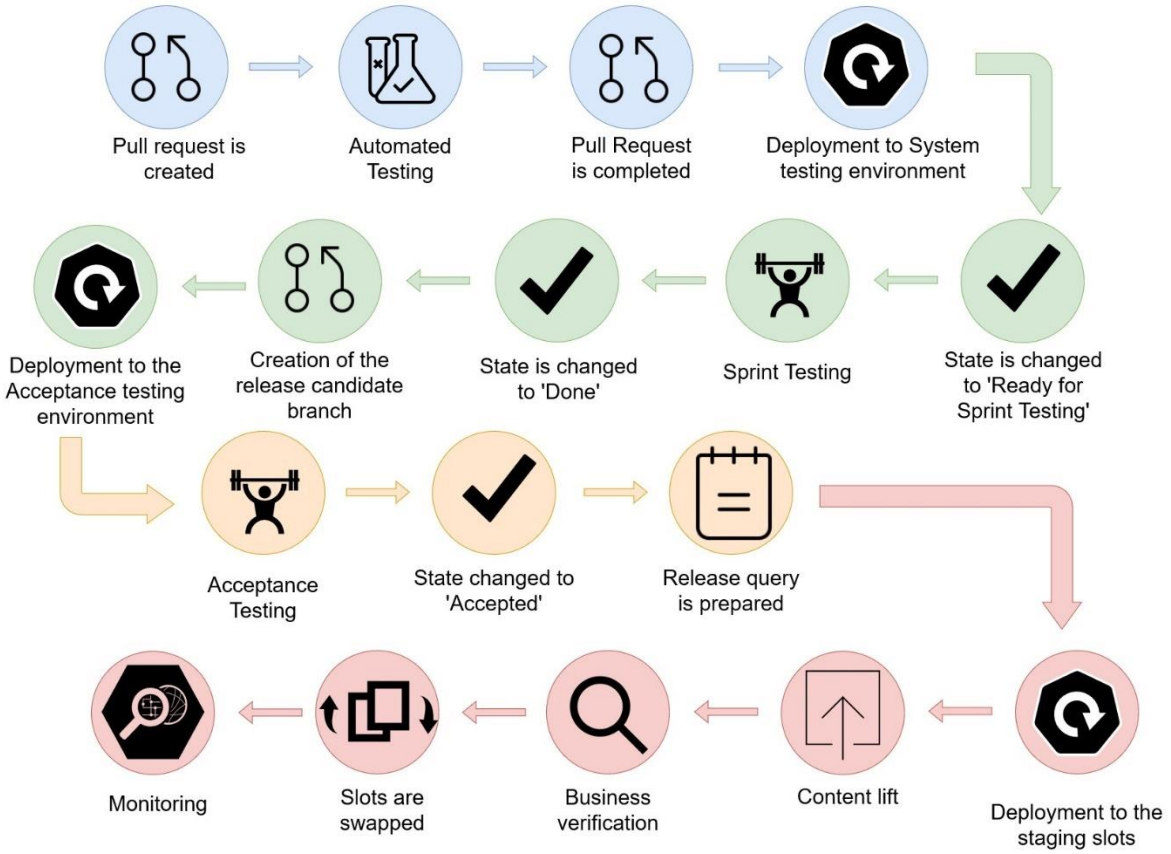

Figure 13. Post-migration Release and Deployment Management Process.

The last release into the production environment before the completion of this thesis was done using the staging slot approach and it was a success. However, there were also some issues with performance during that time as was predicted during the content lifting with response times averages rising to 10 seconds, when normally they are under a second as can be seen in Figure 14. This was also reflected during the performance testing of the production slot during the deployment, some pages took one to two minutes to load when normally it happens almost instantaneously. Additionally, there was a smaller spike in response times during the swapping of the applications with response times rising to two seconds, as can be seen in Figure 15. This however was not noticed by everyone who was testing performance during the swap. For some people, the swap happened completely seamlessly and for some they had longer load times than normally, but they were not as long as during the content lifting. The discrepancy could have been caused by caching because some people could have had their API calls cache while others did not, which could mean that for the people that had to call the API the load time was longer. These spikes in server response times have some impacts on the deployments that are done using the staging slots in the future because the usage of the website is a lot lower in the mornings than during the day. Therefore, it is most likely not advisable to do deployments during peak hours as that could make the response times even worse.

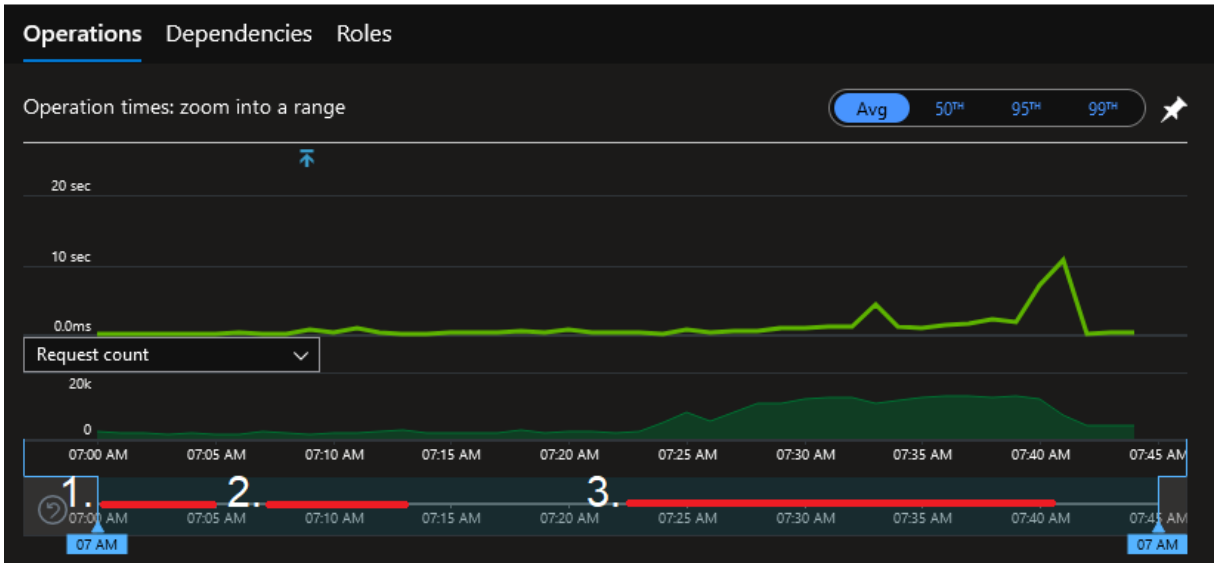

Figure 14. Performance during production environment deployment without maintenance pages. 1. Deploy Infrastructure 2. Deploy Application Services 3. Content lift import.

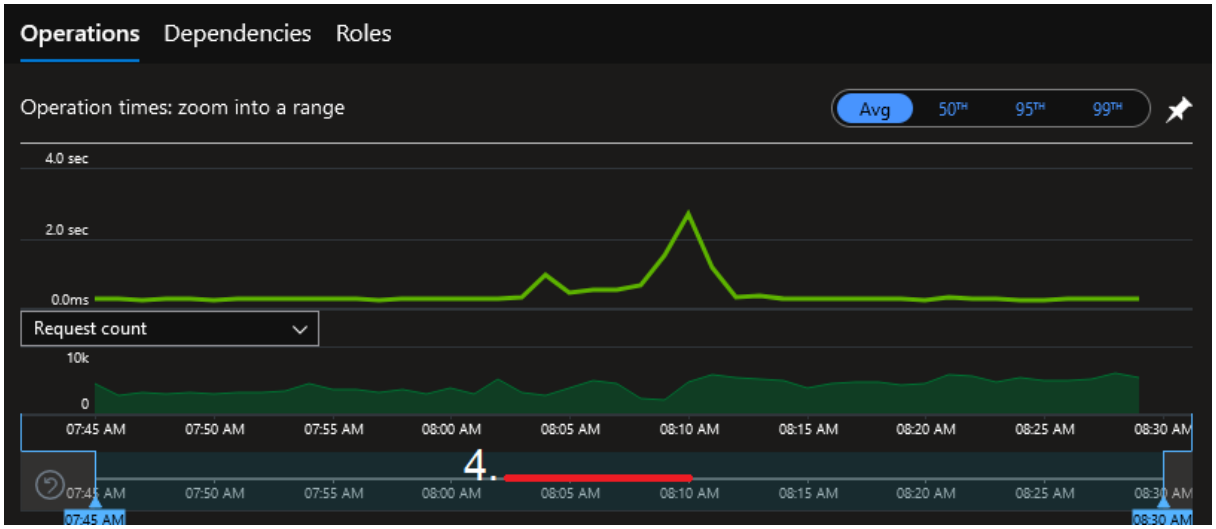

Figure 15. Performance during production environment deployment without maintenance pages. 4. Swap Application Slots

6.2 Comparing the Pre-migration and Post-migration Processes

The migration process had three main objects that are explained in Chapter 4.3 but roughly summarized they were moving away from the morning releases, no maintenance pages and retiring old Release and Deployment Management tools. The first objective is not quite achieved in the scope of this thesis, but the problem is partly solved during those occasions when a staging slot-based release is done. The morning releases are also mainly done because the Maintenance Pages have been turned on for the whole duration of the deployment, which was one to two hours. However, now that the maintenance pages are not required to be turned on when the staging slot-based deployment can be

done the process could be changed for those occasions. The second objective is again partly achieved during those deployments where there are no breaking UI changes so that the staging slot-based deployment can be used. However, the post-migration process requires a staging database to make the maintenance page-free deployments a reality for every deployment. The third objective is achieved, and the old tools have been retired and they are no longer in use.

As the process stayed somewhat similar while tools changed it could be argued that the Developers had the most changes in their daily work. Additionally, the Developers have more clear responsibilities now as the Developer who is going to be releasing takes care of the release branch. The structure of the Pull Request is different in the way that the relevant CI pipelines have to be run successfully before it can be completed. This also means that there is no longer such a long waiting period between the Pull Request being completed and the changes going to be ready for sprint testing and at the writing of this no deployment has ever failed after a Pull Request has been completed. Thus, after the Pull Request is completed, the changes are going to be in the system testing environment in approximately seven minutes. The change also ensures that the develop branch is never broken.

From a Quality Engineer perspective, the process has not changed that much, except the tasks reach the System testing environment quicker after the pull request has been completed. On a more practical level, this means that they get more work on their hand more reliably. The same goes for the Business Specialist who are doing acceptance testing. The production release has been changed a bit and now there is a Teams call for that event, but they are even more different if the staging slots are utilized for the deployment. Additionally, now a business verifier has to approve the deployment together with the Developer who is deploying.

6.3 Measuring Effectiveness

Measuring any concrete effectiveness is difficult when Azure DevOps is compared to TeamCity and Octopus Deploy because all of the tools are essentially just tools for managing virtual machines that are responsible for running Continuous Integration or Continuous Delivery / Deployment pipelines. Additionally, they are sometimes even running the same tests and scripts. This essentially means that depending on the virtual machine or how the tests are optimized the run can take longer or a shorter amount of time. For example, the same virtual machine scale set has been used to run UI tests in both Azure DevOps and TeamCity. Therefore, it could be said that the comparison should be done on what kind of information they provide and what are their user experiences like. The answers to these questions are in the survey results.

7 Interviews

This chapter covers the first data collection method used in the thesis which is the interviews. The chapter is divided into four sections. The first section is about the reasoning behind the questions used in the interviews. The second section introduces the backgrounds of the interviewees and the reason why these interviewees were chosen. The third section contains analysis based on the gathered data and the fourth section summarizes the data in respect to the research questions.

7.1 Interview Questions

The interview questions are based on the literature about Release and Deployment Management. The interview as a whole can be found in the Appendix 1 Section. The interviews are semi-structured, which means that they have a general structure, and all the interviewees have the same questions, but the questions are open-ended [2]. Additionally, the discussion is allowed to diverge from the path by allowing the interviewee to steer in the direction where it is going. The role of the interviewer is mainly just to listen and to ask follow-up questions after the interviewee has already chosen the direction. The purpose of the follow-up questions is to clarify what the interviewee is conveying or to get answers to why and how questions. The interviews are conducted with Microsoft Teams.

The purpose of the Background section is twofold; firstly, it gathers all the relevant background information on the interviewee that might affect their knowledge about the subject this includes role, experience with the pre-migration process and the experience they have of the field in general and experience of working in Scrum. Additionally, experience and interest in Release and Deployment Management are asked. Secondly, asking the interviewee to describe what Release and Deployment Management is and the conversation about what it is in the scope of this thesis makes sure that the interviewee knows what is meant by Release and Deployment Management.

The section Importance of Release and Deployment Management expands on the Background section by getting information on the relevance and importance of Release and Deployment Management based on their role. Furthermore, personal opinions on the importance of Release and Deployment Management are asked, which are going to be used to answer the RQ1

Of Release and Deployment Management section is about compiling information on what is important, hard, or what benefits can be achieved with Release and Deployment Management. The reasoning behind this is, is that the answers can be compared with the literature, to see if people who are working with a Release and Deployment Management system can identify the same hardships and

benefits that are present in the literature. Therefore, the questions and answers from this section are used to answer the RQ1

The section Agile and Release and Deployment Management collects data that could be used to answer the RQ2. Moreover, the questions are formed in the manner the answers can be used to examine if the interviewees form a similar connection between Agile and Release and Deployment Management that is present in the literature.

The Release and Deployment Management features section aims to gather information that can be used to answer RQ2. This is done by asking the actual research question and by trying to make the interviewees reflect on the feature by thinking about the pre-migration system they are working in currently. Additionally, the section acquires knowledge of what interviewees think about DevOps and CI/CD in Release and Deployment Management. This is done because DevOps and CI/CD are relevant in Release and Deployment Management, and they bring tools and features to it.
The Release and Deployment Management Strategies section is for answering RQ4, which means that it revolves around automatization and the requirements and challenges posed to it.

7.2 Backgrounds of the Interviewees

7.2.1 Reasons For Choosing Interviewees

The project consists of three different Scum teams that all work on the same solution and with the same Release and Deployment Management. Each team has a Scrum Master, Product Owner, Lead Developer, UX Designer, Quality Engineers, Business Specialists and Developers. Additionally, the three teams share an Architect and a Release Manager who is working in that role part-time, while mostly working as a Scrum Master on one of the teams. From each role that had multiple people in it, one person is chosen to represent that role. For the Product Owner, Lead Developer and Developer roles the people who have been working with the solution the longest were chosen. For UX Designer, Quality Engineer and Business Specialist, persons from the same team as the author are chosen because it is thought that it could be beneficial to be able to use the team's work and practices as examples. The chosen Scrum Master has the most experience working as a Scrum Master if the Scrum Master who works as a Release Manager is excluded. Therefore, because it is thought to be beneficial that a person who is only working as a Scrum Master is interviewed, technically two Scrum Masters are interviewed because the Release Manager works also as a Scrum Master. There is only one Architect on the three teams, which means that no choice has to be made. All in all, nine interviews were conducted.

7.2.2 Backgrounds of the Interviewees Based on the Interviews

Table 2 Backgrounds of interviewees.

Role	Experience in the Team?	Experience in the Field?	Experience in A Scrum Team?	Experience Of Release and Deployment Management.	Interest in Release and Deployment Management?
Developer	Over 2 years.	6-7 years.	Working in agile, but not in Scrum. before this team.	Experience in using systems, not building them.	Interested in learning more, but maybe not "super" interested in general.
Lead Developer	5.5 years.	~9 years.	Working in Agile, but not in Scrum before this team.	Experience in using systems, not building them.	Professionally yes.
Architect	~2 months.	6 years as an architect and in IAC solutions 4-5 years.	12 years.	Experienced.	Interested.
Quality Engineer	1 year 3 months.	2.5 years.	2.5 years.	Experienced working in a system, not building	Some parts are interesting.
Release Manager / SM	7 years.	20+ years	10+ years.	Experienced.	Interested.
Scrum Master	8 months.	11 years.	11 years, but occasionally as an agile coach.	Experience in processes.	Interested in knowing how it is done.
Product Owner	7 years.	5 years.	8 years.	Experience in processes.	Interested in the outcome.

Business Specialist	1 year.	1 year.	1 year.	1 year while working in the team.	Interested.
UX Designer	~1 year.	3 years.	1 year 9 months.	Very minimal to none.	Some parts.

7.3 Results of the Interviews

The experience measuring questions with their answers are listed above in Table 2.

7.3.1 Importance of Release and Deployment Management

Every role interviewed apart from Scrum Master and UX Designer thought that Release and Deployment Management is part of their work and that it is important to their role as well. Most of the reasons why it was thought to be a part of someone's work are related to working in a Release and Deployment Management system. At the same time, some of the interviewees mentioned that Release and Deployment Management is not the main thing being worked on, but rather it is something that is always affecting in the background. The most common answer to whether Release and Deployment Management is important to one's role could be condensed into the answer the interviewed Product Owner gave: "Yes, how would we otherwise get anything out to the customers?". The answer emphasises the thinking of the interviewees in the sense that software development and the work done by the team is not done for the sake of developing something, but that in the end, it revolves around bringing value to the customers.

Neither of the Scrum Masters thought that it belonged to the job description of a Scrum Master nor that it would be important to their role directly. The indirect influence comes from the hope the other interviewed Scrum Master expressed "It's important to my role in the way that I hope we would have a process that works well.". This could be seen as the Scrum Master having the power to facilitate improvements to the process. The interviewed UX Designer did not think that it was part of their work, but they did think that it is indirectly important to the role of UX Designer. The reason behind the indirect importance was that Release and Deployment Management oversees the process in which the designs are released to the customer.

The responses to the question if your team pays enough attention to Release and Deployment Management can be roughly divided into three groups. In the first group the answer is a clear no, in the second group people think that this thesis and the new architect joining have caused a big shift in

attention and because of it enough attention is being paid, the third group is indecisive and sees multiple sides to the question.

The first group thinks that the problems in the pre-migration process prove that enough attention is not being paid to Release and Deployment Management. Examples of these problems are unreliable automation, breaking test environments and the process itself not being smooth enough. One of the interviewees in this group theorised that the reason behind not enough attention being paid might be that there has not been a price calculated for the dysfunctional process. This means that if the amount of resources being wasted on existing problems would be calculated it would make it clear that it takes up more resources to work in a dysfunctional process than to fix the process.

The second group thinks that enough attention is paid now because of this thesis and a new architect joining. Some persons in this group identified that there have been long-existing problems in the Release and Deployment Management that need to be fixed. For instance, moving the Release and Deployment Management tools to Azure DevOps is seen as something that has been recognized a long time ago already.

The people in the third group don't give a clear answer to the matter. One of the persons in the group for instance thinks that as multiple different parties are competing for the teams' resources it might not be plausible for the team to make the decision themselves to spend the resources on improving the process. Another person in the group thinks that enough attention is being paid to some things and then to others not.

All the interviewees thought that it would be worthwhile to spend time in Release and Deployment Management. The process is also seen as something that should be iteratively improved and that there should be thought behind managing the process. Moreover, some interviewees said that the process should be working from the beginning and considerable amounts of time can be used to create it. Some of the answers to the question were similar to the ones from the questions about the benefits of Release and Deployment Management and therefore they are going to be listed in the next section.

7.3.2 Release and Deployment Management

The answers to questions 11, 12 and 14 are listed together as they had such similar answers. Additionally, this section answers to what are the best practices and what is thought to be hard in Release and Deployment Management.

The most common benefit that was attributed to Release and Deployment Management was quality. Eight of the nine interviewees identified that the end product would be of higher quality if there was a Release and Deployment Management system in place. The second most commonly identified benefit was that with Release and Deployment Management the amount of manual work needed to deliver code can be reduced through automation. Less manual work and automation were tied into a single category because it is felt like the other is given if the other part of the equation is mentioned. For instance, some interviewees mentioned that with Release and Deployment Management things can be automated and others mentioned that less manual work is required with Release and Deployment Management. All in all, the shared category of less manual work and automation was mentioned by seven of the interviewees.

Tracking and traceability were mentioned both by six interviewees and that makes them share the third place in the list for most commonly identified benefits. Tracking means the ability to track the release and deployment processes, but traceability goes beyond it by extending the tracking possibilities through historical releases and deployments. Although both of them were always mentioned by the same interviewee, which could mean that they are identified as something that goes hand in hand. The terms tracking and traceability were not always mentioned, but if an interviewee said that one of the benefits is that it can be known what is released and when it is counted towards the interviewee identifying tracking as a perk. The same applies to the term traceability, if an interviewee mentioned that keeping records or that they thought that information of the past releases and deployments is a benefit they were counted as thinking that traceability is a benefit.

Five interviewees out of nine mentioned either a form of flexible value delivery or being able to quickly deliver value as a benefit of Release and Deployment Management. Four persons from that group mentioned both flexible value delivery and being able to quickly deliver value, but two persons mentioned only either one. If a person said something along the lines of releasing when needed or being able to release at will they were counted as thinking that flexible value delivery is beneficial. As for the being able to quickly deliver value synonyms for quick were considered for this category also if the interviewee mentioned that the process can be sped up with Release and Deployment Management.

As there are nine interviewees the identified benefits above were all majority opinions, the minority opinions are listed in this paragraph. Again, the interviewees might not have said the exact words of the category they were listed for, but something along the lines of that. Benefits that were mentioned four times are smooth flow of work and common processes. Benefits that were mentioned thrice were transparency, replicable value delivery and being able to recover from failures. Benefits that were mentioned twice were productivity, predictable value delivery, seamless delivery and the ability to be

able to track error cases. Benefits that were mentioned once were system integrity and safe and stable value delivery.

The unofficial most common benefit that might not have been implied directly. but was present in all of the interviews was that through the other benefits, time could be saved. Thus, it could be argued that the interviewees saw that the goal of Release and Deployment Management is to save time in the end. It does not matter whether it's saved by delivering a quality product that does not need any fixing later, or through the process requiring less manual work, working becomes more efficient and that saves time. The interviewees managed to list all of the attributes that were found in the literature review. Additionally, common processes came up as a benefit, which is akin to the thinking that the deployment process must be enforced by all including both Dev and Ops personnel [33].

All of the interviewees answered roughly at least one of the following best practices in Release and Deployment Management: testing, automation, documentation and frequent releases to the production environment. Testing was mentioned in two different forms, some of the interviewees answered that it should be automated testing, but some answered only that everything should be tested before released. Automation was answered by many sometimes together with testing and sometimes as its own thing. Documentation and common practices are grouped together as documentation because they both are about knowledge sharing. One of the interviewees highlighted the relationship between documentation and automation as “Everything that can be automated should be automated the rest should be documented.” Some of the interviewees mentioned that they would like to see more deployments to the production environment and one of them even recalled the phrase “small and often”.

The most common answer to the question “What is hard about Release and Deployment Management” with three interviewees answering it, is working automation. One of the interviewees with that answer explained the reasoning to be because it unlocks everything else or if it’s not working then nothing is. The second common answer with two persons answering it was that it is hard to know what should be actually released and when should it be released. Having sufficient knowledge of the process was also thought to be hard.

7.3.3 Release and Deployment Management Features

Features as term turned out to be too vague and the answers for what are the most important features in Release and Deployment Management reflected the vagueness of the term. The single most important feature turned out to be automation, which was mentioned by five interviewees. Automation included automated testing in three of the answers and in one of the answers the interviewee was

talking about automatically creating documentation through the process. Visibility, transparency, or traceability were mentioned in four of the interviews and they are grouped together into a single category, because they were often talked together, or they were thought to mean similar things. Generally, visibility, transparency, or traceability were thought to be related to knowing what is where and what is going to be released or what was released. Other answers to the question included ease of use, configurability, quality and planning.

The most common answer to the question “What is currently done well regarding Release and Deployment Management” was some version of “We have a process in place”, six of the nine interviewees answered this. The interviewees who answered this thought that the process enables the team to have consistency and that people know what they are expected of. The second most frequent answer with three out of nine interviewees answering this, was that the automation that is in place is good, two of them being specifically about test automation. In the last place is documentation with two answers.

The most common answers to the question “What people wanted to do differently” were more deployments and more automation. four of the nine people wished that there would be more deployments or that the work items would flow through the test environments smoothly. Three of the nine people answered that they would want to see more automation. Three people also identified (one of them on another question) that the knowledge of deployments and the tools was not spread around enough, meaning that there is too much dependency on a few senior Developers. Two of the nine interviewees thought that we should migrate to Azure to get full advantage of the offered tools.

All the people who knew what DevOps is thought that it makes Release and Deployment Management better or that it is the standard way of working, some people also answered that it would be weird to have a Release and Deployment Management system without having DevOps. The term DevOps has been muddied a bit now that Azure DevOps also exists, so usually it was needed to clarify which the question was about. CI/CD was seen similarly to DevOps; people thought of it as a positive thing or a somewhat standard thing in Release and Deployment Management.

7.3.4 Agile and Release and Deployment Management

The interviewees were almost unanimous in their answers related to Agile and Release and Deployment Management. All of the interviewees answered that the Agile-related practices and principles affect Release and Deployment Management, examples of these practices and principles are releasing in sprints and working iteratively. Two of the interviewees mentioned that Release and

Deployment Management don’t have to be tied together if, for instance, the release is not part of the Agile team’s work. Seven of nine interviewees thought that Release and Deployment Management is something that enables working in agile and that bad Release and Deployment Management can make the process less agile or slower. Scrum's influence on Release and Deployment Management was seen the same way as Agile affects Release and Deployment Management, which is no surprise because most of the ways people thought that Agile affects Release and Deployment Management are present in Scrum as well.

Eight of Nine interviewees answered that Release and Deployment Management is part of the Agile processes. The reasonings for the answers seemed to be along the lines of that the processes cannot be separated that they should flow together or that Release, and Deployment Management is baked into the Agile processes. The only interviewee who did not think it was part of the agile processes, nevertheless, thought that Agile shapes the way Release and Deployment Management is done.

Release and Deployment Management tools were seen as something that can influence the Scrum process in a positive manner by all of the interviewees. The positive effect of the tools was thought to be that the process could be made faster, smoother, more agile, and increase the trust in the integrity of the product. All of the interviewees also thought that the tools could be used to make following the Scrum process easier. Albeit two of the interviewees mentioned the Agile Manifesto phrase “People over tools”, which also highlights that the tools are in place to enable the work not to dictate it.

7.3.5 Release and Deployment Management Strategies

When asked to choose between a feature or a time-based Release and Deployment Management strategy six out of the nine persons answering thought that a feature-based strategy is better, two were indecisive and did not have a clear favourite, and one thought somewhere in between like a hybrid model would be the best. For the first time in the interviews, there is a clear distinction between the people who have more technical backgrounds were of one opinion and the rest were of another. The people who are said not to have a technical background are the PO, Business Specialist and UX Designer. All of the people with a more technical background thought that the feature-based strategy was better.

When asked about the pros and cons of the feature-based strategy, six out of nine people thought that deploying smaller features means that a single feature reaches the customer quicker. Both people who do a lot of testing (Quality Engineer and Business Specialist) thought also that the feature-based strategy would have a smoother flow of work and there would not be huge pile-ups of features to test.

Two of the nine persons gave clear cons, and they were essentially the same; as the team has not worked with such a system before it may be difficult to begin with.

The most common pro for a time-based strategy with four people saying it was that it is nice to have a specific date on which to release or that releasing after a sprint makes sense. Two out of nine people thought that it's good that it makes testing whole flows possible. Other pros include that if the code is longer in a test environment it could make it safer, delivering what is ready during that time. The identified cons were features cannot be released when they are ready, if something breaks the release nothing can be released, it takes a lot of manpower to release, doing extra releases is painful and it might make people rush their work. None of the pros were mentioned more than once, but a pro for the time-based strategy could be considered a con for the feature-based strategy and vice versa.

The most common answer to the question “How much should be automated?” was some form of “Everything that is possible to be automated should be automated”. Everything possible means mostly that all the repeatable parts of the Release and Deployment Management process. Only in places where intelligence is needed, or some important decision is met most of the interviewees thought that that step should remain manual. Two of the nine people did say or describe that they would essentially want the whole Release and Deployment Management to be automated.

The difference between the terms continuous deployment and continuous delivery was not that clear and in seven out of the nine interviews they were explained, which is not surprising considering the ongoing debate in IT literature as well. Five out of nine people thought that Continuous Deployments would be better, and four interviewees thought that Continuous Delivery would be better. Something to bear in mind is that the current solution that the teams currently are working on was described in the interviews as a highly complex system that has a lot of dependencies by three out of those four interviewees who preferred Continuous Delivery. Two interviewees out of those five who preferred Continuous Deployments also said that it would be the preferred system, if possible, which could indicate that it is thought of as more difficult to manage.

Six out of nine people identified that the benefit of feature toggles is that they can be used to turn on and off features, which allows the team to have a fail-safe and release functionality that for instance depends on other dependencies. However, four out of nine people also were of the opinion that cleaning them up creates technical debt and two out of those four people thought that it’s difficult to know when the toggle can actually be removed. Some of the interviewees also thought that the manual aspect of turning them on and off might create problems. Six out of nine interviewees thought that feature toggles presented an opportunity to have a completely automated Release and Deployment Management system and that it could be a solution to the problem created by Content Management.

Nevertheless, three out of nine interviewees mentioned that the current way of managing feature toggles is not proficient enough to use them to automate the whole process.

Content management was seen by all of the interviewees as something that makes Release and Deployment Management harder, mostly because of the manual aspect that someone needs to set the content. Therefore, it was also identified as something that would not make it possible to have a completely automated process.

7.4 Summary of the Interviews Based on the Research Questions

Table 3 Summarizes the results of the interviews based on the research questions. The table is further used together with Table 5 to derive the conclusions of this thesis.

Table 3. Summary of the interviews based on the research questions.

Research Question	Summary
RQ1. How does Release and Deployment Management support software development?	The importance of Release and Deployment Management is derived from necessity, but it is happening more in the background, and it is rarely the main focus. The most commonly identified benefits and how often they were identified that Release and Deployment Management brings are: 1. Quality (8/9) 2. Less manual work through automation (7/9) 3. Tracking and Traceability (6/9) 4. Flexible and Quick value delivery (5/9) 5. Smooth flow of work and Common processes (4/9) 6. Transparency, Replicability, Failure recovery (3/9) 7. Productivity, Predictability, Seamless delivery, Error tracking 8. System integrity, Safe and stable delivery Additionally, all the interviewees indirectly implied that it saves time. Release and Deployment Management's importance varies from role to role as the Scrum Masters and the UX Designer did not think that it was important to their role. Nevertheless, the Scrum Masters and the UX

	Designer did acknowledge that there is an indirect influence on their roles as well. Apart from those two roles, the members of all the other roles thought that they were important to their roles. Thus, Release and Deployment Management supports and influences each role differently.
RQ2. Do Release and Deployment Management tools make it easier for a scrum team to follow defined workflows?	All of the interviewees thought that Agile practices affect Release and Deployment Management. However, some mentioned that deployments and Agile do not have to be tied together. Seven interviewees thought that effective Release and Deployment Management enables working in Agile, while an ineffective one can make the process less Agile. Scrum's relationship to Release and Deployment Management was seen similarly to Agile's. Eight interviewees thought that Agile and Release and Deployment Management are intertwined and that they should flow together. All of the interviewees thought that Release and Deployment Management tools can make following the Scrum processes easier and they can influence the Scrum process in a positive manner. The positive effect of the tools was thought to be that the process could be made faster, smoother, more agile, and increase the trust in the integrity of the product
RQ3. What aspects of Release and Deployment Management tools are most valued by a Scrum team?	Release and Deployment Management tools should be able to be updated iteratively. The most important feature offered by Release and Deployment Management is automation, which was answered by five interviewees. Automation can mean test automation, automatically creating documentation or automated deployments. The second most important feature is to offer visibility, traceability and transparency, which means that the tools should offer a clear overview of the process. This was mentioned four times. Feature-based deployment strategy is preferred by the interviewees of technical background (Dev, Lead Dev, QE, SM, Arch), while a time-

	based or hybrid approach is preferred by the interviewees with a non-technical background (PO, Business Specialist, UX Designer). Therefore, the Release and Deployment Management tool should offer configurability, so that the strategy of choice can be executed. Configurability is further emphasized by some of the interviewees as all software projects are different and they all have different needs. Turning features on and off was identified by six out of nine interviewees as the purpose of feature toggles. Thus, it could be argued that Release and Deployment Management should offer the possibility to manipulate functionality in different environments.
RQ4. How much of the deployment can and should be automated by using modern CI/CD solutions?	"Everything that is possible to be automated should be automated", was mentioned by all of the interviewees, which roughly means that all repeatable steps should be automated and only steps in which human intelligence is needed should be manual. Five out of nine interviewees would prefer to have Continuous Deployments and the rest would prefer Continuous Delivery. However, the interviewees were thinking about it in the context of the solution they are currently working on, which was described as a highly complex system with a myriad of dependencies. Therefore, the "should" -part of the research question is heavily system-dependent, but the willingness to automate is there. Six out of nine interviewees saw Feature toggles as a way to automate the whole process even though the limitations placed by Content Management.

8 Survey

This chapter begins by presenting information on how the survey was conducted. After that, the chapter introduces the reasoning behind the questions in the survey and how they can be used to answer the research questions. The rest of the sections are for analysing the data gathered from the surveys. The first of those sections is about the backgrounds of the respondents and the next six sections are divided by themes. The last section of the chapter summarizes the collected data in respect to the research questions.

The survey is conducted twice on all of the three teams working on the same solution and with the same Release and Deployment Management. The first time the survey is carried out is before the migration and the second time is after the first system has been in use for one to two months so that the participants might not recall what they answered the first time and that they have gotten a feel for the new system. It is thought to be beneficial that the survey participants could not recall what they answered the first time so they will answer based on how they feel instead of directly comparing the two systems. Additionally, the new system cannot be surveyed before it has been used for long enough that people actually know what it is like. The survey as a whole can be found in the Appendix 2 Section.

The survey is a longitudinal survey that is conducted twice, and the system used for the surveys allows the pairing of the participant's first and second answers, therefore the relations between the answers can also be examined. The survey consists of mandatory close-ended questions and optional open-ended questions, as the main task of the survey is to collect quantitative data, but nevertheless, if someone wants to give qualitative data it is accepted as well. As mentioned before the survey is conducted on all three teams who are working with the same Release and Deployment Management practices and with the same solution. This means that the survey is emailed to three Product Owners, three Scrum Masters, three Lead Developers, three UX Designers, four Quality Engineers, eight Business Specialists, and sixteen Developers. In total this means that the forty people are expected to answer the survey.

The first survey was sent out on 2.11.2023 and the second survey was sent out 140 days later on 21.3.2024. When the second survey was sent out the teams had been working with the post-migration Release and Deployment Management process for 91 days and there had been three production releases done. Two of the first ones were done with maintenance pages on and the last one was made without maintenance pages by using the staging slots.

8.1 Reasoning Behind Questions

The survey is based on the current IT literature on Release and Deployment Management, the interviews conducted in this thesis and the system usability score (SUS). On the top of the first page of the survey is an explanation about what is Release and Deployment Management and it has been changed from the interviews because the explanation given in the interviews was thought to be a bit too technical by the non-technical people. The first six questions collect data on the participant's background that can later be used to group the participant according to experience, role etc. Questions from seven to sixteen measure the SUS of the system in place. Questions from 17 to 32 measure the system's qualities based on the claimed attributes of Release and Deployment Management in IT literature. The attributes are listed in Table 1. Question 34 is based on the interviews because the benefits and the need for documentation came up somewhat frequently in the interviews. These questions based on the IT literature are used to answer the RQ1.

The production releases section, or questions from 36 to 55, are exclusively from the production release point of view. This is because in the interviews people who did not have a tech background thought that the term “Release and Deployment Management” and the accompanied explanation for it were something technical and the connection between it and their work was vague. However, production releases were something that everyone had more to say about. Therefore, in the event that the term Release and Deployment Management is seen as something that does not influence the non-technical people, they hopefully will have something to say about production releases. The section covers Release and Deployment Management strategies by making the participant evaluate what are their opinions on the strategy in place. The questions are based on the literature as well as on the interviews and the answers are used to try to answer RQ3.

“Our release and deployment management process...” matrix gathers data for RQ2. Additionally question 32. “I think our release and deployment management supports my way of working” can be used to answer the question to a degree as well.

Questions 60-65 are about feature toggles and about CMS. Question 60. is from the interviews, as some of the interviewees thought that the process for managing feature toggles was not great.
The other questions are from IT literature and the questions are mostly used for answering RQ3, but as feature toggles have been thought to be related to automation, they could be useful for RQ4 as well.

“I think our Release and Deployment management tools are..” matrix measures what people think about the tools in use, additionally it measures if we are taking full advantage of them. This is done to get a fairer comparison between the tools, for instance, if people give negative feedback on the tools,

but also think that the tools are not fully utilized it could mean that the tools could be good if fully utilized. Questions 71 to 74 are for answering RQ4 and questions 75 and 76 are wildcard questions meaning that the answers given could affect all the research questions.

The “Closing comments” section is for gathering more background information and to measure what people think of the Release and Deployment Management on a more general level. Additionally, there is an open-ended question for gathering comments and feedback and a question where people can mention something that might have been missed in their opinion.

8.2 Background of the Responses

Before the survey results can be analysed it should be mentioned that the first survey was filled by 36 people, but the second one was filled by 33 people. The reason for this is that some people left the team, and some people were on vacation during the second survey. Additionally, one Developer moved to the role of Lead Developer after the first survey, which is why there are more tech leads now in the group LD/Arch in the second survey. This chapter and the next chapter are based on the first survey’s results apart from the statement “I think we are mature in Release and Deployment Management.“ since the data collected was similar no additional value can be had of it.

As mentioned before, the survey is filled by three different Scrum teams who are working on the same solution. The project has existed as it is for about eight years at this point and during that time people have left and joined and at this point, only nine out of the 36 people have worked with the project longer than 5 years. The experience with the solution is visualized in Figure 16, the answers were free-form in the survey, but they have been categorized in this diagram to offer better visualization. Some people in the “?” category could also have worked with the solution from the beginning, but as they did not specify explicitly whether were they talking about years or months they are left uncategorized. The Release and Deployment Management process and tools that were in use before the migration have been used for more than five years already. Therefore, the vast majority of the people have not been involved in setting up these environments and processes. However, some of the processes and tools have most likely seen some updates during that time.

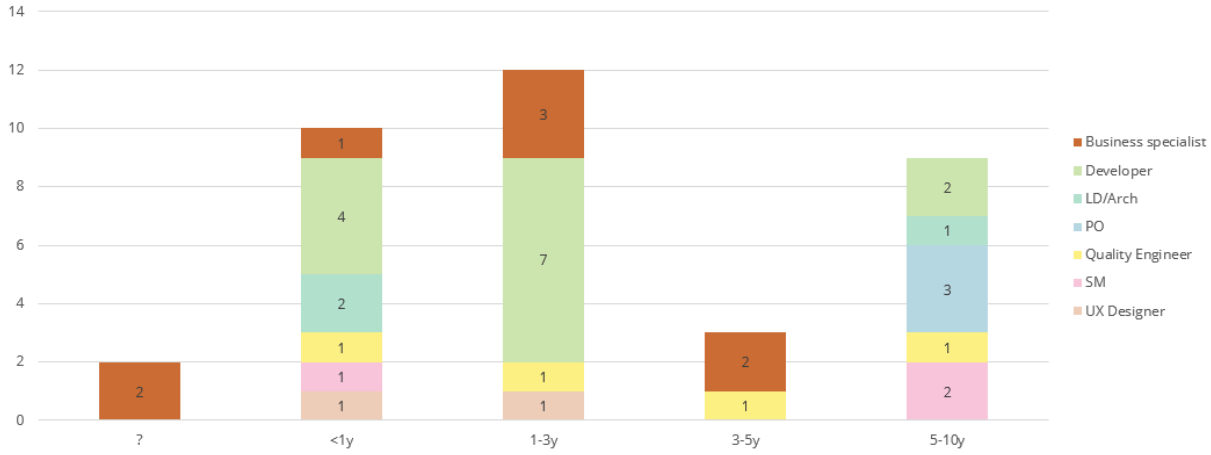

Figure 16. Experience with the solution.

Even though most people working in the solution have not worked with it for that long most people have more than 5 years of experience in the field they are working in now, as is shown in Figure 17. This could indicate that the people have been working with Release and Deployment Management systems and perhaps even with multiple different ones.

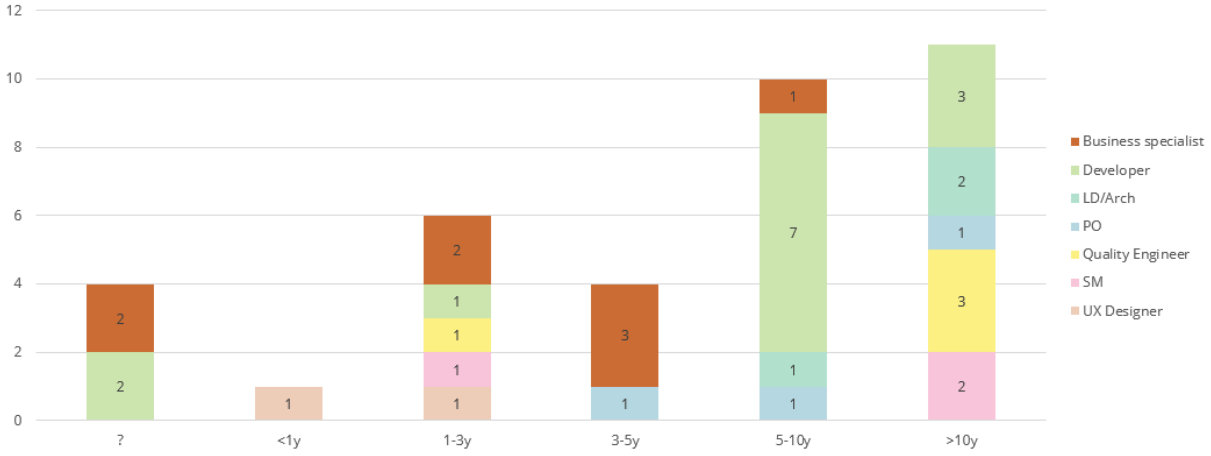

Figure 17. Experience in the field.

8.3 Experience in Release and Deployment Management

The experience in the different fields of Release and Deployment Management shows some contradicting data. The data is illustrated in Figure 18. Only 23 respondents think that they have been using Release and Deployment Management systems during their professional careers. Even though all the other mentioned options could be counted as using a Release and Deployment Management system. There is a clear division as well in the roles in this question, as all of the Developers think that they have been using the systems, but not every aspect of it. In contrast, the Business Specialists think that they have been working with some of the fields in Release and Deployment Management, but

they have not been using Release and Deployment Management systems. This could be related to the effect that was noticed in the interviews that people don't think they are working on Release and Deployment Management systems rather it's something in the background. Additionally, something noteworthy is that only 20 people think that they have worked with DevOps and 15 with CI/CD. Even though the system they are working on now is DevOps-based and arguably even CI/CD-based. The same can be observed in Figure 19, which is the same chart but with only experiences from the current solution. Additionally, some of the experiences the respondents have had from the current teams were not reported in the overall experience question.

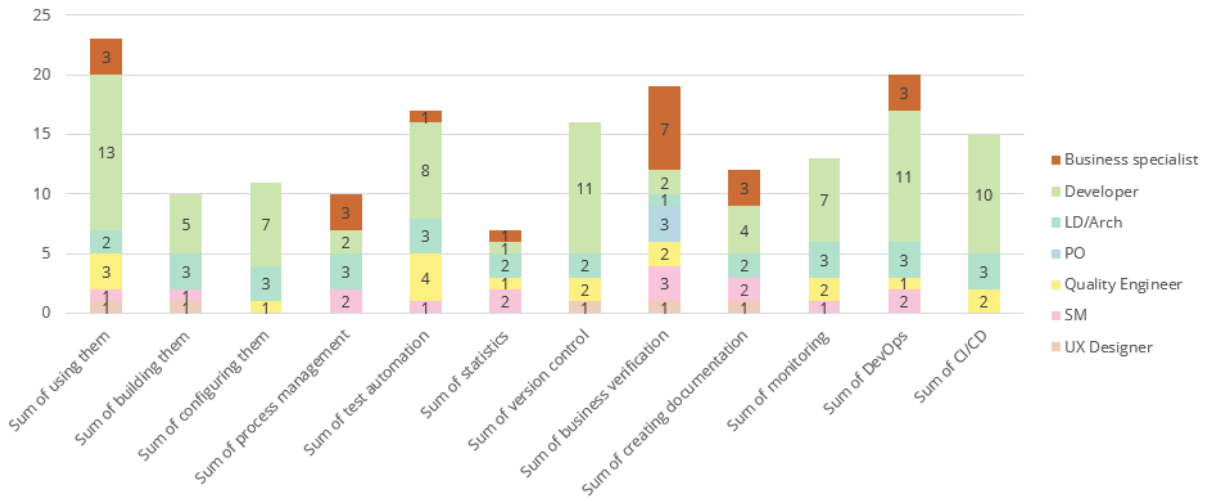

Figure 18. Experience in different fields of Release and Deployment Management

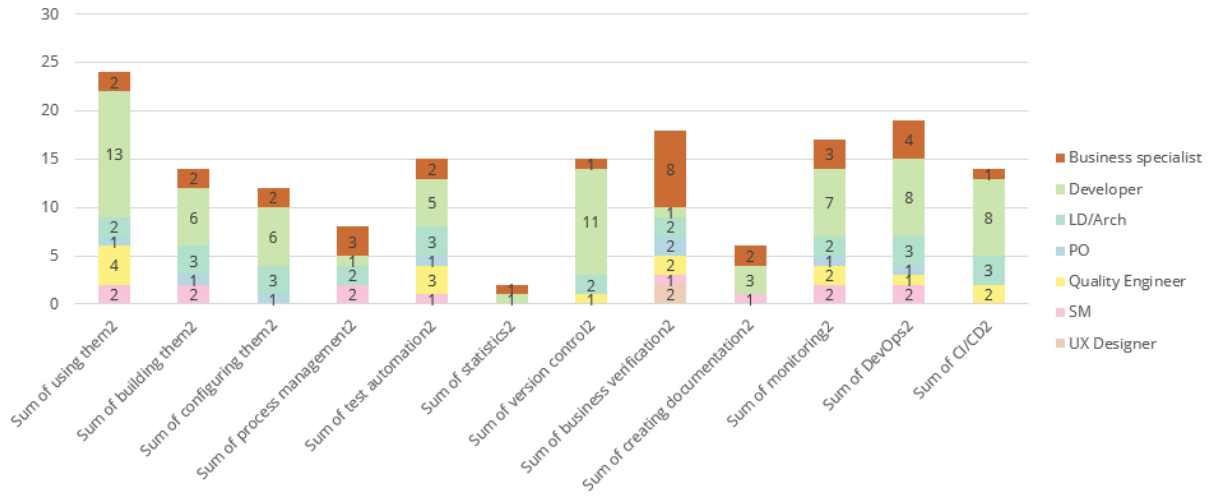

Figure 19. Experience in different fields of Release and Deployment Management in the current solution

27 people out of 36 only reportedly spend less than 30 minutes per day, as is seen in Figure 20. Additionally, one person even commented in the comments/feedback field that they only spend time

with Release and Deployment Management when there are production releases. This highlights how people see Release and Deployment Management as they don't think it is considered something they are spending time with, even though they are working in a system that has it. Only two people would strongly agree that they are experienced in Release and Deployment Management and five would somewhat agree, while the rest are either neutral or they would disagree, this can be seen in Figure 21. Out of the disagreeing people, sixteen would strongly disagree as well. There could be a causality between these two statistics since it could be difficult to know if someone is working with Release and Deployment Management if they don't know what it is. A respondent who strongly disagreed commented on the free-form field that they are not involved in the rest of the process they are just doing business verification. Whilst four of the eight respondents who commented on the free-form field answered that they know our process. Considering that 21 of the respondents have worked in the field they are working in now for five or more years it could be considered quite low that seven persons would be on the side of agreeing with the statement that they are experienced and that eight have a neutral opinion on the matter. This could mean that experience in the field does not mean that one is experienced in Release and Deployment Management and that it is more of a niche, even though it is present in every software project.

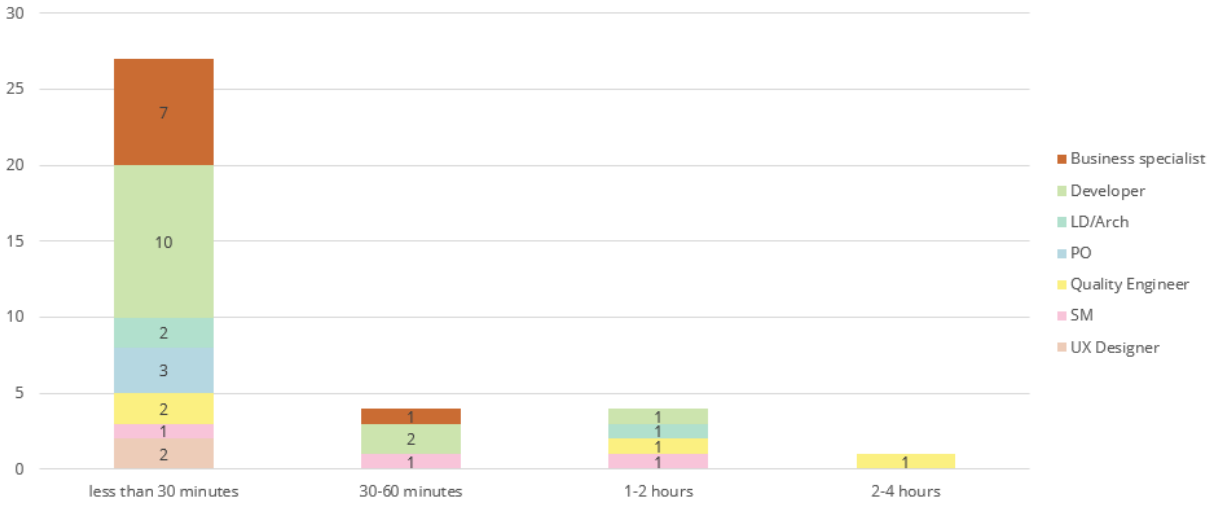

Figure 20. Time spent a day in Release and Deployment Management

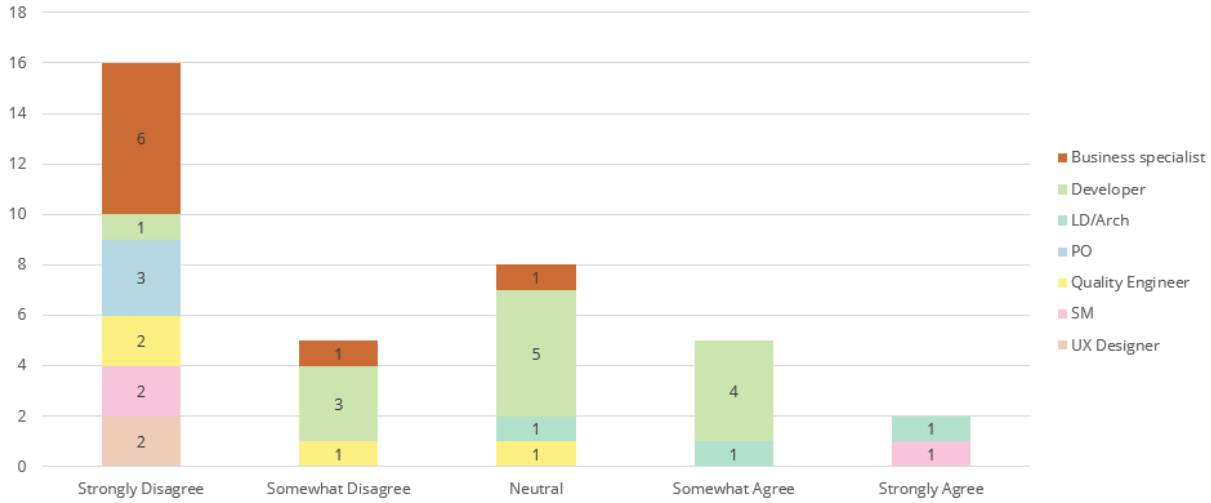

Figure 21. “I think I am experienced in Release and Deployment Management.”

The experience levels of the individuals reflected to a degree the answers if the respondents thought that the team was mature in Release and Deployment Management in the pre-migration survey as one person strongly agreed and five would somewhat agree, as seen in the left-side chart in Figure 22. The majority of people were neutral, did not know or somewhat disagreed. The free-form answer indicates that some people think that the teams are perhaps mature in the way it’s done now, but the processes and practices could not be considered as mature. six out of the eight respondents to the free-form field thought that the process could be improved. In the post-migration survey, the results of the free-form field were quite the opposite because five out of 8 persons thought that the teams were not mature with the process and four of them thought that because the process was new. Two Business Specialists thought that they could not answer the question because they had nothing to compare the process with. Nevertheless, there was an increase in people who thought that the process was mature with three more people answering somewhat agree and three people less disagreeing. Based on the free-form answers it could be thought that the people who agreed most likely thought that the post-migration process itself is more mature, but the ones who disagreed thought that the teams are not mature in working with the post-migration process.

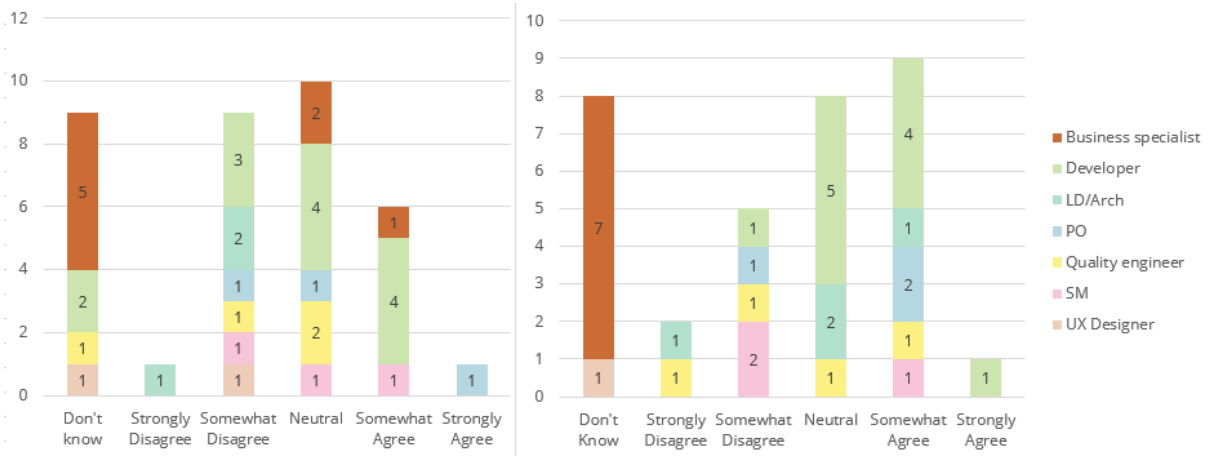

Figure 22. “I think we are mature in Release and Deployment Management.”. Pre-migration results are on the left and post-migration results are on the right.

8.4 Scoring the Release and Deployment Management Systems

The Pre-migration and Post-migration Release and Deployment Management systems were scored using two different methods. Firstly, the System Usability Score is used and secondly, the system is scored based on the qualities attributed to Release and Deployment Management systems in the current IT literature. The Attributes are listed in Table 1. The latter system is loosely based on the System Usability Score and the validity is further discussed in the Discussion chapter.

8.4.1 System Usability Score

The system usability score can be used to calculate a score that can be used to compare different questions, a score over 68 is average while under it is below average [63]. In the survey every closed question had the option of “Don’t Know” and as it is not normally used in SUS it is thought to be the same answer as neutral. This is because in SUS if the respondent does not know what to answer they should answer neutral; “If a respondent feels that they cannot respond to a particular item, they should mark the centre point of the scale.” [64]

The average pre-migration SUS score is 46,0417, which means that it is quite a lot below the average. If a person replies “Don’t Know” or neutral to all questions their SUS score would be 50 and as there were quite many “Don’t Know” answers from POs, SMs, UX Designers and Business Specialists; from 160 answers 109 (68.12%) “Don’t Know” answers and 19 (11.87%) neutral were given, which resulted in the average score of 45,4688. Nevertheless, this is not the reason for the low score, because the average score from the rest of the respondents is 46,5. There is no clear pattern detectable based on work experience with the pre-migration process and the SUS score nor is there a relation between

overall work experience and the SUS score. Figure 23 is a chart in which the SUS scores are sorted by role, which shows that only two people answered a favourable SUS score.

Even though no patterns can be in the Pre-migration survey alone that is not the case in the post-migration survey, as in Figure 24. A clear pattern emerges all of the Developers and the members of the LD/Arch group scored the system higher than 50 apart from one Developer. The Scrum Masters did not see an improvement in the system, as in the pre-migration system two SMs gave a score of 45 and one 37.5, but in the post-migration system, they gave 37.5, 40 and 45. Three Quality Engineers scored the system higher than last time, but one scored it lower. Similarly, there was a slight improvement in the scores given by the Product owners and a similar increase in the points given by the Business Specialists can also be seen, albeit that most of them are in the perfect neutral score of 50. This is due to neutral and “Don’t Know” answers, which occurred in the post-migration survey at somewhat similar rates. POs, SMs, UX Designers and Business Specialists gave 80 “Don’t Know” (57.12%) and 24 neutral (17.14%) answers out of 140 possible statements. The average score for that group is 48,035, which is only 3 points better than the pre-migration score of 45,4688, thus no significant change can be noticed. That being said, there was a significant increase in the other group with Developers, LD/Arch and Quality Engineers as the average for that group is 62,23 and without the Quality Engineers, the average would be 66,16, which is almost the SUS score average. Scoring a Release and Deployment Management system using the SUS score might not give a fair comparison as the SUS score is meant to be used for all kinds of systems and the project the Release and Deployment Management system at hand is designed for was said to be highly complex in the interviews. Therefore, another scoring system that uses the Release and Deployment Management attributes is used as well.

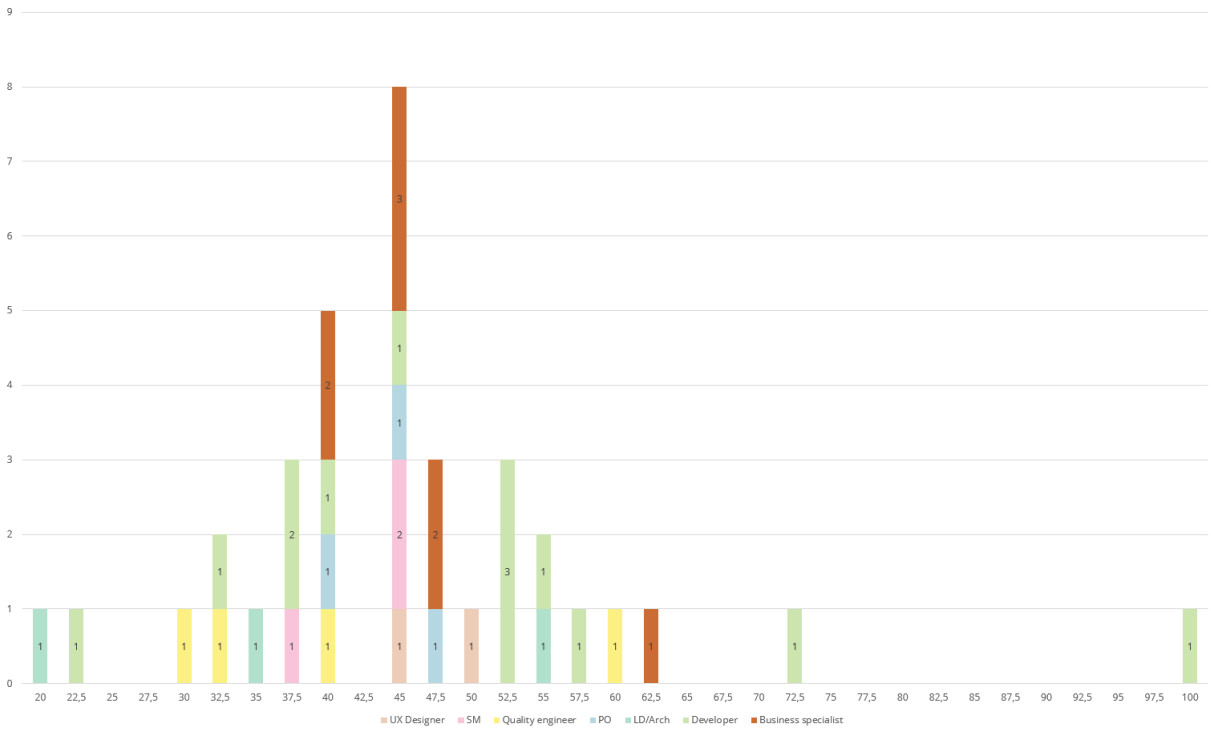

Figure 23. Pre-migration SUS scores by role.

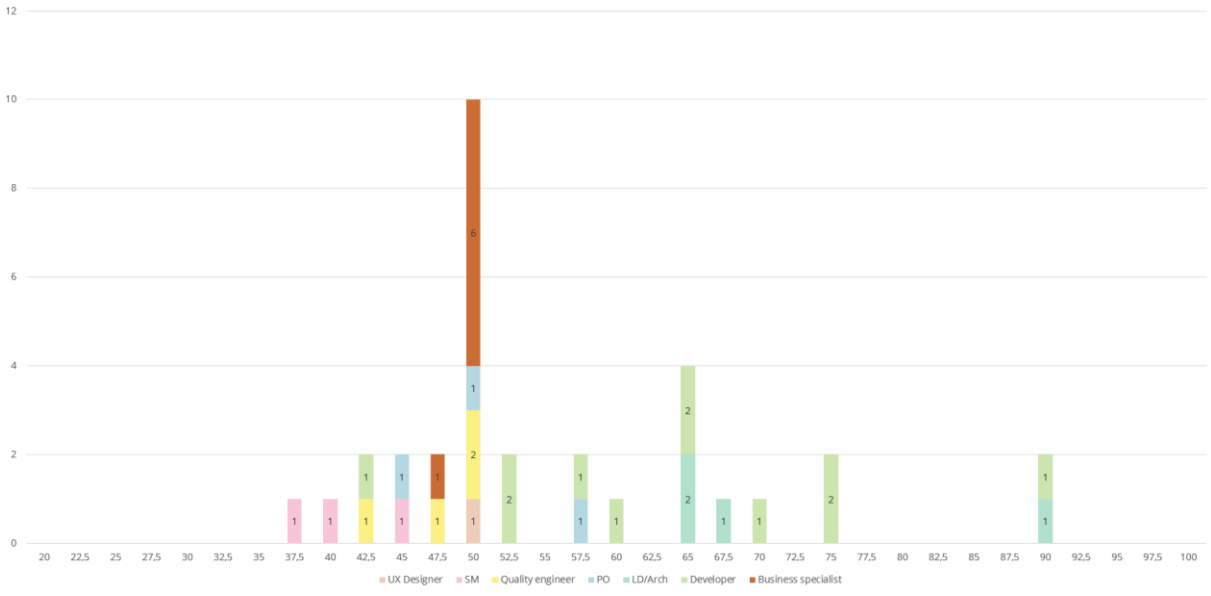

Figure 24. Post-migration SUS score by role.

8.4.2 Score Based on the Release and Deployment Management Qualities

The matrix contains the survey questions from 17 to 32 that measure the system's qualities based on the claimed benefits of Release and Deployment Management in IT literature and it is condensed into a similar score as the SUS score. Each statement is assigned a value from one to five in a similar manner as when measuring the SUS score, “Strongly Disagree” is one point, “Somewhat disagree” is two, “Don’t know” and “Neutral” are three, “Somewhat Agree” is four points and “Strongly Agree” is five points. As all the statements are positive there is not a need to minus anything from the points given. Using this system allows the condensation of the 14-point matrix into one table that can be used to compare the pre-migration and post-migration systems, additionally, the individual scores can be used to get some kind of idea of what aspects of the systems are better than others. The point system is not meant to be scientifically perfect but rather it should help in condensating the data from 14 different statements into one chart. As there are 14 statements that each can give five points at best the maximum score for the system is 70.

Figure 25 shows the calculated scores from the matrix in the pre-migration survey and there are few similarities between the SUS score and the score calculated from the matrix. Three Business Specialists have a similar total score somewhere in the middle and there are similarities between people giving the least points and the most points. In the SUS matrix the “Don’t know” was answered 68% of the time by POs, SMs, UX Designers and Business Specialists, but in this matrix “Don’t know” was answered 52% of the time, which could mean that the questions were more understandable for people in those groups. Neutral on the other hand was responded in both matrices 11% of the time. 52% of “Don’t know” answers could still be considered as a significant amount, if a respondent would answer “Don’t Know” or neutral to all of the statements, they would end up with a score of 42. The average score on the pre-migration system is 43,52. which is quite close to the neutral score of 42. Nevertheless, as Figure 25 shows 17 respondents scored the system under 42 and 19 scored it 42 or more, which also explains the neutral score.

The post-migration system got an average score of 50,06, which is more than what the pre-migration system got. Only four respondents scored it lower than the average score of 42, as can be seen in Figure 26. However, a similar grouping of respondents can be seen in the average score of 42, which is explained again by the “Don’t Know” answers. All of the answerers with the exact score of 42 answered all “Don’t Know” answers except the Quality Engineer who answered all neutral answers.

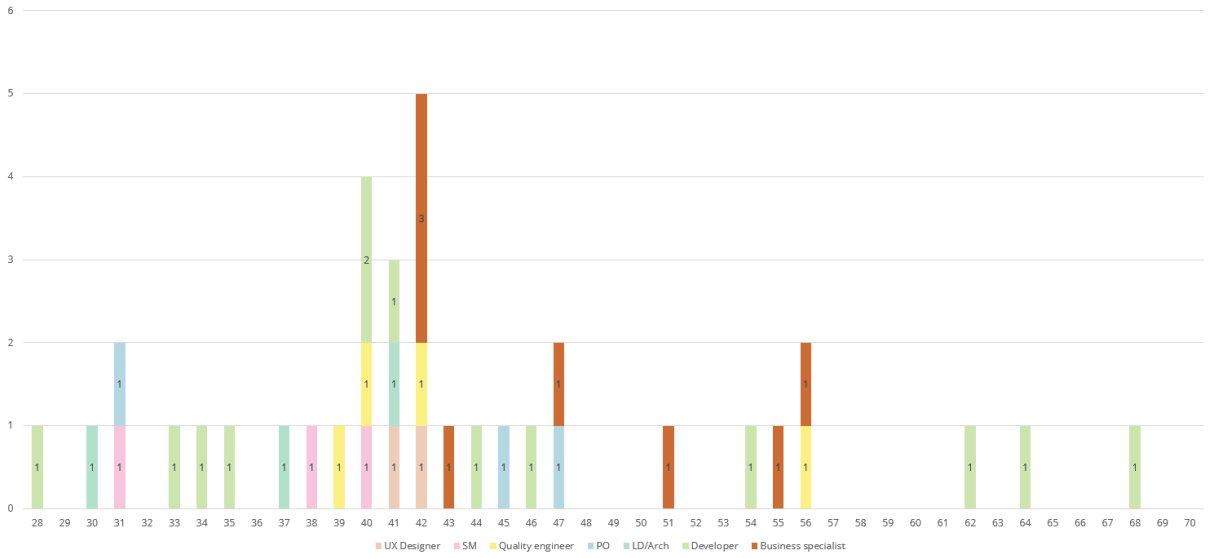

Figure 25. Pre-migration Release and Deployment Management attribute matrix score

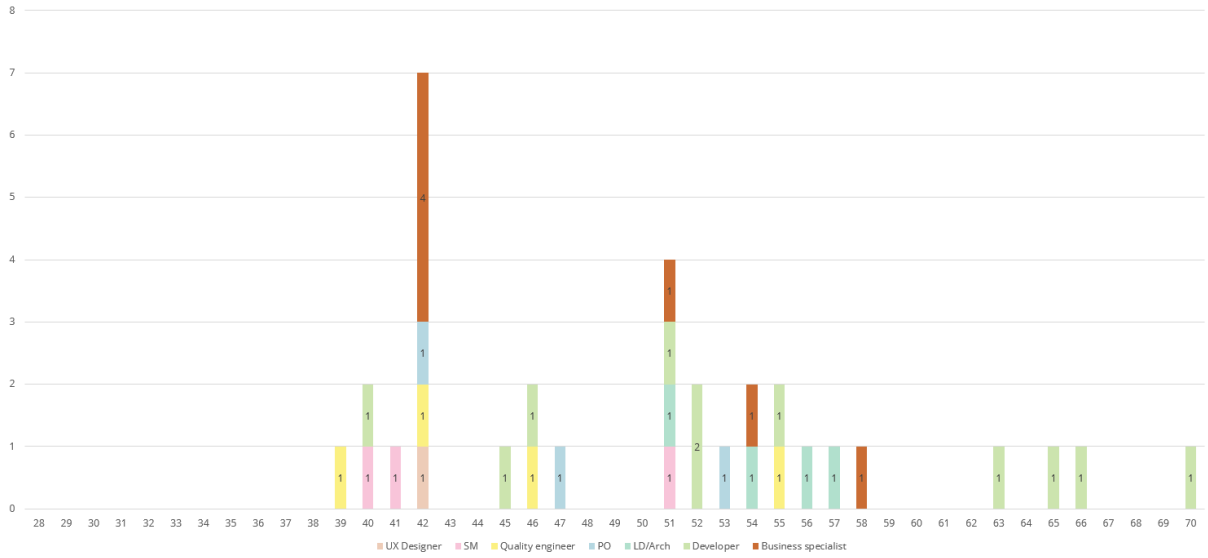

Figure 26. Post-migration Release and Deployment Management attribute matrix score

Table 4 Shows all the statements with their average scores sorted by the change from largest to smallest. The change column can be used to deduce which factor of the system was most improved by the migration. The scoring goes from one to five: the worst score is one, the neutral score is three and the best score is five. Thus, it could be thought that if a statement has a score that is higher than three the respondents on average think positively about it, and the same can then be said that if the score is less than three it is on average seen negatively. As can be seen in the table most of the statements are seen in a positive view on average in the pre-migration system, but there are still scores below three as well. However, in the post-migration scores, none of the fields had a negative score.

The open-question below the matrix that asked for additional comments had nine comments in the pre-migration survey and out of those five were about how the statements were too technical to understand

for their role. Three of these were Business Specialists, one a UX Designer and one a Quality Engineer. One of the remaining four comments could shed some light on the scores given to the statements, as the comment highlighted that the Release and Deployment Management System works ok when everyone is behaving, but that it is too easy to make changes that interrupt the workflow of others, the respondent theorized this could be due to automated testing being slow. Workflow that is easily interrupted could be the reason why in particular "...have a smooth flow of work" had a score less than three. Another respondent felt that it has been stable, but since there have been a lot of new people joining who have not been properly introduced or the past knowledge could struggle. Another respondent thought that they might have given a bit too critical answers as the process has generally worked.

The post-migration survey had seven respondents commenting three of whom thought that it's hard to answer due to their role. Two of them thought that the process was still new and needed more time to get used to. One respondent thought that the process was improved, but that the changes should have been better communicated and another one thought that the new process is more productive and that it saves time. There was also someone who thought that the process was unnecessarily complex.

Table 4 Release and Deployment Management statements and scores

Statement	Change	Pre-migration score	Post-migration score
...deliver value in a replicable way	+0,76	3,03	3,79
...have higher system integrity	+0,71	2,78	3,48
...deliver value in a safe and stable way	+0,67	3,08	3,76
...keep our quality of work high	+0,60	3,25	3,85
...deliver value in a predictable way	+0,56	3,14	3,70
...flexibility when delivering value to our customers	+0,55	3,03	3,58
...have a smooth flow of work	+0,54	2,97	3,52
...quickly deliver value to our customers	+0,48	2,92	3,39
...easily track down error cases	+0,36	2,94	3,30
...easily keep track of changes	+0,35	3,14	3,48
...be more productive	+0,27	3,25	3,52
...easily track the states and environments our tasks currently are in	+0,26	3,19	3,45
...work in a transparent way	+0,22	3,44	3,67
...do less manual work	+0,21	3,36	3,58

8.5 Production Releases

The question "I like the way we do our production releases" has some patterns in the pre-migration survey that can be recognized in the left-side chart in Figure 27. None of the Business Specialists

dislike it while the Developers and LD/Arch seem to be leaning towards disliking it. 20 respondents out of 36 gave a free-form answer and the most common reasons for disliking were it takes a long time, it’s a big thing to do and that there are uncertainties when releasing meaning that unexpected things might go wrong, all of these cons had three answers mentioning them. Other mentioned cons were maintenance pages, rigidness and manual work. On the other hand, the positive aspects of the release process are that it's well-coordinated and that it’s easy to do, both of these were mentioned in two answers. Other pros were that customers won’t notice the deployment because it happens in the early hours of the day and that it's efficient.

The statement was seen more positively in the post-migration survey with only three respondents somewhat disagreeing with the statement as can be seen in the right-side chart in Figure 27. The free-form answers also have changed clearly between the systems. 15 respondents answered the free-form questions and four of them thought that the process has improved, but it’s still a work in progress. This time around the answers changed from production releases are not noticed because they happen so early, so they are not noticed because of the staging slots, this was answered by two persons. One person thought that the naming of the slots is confusing because now the production environment has a slot called “production” and people sometimes just said production during the deployment with the staging slots. One person from the group LD/Arch thought that it is good that also a business verifier has to approve the deployment to the production slot, but one respondent who has to do the business verification thought that this was not good. Some thoughts that were only mentioned once include it is harder to cherry-pick to the release branch, there should more deployments, the teams are more prepared for releases, Azure DevOps is better than the last tools, the tools are better, but the teams should always be able to deploy without maintenance pages.

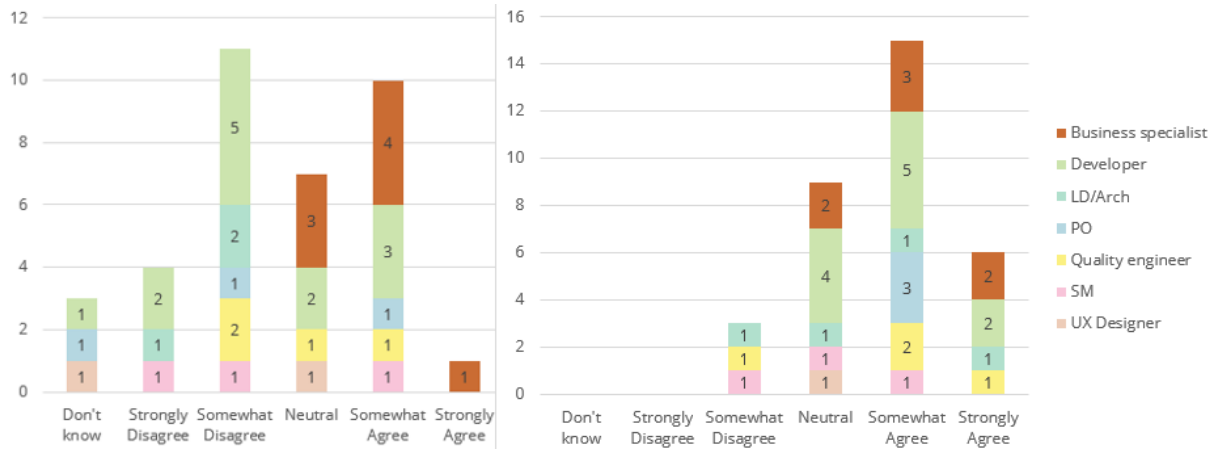

Figure 27. "I like the way we do our production releases.” Pre-migration results are on the left and post-migration results are on the right.

19 out of 36 people think that the amount of production releases there are in the pre-migration process is somewhat sensible and 5 would strongly agree that the amount is sensible, which can be seen in the left-side chart in Figure 28. Out of the 16 free-form answers given six thought that more releases or that releasing when we want would be beneficial. Reasons for this are increased agility, and that value could be delivered quicker. Four persons thought that the releases have a sensible amount of work and that it matches well the amount of work that is done. two persons thought that if a new system could be introduced then more releases could be done.

A similar pattern can be recognized in the post-migration survey free-from answers where 10 out of 12 respondents who answered the free-form answer thought that they would like to see more releases, out of those 10 respondents five thought that it would still require more work and one person thought that the new process would allow for it. Two respondents thought that the number of releases post-migration system is not a big problem. The graph from the post-migration survey would also suggest that even more people want to see more releases than before, since in the pre-migration survey 25 respondents were on the side of agreement and in the post-migration survey 16 were. It is almost like they did not know that there could be more releases, and therefore, were happy with the number of releases.

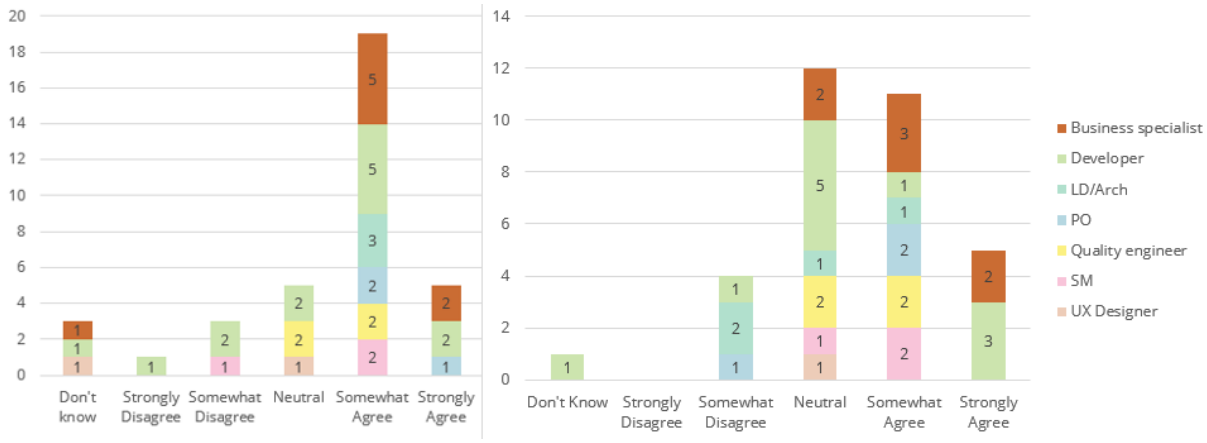

Figure 28. "The amount of production releases we have is sensible." Pre-migration results are on the left and post-migration results are on the right.

The left-side chart in Figure 29 indicates that only two people strongly disagree that the teams are not good at planning what features are in the releases and three people somewhat disagree. This means that the overall feelings are quite positive for the pre-migration process. Three respondents out of eight who gave free-form answers thought that the process is based on time and not features. Their thinking is that whatever is completed is released and no other thought process is behind it. Other comments were that the team size makes communication harder, the plan and communication are both good and

the process is easy to understand. However, another respondent added that sometimes there are also slip-ups and that the process is good on paper, but it leaves no room for mistakes.

A similar effect can be observed with the results to this question as in the previous one, more people seem to view this more negatively, because in the pre-migration survey 20 respondents viewed the topic positively and in the post-migration survey twelve did, even though the planning process was not changed only how it was executed. This may be due to the respondents realizing how much more work and practice it would take to have more releases and how important planning the releases for it is. Five of the 11 respondents who answered the question thought that there is no planning involved or that it is more reactive than actual planning, which is similar to the answers in the pre-migration survey. Two though that the teams still have issues with last-minute decisions and that sometimes features are sneaked into the release. However, two respondents thought that the planning around the release is good.

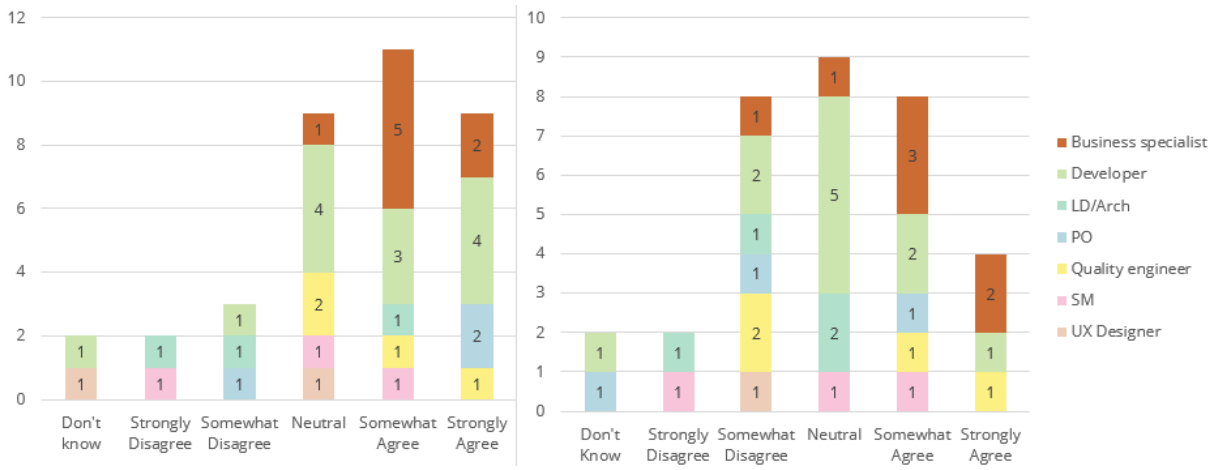

Figure 29. "I think we are good at planning what features we will have in our releases." Pre-migration results are on the left and post-migration results are on the right.

15 out of 36 respondents somewhat agree that is easy and reliable to find what our releases contain while five people strongly agree, this is seen in the left-side chart in Figure 30. Thus, it could be said that the process is seen positively by most people. Reasoning behind this positive attitude can be deduced from the free-form answers, in which the most common positive factor was that the release query contains the information of what is being released or that the communication is good. However, these factors were mentioned by three individuals. While seven people commented that the manual work included in marking the work item's release date is unreliable because of the manual aspect. This means that the release date is sometimes missing. Three people thought that there are good technical tools that can be used to know what is going to be in the release.

The results of the post-migration survey were similar to the pre-migration survey as can be seen in the charts in Figure 30. The answers in the free-form field were similar as well, in which four respondents with more technical roles thought that it is easy to find with the technical tools and two Business Specialists thought that the query is good, in total there were 12 answers. The same risks as in the pre-migration survey were identified by three respondents as well in the manual work of the query. There were some problems with the Git history of the master branch and two persons commented about that. Something new is that two people thought that the new strategy that there is a Pull Request for the release branch makes it easier to find what is in the release.

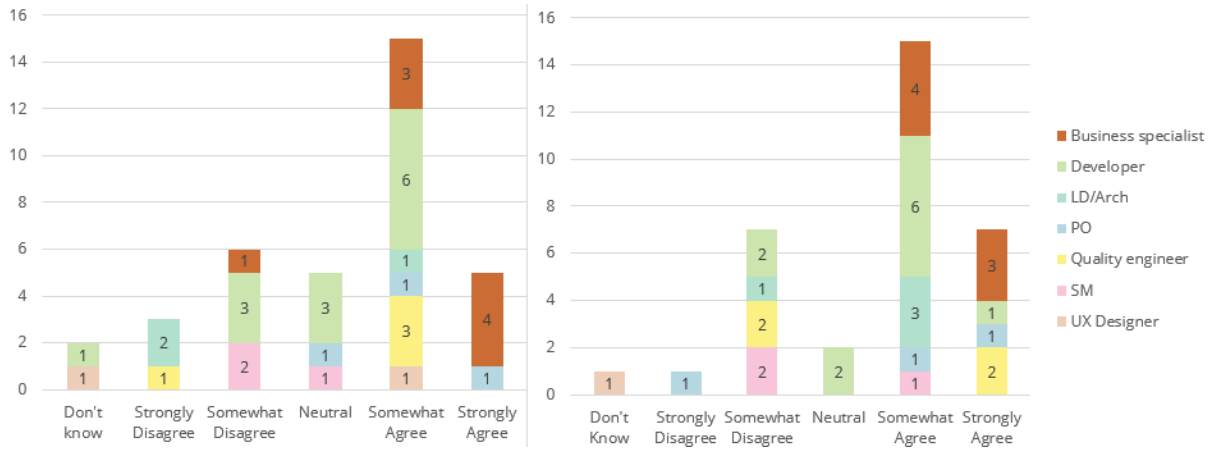

Figure 30. "I think it is easy and reliable to find out what our releases contain." Pre-migration results are on the left and post-migration results are on the right.

10 out of 36 people somewhat disagreed with the statement that the production releases would cause them to rush their work and six people strongly disagreed in the pre-migration survey. However, three out of four Quality Engineers think that it makes them rush their work as seen in the left-side chart in Figure 31. At the same time most of the Developers did not think that it made them rush their work, some light can be shed on this division when the answers to the free-form questions are inspected. Five out of 16 people who answered the question thought that testing time can be short sometimes. Moreover, most of the answers mentioned that especially verification done in the acceptance testing environment is sometimes rushed or that there is a rush to get the work item into acceptance testing. The second most popular answer with four people answering it, is some kind of variant that there is not a need to rush as there are rarely deadlines and if it seems like something is not going to make the next release it can just be released in the one after that. The same is also noticed in open-source projects [17], [18].

The results for the statement were more insightful as fewer people answered “Don’t know” in the post-migration survey than in the post-migration survey as can be seen in the right-side chart in Figure 31.

Additionally, fewer people seem to think that it makes them rush their work, especially among them are the Quality Engineers. The free-form answers reflected this change because only one person mentioned that there might be a rush to test from the 10 answers given. The answer that there is no rush because there are rarely deadlines and that it can just be released in the next release was popular in this survey as well with 5 responses that had that answer.

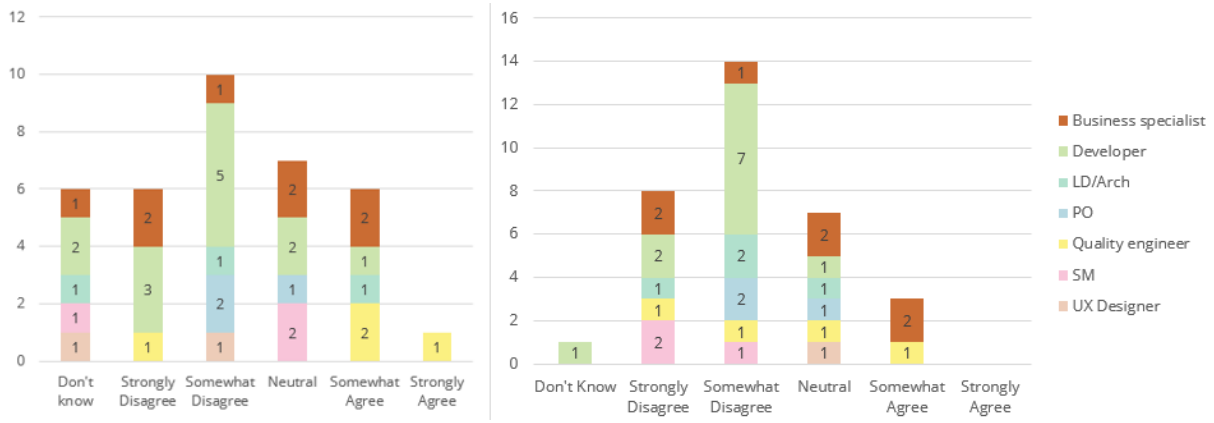

Figure 31. "The way we do production releases makes me rush my work." Pre-migration results are on the left and post-migration results are on the right.

14 out of 36 would somewhat disagree and four would strongly disagree with the statement “we rarely have problems with our releases”, thus it could be said that half of the people disagree with the statement as seen in the left-side chart in Figure 32. A certain somewhat disagreeing mentality is also conveyed in the free-form answers as the most common answer was that sometimes little things go wrong, but there is no need to roll back nor are there major issues, this answer was given by four out of 15 people. Two answers were about how recently there have been more problems and there were another two about how sometimes releases might have to be postponed. The cause of the problems was theorized to be that it is hard to get the branches working together nicely, technical issues and that there are multiple points of failure in the Release and Deployment Management process. All the problems were mentioned only once.

The post-migration survey had fewer people who thought that there were problems during the releases, since in the pre-migration survey 18 respondents disagreed with the statement that there are rarely problems, while 10 respondents disagreed with the statement in the post-migration survey, as can be seen in Figure 32. The increase in positivity is also reflected in the free-form answers where 3 respondents think that there are fewer problems than in the past, another three respondents think that the change has caused a few stumbles but nothing major and another three respondents responded that there are rarely hotfixes, rollbacks. Someone mentioned it is too early to tell since the change was done so recently and one thought that there are still things to be fixed. Overall, this is quite similar to

what was seen in the pre-migration survey, but it could be that the problems are smaller or that they are more accepted because the system is new.

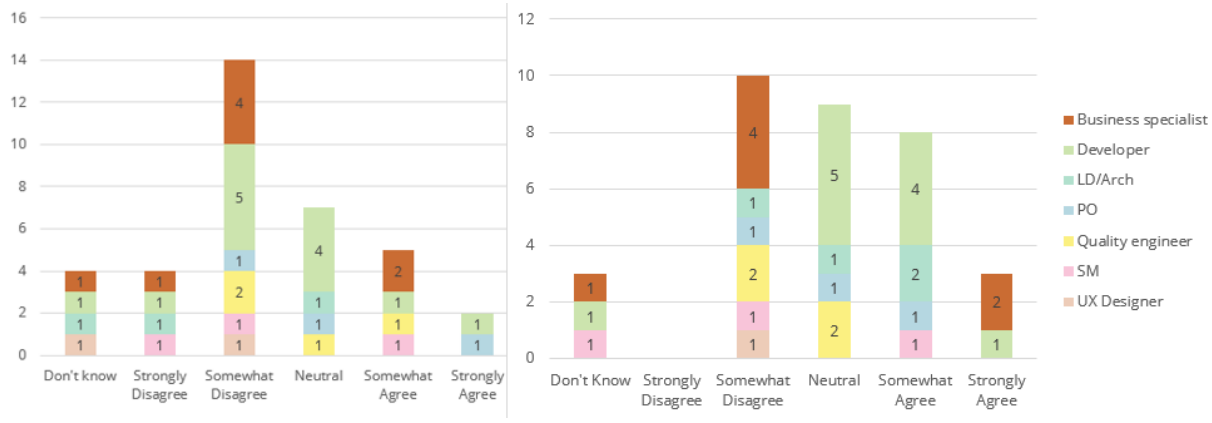

Figure 32. “We rarely have problems with our releases.” Pre-migration results are on the left and post-migration results are on the right.

Most people are neutral about the stressfulness of the releases in the pre-migration system with 12 people answering it out of 36. The rest of the votes were divided rather evenly with just three people more on the side of disagreement as is portrayed in the left-side chart in Figure 33. Even though most people did not think that the releases were stressful five out of 15 free-form answers are about how the release is stressful for the Developer who is doing the release. Some of the reasons for this is that since the release happens so early in the morning the releasing Developer lacks the support of the whole team and there are multiple things that could go wrong. Additionally, one of the respondents thought that if something does go wrong there is pressure to fix the situation quickly as downtime costs the company money. Two respondents mentioned that it is not stressful for them because they are not responsible for releasing.

The stress levels are decreased by the post-migration system given that seven fewer people agreed with the statement, as can be seen in Figure 33. The free-form answers have a lot of different opinions, but there were similar opinions in the post-migration survey as in the pre-migration survey. This includes such opinions as it is not stressful, because it’s not part of someone’s job, the Developer whose responsibility it is has a lot on their plate and that releasing in the morning is stressful. Other factors that cause stress are CMS changes and the fact that the process is new. Stress relieving factors are staging slots and the call during the production deployment morning.

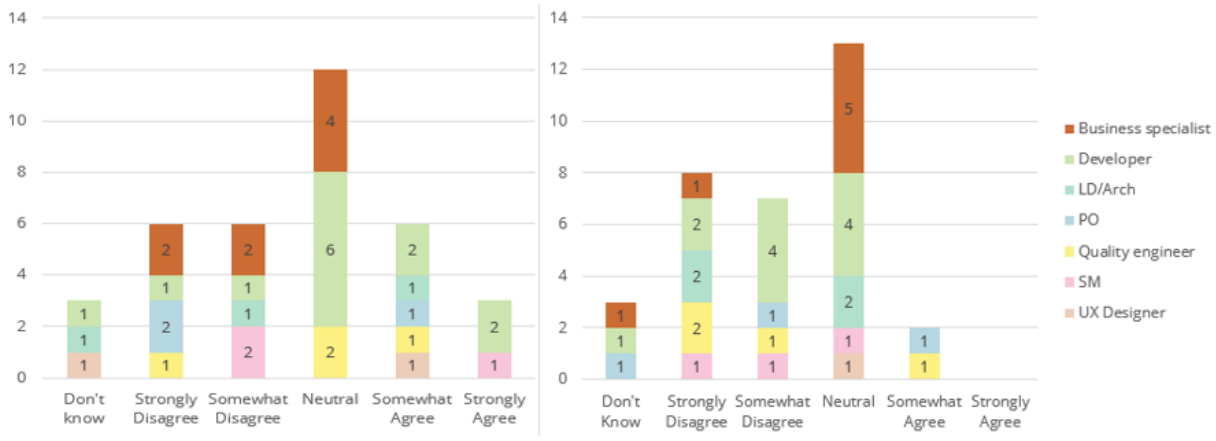

Figure 33. "Our production releases are stressful." Pre-migration results are on the left and post-migration results are on the right.

Most people don't know or are of neutral opinion on the matter "It is easy to recover from failures when doing production releases." and there is one person more on the side of agreement, as seen in the left-side chart in Figure 34. Three out of the 11 who answered the free-form field thought that diagnosing the problem is hard and two of those people thought that it is hard because there are so many things that could go wrong. Two respondents thought feature toggles help in recovering from failure and two thought that there is rarely a need to rollback.

The feelings towards the failure recovery were quite the same, meaning that most of the respondents answered "Don't know" or neutral on the matter. The mentality could be explained in the free-form answers as four out of eight respondents who answered the question thought that there have not been such problems that would measure this. Two thought that there are fail-safes for possible faults and another person thought that it would depend on the problem.

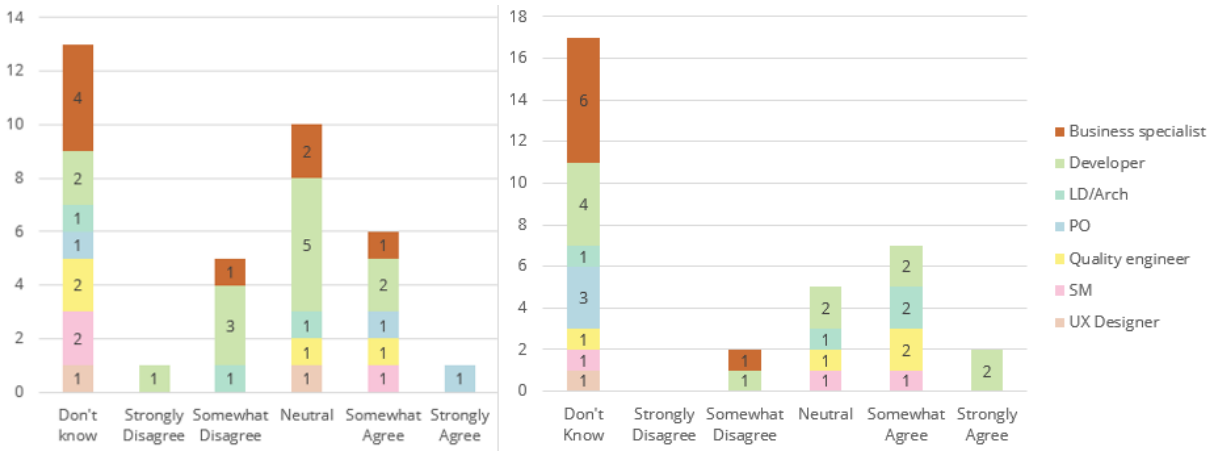

Figure 34. "It is easy to recover from failures when doing production releases." Pre-migration results are on the left and post-migration results are on the right.

11 out of 36 respondents somewhat disagreed and five strongly disagreed with the statement that the releases require a lot of effort in the pre-migration system, while only seven somewhat agreed and out of those seven six were Developers as can be seen in the left-side chart in Figure 35. Without the Developers voting for somewhat agree there would have only been one person thinking that it does not require a lot of effort. The free-form answers by the Developers who voted Somewhat Agree suggest that the Developer part in the release could be thought of as somewhat easy. Two Developers answered the free-form question and one of them thought that the procedure is simple as it is just pressing a few buttons, while the other one thought that with few changes it's easy but when there are more it might get complicated. Nevertheless, Six out of the eleven people who gave a free-form answer thought that it takes a lot of effort, some of these people thought that more could be automated, some thought the release itself is a big ceremony, or that there are a lot of chores to do to release.

Almost a similar number of respondents think that the releases don't require a lot of effort and fewer people think that it does as the biggest group answered neutral instead of somewhat disagree in the post-migration survey, as seen in Figure 35. Five out of eight persons who gave a free-form answer thought that it requires a lot of effort. Reasons for that were that the morning releases take effort, the system is still evolving, there is a learning curve and a lot of effort is required prior to the release. One respondent thought that the effort required could be lowered by using the staging slots to deploy without maintenance pages.

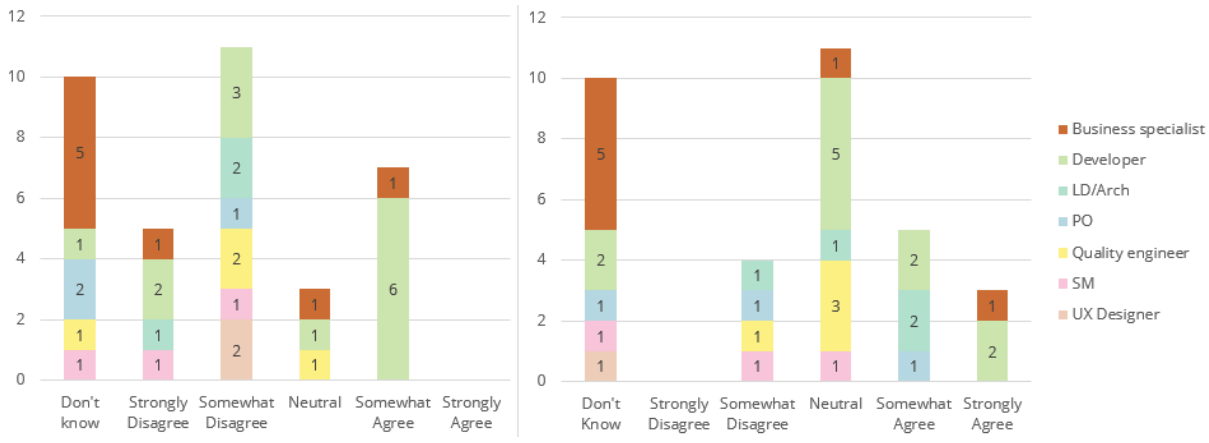

Figure 35. "It does not require a lot of effort to do production releases" Pre-migration results are on the left and post-migration results are on the right.

A slight majority thought that the releases were not seamless for the customers in the pre-migration system as one person more somewhat disagreed than somewhat agreed and one person more strongly disagreed than strongly agreed as seen in the left-side chart in Figure 36. On the free-form answers, it could be said that everyone acknowledges that the service has downtime during deployments, but the

relation to that downtime varies from respondent to respondent. Six persons out of 16 think that the customers do not notice the downtime, some think this is because the releases happen during the early morning. Six people out of 16 mention maintenance pages and out of those people four persons strongly disagree with the statement that the releases are seamless, one somewhat disagrees, and one somewhat agrees. Two of those persons strongly disagreeing are Product Owners and it can also be noticed that all the POs think that the process is not seamless as there is downtime during the releases, which is caused by the maintenance pages. Another negative mentioned by two respondents is that even though the release happens early there are still customers already using the service, which is not optimal. Two respondents think that the process is seamless as long as it works.

When the charts in Figure 36 are inspected it can be noticed that the post-migration system seems to be more seamless for the customers than the pre-migration system, as 15 respondents disagreed with the statement in the pre-migration system and six respondents disagreed with the statement in the post-migration system. Six out of 12 respondents who gave a free-form answer thought that they are seamless because there is an option to deploy without maintenance pages. Two respondents thought that maintenance pages are still needed, and two other respondents thought that they are inevitably needed in some cases when dependencies have deployments. One respondent thought that the staging slot release is not completely seamless as it causes slowness for the customers.

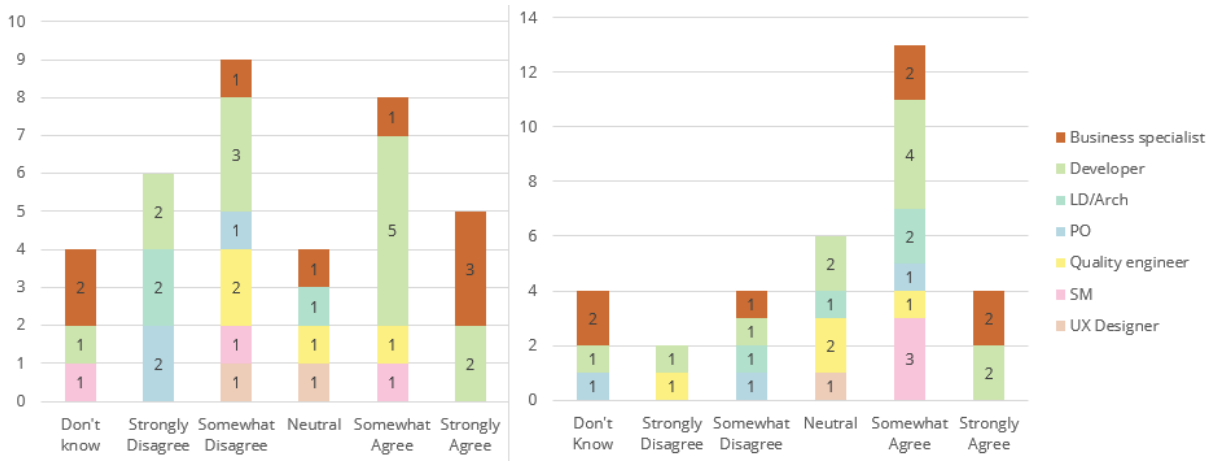

Figure 36. "Our production releases are seamless for our customers." Pre-migration results are on the left and post-migration results are on the right.

8.6 Agile and Release and Deployment Management

Figure 37 Displays the results for the “Our Release and Deployment management” matrix for the pre-migration system and as can be seen from the Figure there were more positive than negative answers. That being said, most people were neutral or did not know if the Release and Deployment Management process made it easier for them to follow Scrum. Four respondents had additional

thoughts on the topic and two of them were of the opinion that the pre-migration Release and Deployment Management is not agile or that it hinders agility. The reasons were bottlenecks and the process is slow. One of the answers is about how the respondent is unsure how Release and Deployment Management affects Scrum, and this answer could be one of the reasons why there were so many neutral answers given. Albeit, in the interviews all of the interviewees identified the relationship between Scrum and Release and Deployment Management.

There is a slight increase in positivity in the statements regarding Agile and the post-migration system when it is compared to the pre-migration system. This can be seen by comparing the post-migration chart in Figure 38 with the pre-migration chart in Figure 37. The charts for statements "…supports our way of working in Scrum." and "…makes it easier for me to follow Scrum." have more strongly agree answers and fewer respondents answering neutral or disagree. The chart that does not have a clear increase in positivity is the "…is well integrated into our agile way of working." chart, in which there is one more somewhat disagree answer and the same number of neutral answers. Similar answers were given in the free-form answers, two persons thought that the process itself is heavy and slow, and one person thought that the way testing is done is waterfall-like, which is a similar answer to the bottleneck answer. However, two persons had answers that were not seen in the pre-migration survey, which were that the process could support more but the teams are not utilizing the capabilities. Nevertheless, it could be argued that on average the respondents thought that the post-migration process supports the ways of working in Scrum better. Albeit the process is not seen as an improvement regarding agility, which could be because the process is quite similar to the pre-migration process and only the execution of it was changed.

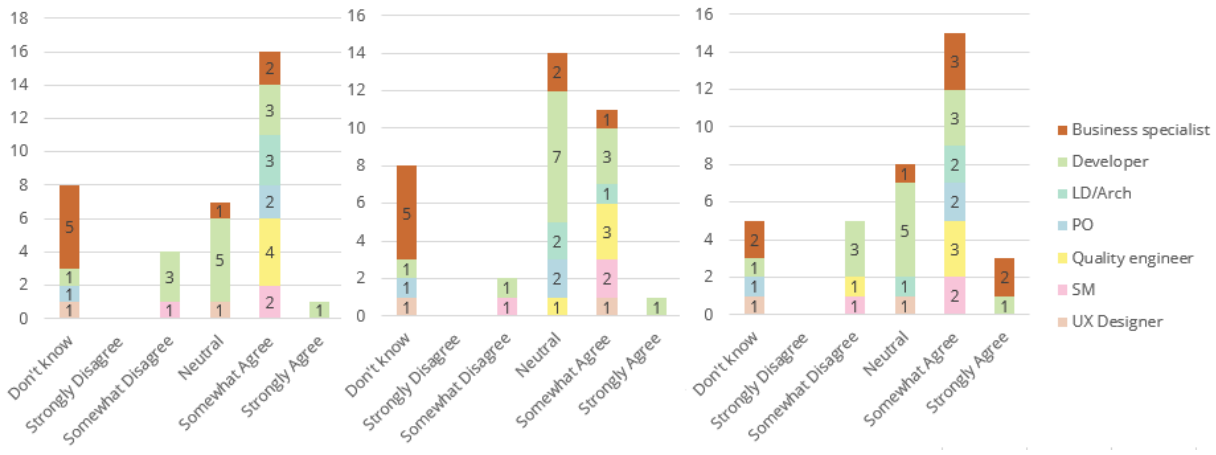

Figure 37. Pre-migration results for “Our Release and Deployment Management process..." matrix: Left "…supports our way of working in Scrum.". Middle "…makes it easier for me to follow Scrum.". Right "…is well integrated into our agile way of working."

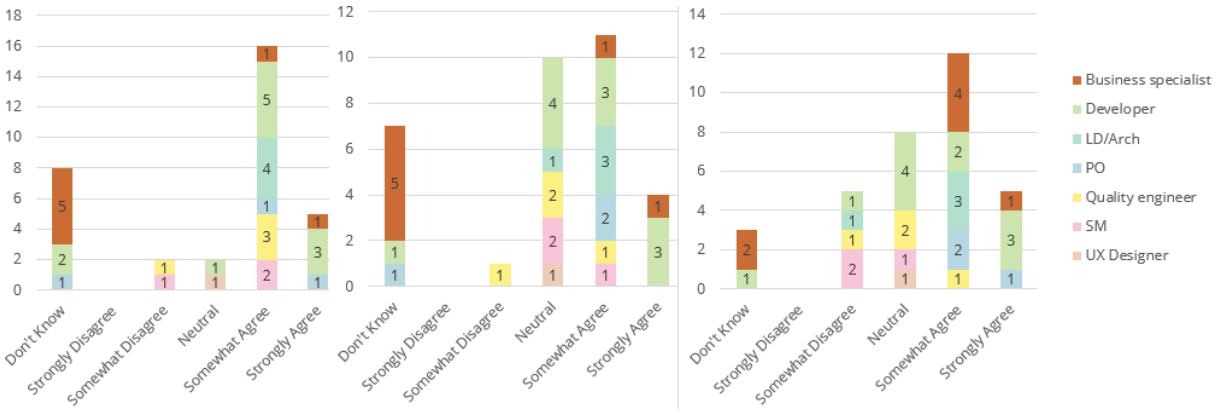

Figure 38. Post-migration results for “Our Release and Deployment Management process..." matrix: Left "…supports our way of working in Scrum.". Middle "…makes it easier for me to follow Scrum.". Right "…is well integrated into our agile way of working."

The majority of people are of neutral opinion when met with the statement "I think our release and deployment management supports my way of working." in the pre-migration system, as can be seen in Figure 37. This can be explained in the free-form questions to which 11 persons answered, as the most common answers with three persons answering both were that it works fine but it could be better, and that automation could reduce the amount of manual labour. Both of these answers were given only by Developers or respondents from the group LD/Arch. Moreover, a Developer thought that releasing is full of chores, which is not optimal. A Quality Engineer thought that there are sometimes testing bottlenecks. Two Product Owners and one Business Specialist thought that they could not answer the questions as they are not working with it or that not working hands-on with it.

Better feedback is given for the post-migration survey as there are fewer people thinking that it does not support their way of working and more people thinking that it does, as can be seen in

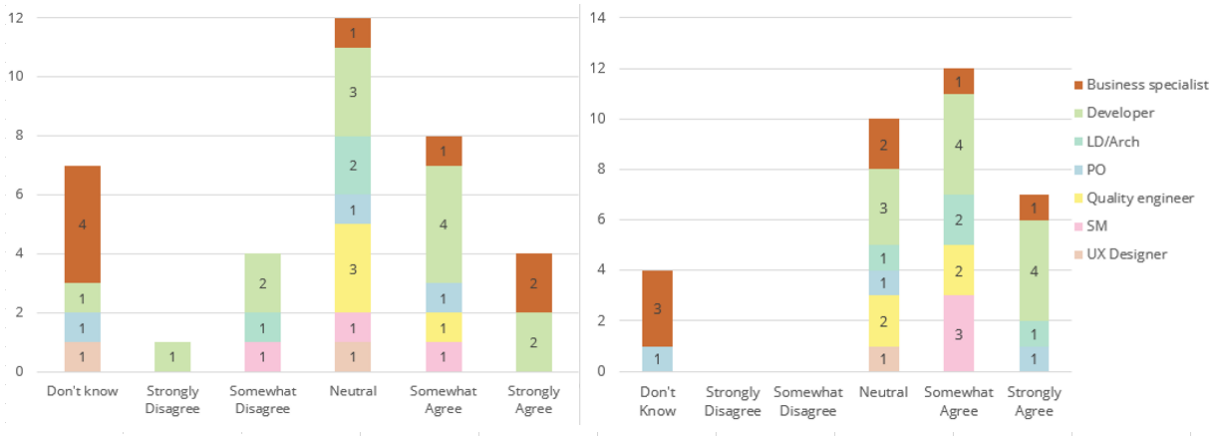

Figure 39. The free-form feedback was also better, but there were still six persons out of 11 who answered the question and thought that there could be more fine-tuning, more flexibility and that the system is not fully utilized. One respondent thought that the quality of deployments is better and

another one thought that there is a clear schedule and that they are easier to plan. A Developer thought that the Azure deployment process is easy and straightforward to follow. Three Business Specialists answered the free-form question and one of them thought that they are not using the system. However, two Product Owners this time wished for more flexibility instead of thinking that they are not working with the system.

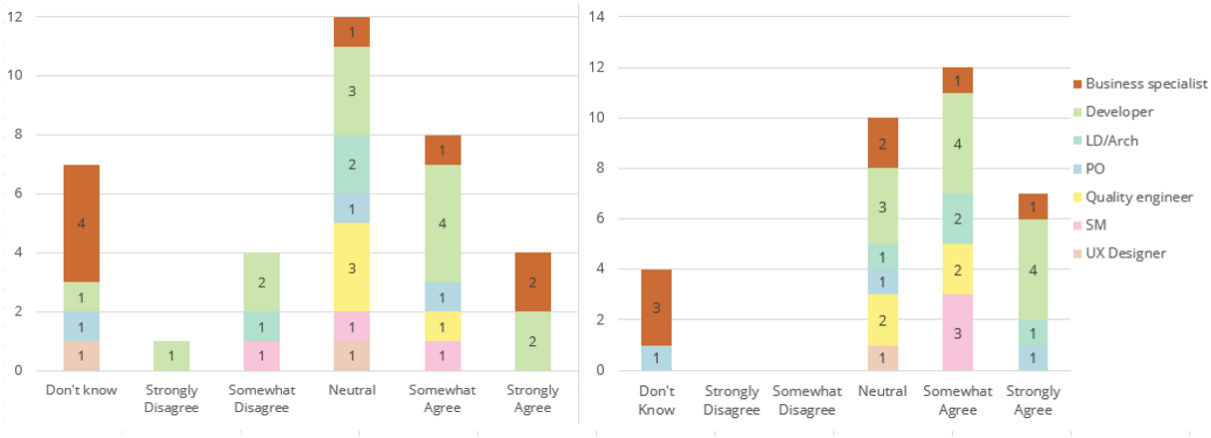

Figure 39. "I think our release and deployment management supports my way of working." Pre-migration results are on the left and post-migration results are on the right.

8.7 Feature Toggles and CMS in Release and Deployment Management

Most respondents are of the opinion that the teams are good at using feature toggles with 17 out of 36 respondents somewhat agreeing and five strongly agreeing with the statement, as seen in the left-side chart in Figure 40. The product owners had somewhat similar opinions on the matter as 2 of them strongly agreed and one somewhat agreed and all of them mentioned that new functionality is usually behind a toggle in the free-form answers. Two respondents of out 12 who gave a free-form answer mentioned that old ones are rarely removed. Apart from these two opinions, everyone else seemed to have almost unique opinions on the matter, but overall, there were more positive than negative thoughts about them, which is akin to what was noticed in the interviews.

The opinions on whether the teams are good at using feature toggles or not have not changed between the pre-migration and post-migration systems, which is not surprising because that process was not changed. There are more strongly agree answers, but there are fewer somewhat agree answers and more neutral answers in the post-migration survey as seen in Figure 40. Therefore, it could be said that the overall feeling regarding feature toggles has remained the same. The free-form answers were more negative than previously only one respondent out of 11 respondents who gave a free-form answer had a positive comment on the topic. The most common answer with five people giving it was that the

teams have bad practices with feature toggles. One explanation given by a Developer was that sometimes feature toggles are added just in case and that they just make the codebase messier, while a Quality Engineer thought that the bad practices in handling feature toggles before they go into production, which could leak defects into the production environment. The second most common answer with three persons answering it is that the toggles are not maintained.

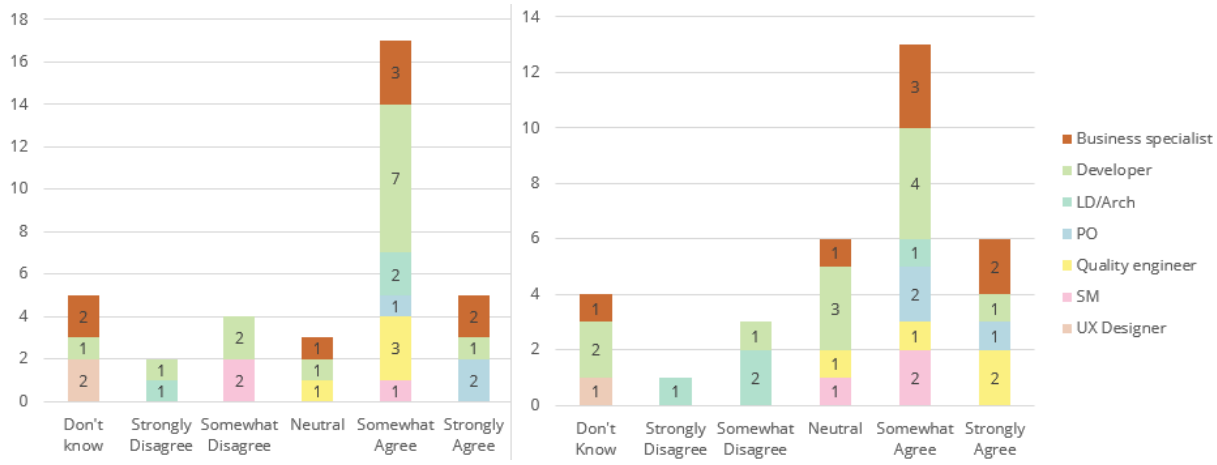

Figure 40." I think we are good at using feature toggles." Pre-migration results are on the left and post-migration results are on the right.

12 people out of 36 strongly agree that feature toggles are an integral part of the teams' Release and Deployment Management, and the same amount of people also somewhat agree, meaning that the majority thinks that they are an integral part as seen in the left-side chart in Figure 41. Reasonings for these answers can be found in the free-form answers, in which three out of nine respondents thought that Feature Toggles give the flexibility to deploy a feature without it being ready and the deployment being about it. Moreover, two respondents thought that they could be used to deploy without worrying about dependencies being ready, another two persons thought that they could be used as a failsafe and another two persons thought that they are useful. One respondent did mention that not everything can be toggled.

The same number of respondents (22) agree with the statement that feature toggles are an integral part of the teams' Release and Deployment Management in the pre-migration and post-migration systems. However, one difference is that in the post-migration survey, there are 14 respondents who strongly agree with the statement, but in the pre-migration system only 12 did. Nine persons gave free-form answers which were all unique, but they did have some same themes. Three respondents thought that they are essential either because of dependencies, turning features on and off or that they are essential for being agile. Other positive comments included they give stability, allow earlier releases and

sometimes they are needed because it takes a long time to turn on a feature. Negative aspects of them were their manual aspect, lack of testing and that they are not well implemented.

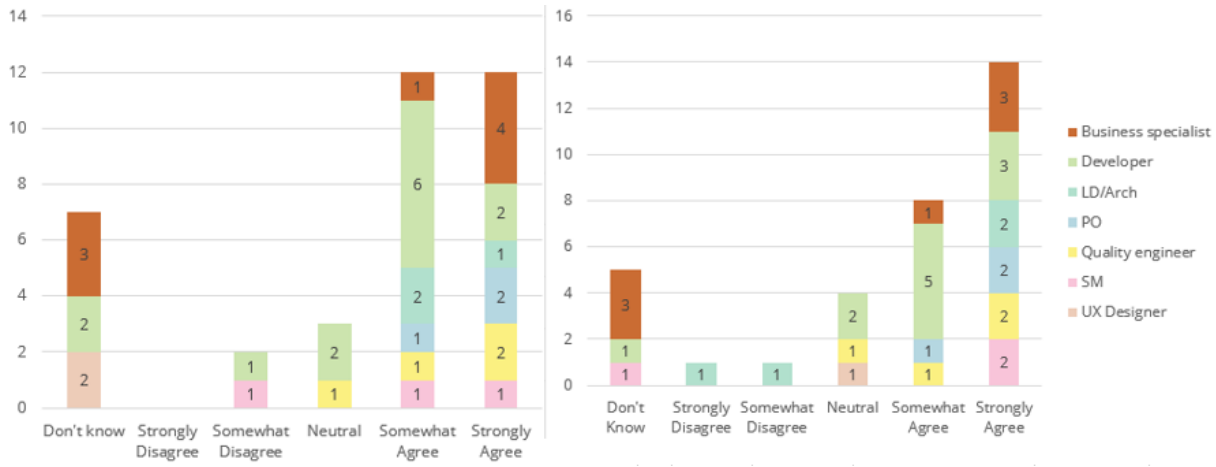

Figure 41. "I think feature toggles are an integral part of our Release and Deployment Management." Pre-migration results are on the left and post-migration results are on the right.

The most common answer to the statement "I think Content Management hinders our Release and Deployment Management." is “Don’t Know”, as seen in the left-side chart in Figure 42. Apart from these, there seems to be somewhat of a division between Business Specialists and product owners thinking that it does not and the majority of Developers and LD/Arch thinking that it does. The division is made more understandable when the free-form answers are inspected. The five respondents who disagreed with the statement and left a free-form answer were of the opinion that the act of content management does not interfere with their way of working, that it is necessary for a web page to work, or that the content managers themselves are useful. On the other hand, the seven respondents who thought that it hinders Release and Deployment Management and left a free-form answer were thinking about content lifting and content exporting except one of them whose point is about problems with CMS in general. The hindrance that these persons were thinking about was that content lifting and exporting is awkward, it introduces a manual aspect, and it may be difficult to fully automate the process with it and it is a point of failure that requires communication and documentation.

The opinions regarding whether content management hinders Release and Deployment Management did not move strongly in any direction, as can be seen in Figure 42. The answers to the free-form question were similar in the post-migration survey as in the pre-migration one respondents who thought it hinders the process were thinking about the system and those who did not think so thought about the Content Managers. The most popular answer with three out of nine persons answering it was

that the Content Management System that is used is unreliable. Two other respondents thought that seamless or continuous deployments are made harder by content management. Additionally, one person thought that the manual aspect that is added by the verification of the content lift is a hindrance. Moreover, a respondent thought that the content management responsibilities are unclear as a lot of the content is actually static and could be added during development. It could be argued that the free-form answers from the surveys combined reached the same conclusion that is met in the Content Management Systems chapter; content management is a necessary manual step for a flexible web page.

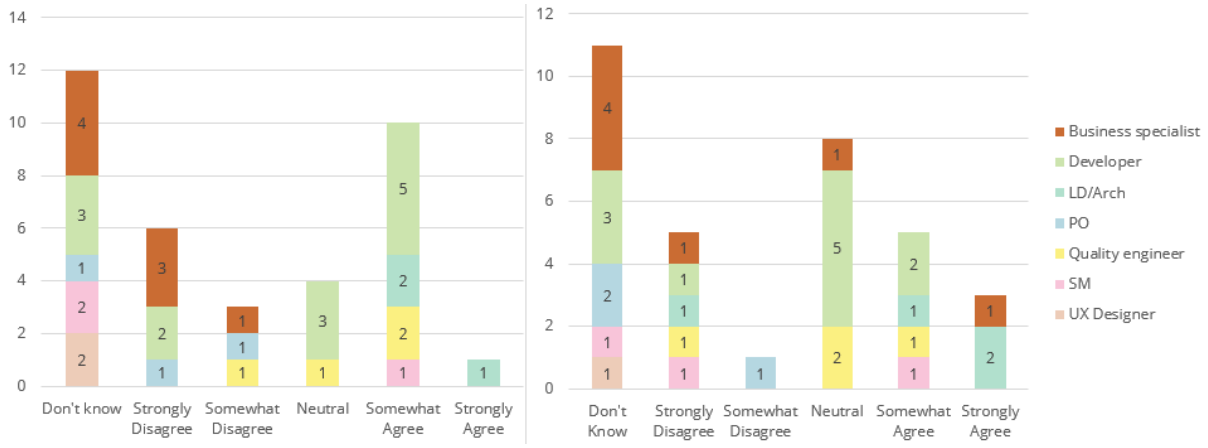

Figure 42. "I think Content Management hinders our Release and Deployment Management." Pre-migration results are on the left and post-migration results are on the right.

8.8 Release and Deployment Management Tools

The pre-migration survey results for the matrix “I think our Release and Deployment Management tools are...” are displayed in Figure 43. Pre-migration survey results for “I think our Release and Deployment Management tools are...” matrix. Left “...easy to use”. Second from Left “...offer valuable knowledge”. Second from right “...up to date”. Right “...fully utilized” s Figure 43. From the figure, it can be noticed that most of the Business Specialists, UX Designers, and Product Owners answered “Don’t Know” most of the time, which could be due to them not directly working with the tools. According to the chart most people who did not answer “Don’t Know” were neutral or somewhat agreed to the statement that the tools are easy to use. These answers follow a similar pattern as the previous answers about releasing where some thought that deploying is easy because it is just pressing a few buttons. Most respondents were neutral about their stance on the statement that the tools would offer valuable knowledge with only six out of 36 respondents responding something else than “Don’t Know” or neutral. However, this neutrality is not carried over to the statement about the tools being up to date to which most respondents who did not answer “Don’t Know” disagreed with the statement.

The neutrality was repeated again in the statement about whether the tools are fully utilized, and the free-form answers do not help in this matter either. Instead. two out of seven respondents who left a free-form answer thought that Azure DevOps should be used more, and one respondent thought that it's not good that the tools are scattered among multiple platforms. Two other respondents thought that the tools should be updated or that this could be where some of the technical debt is located.

The post-migration results for the statement "...easy to use" have a few more respondents thinking that the post-migration tools are easier to use. This can be seen by comparing Figure 44 with the pre-migration Figure 44. Similarly, the respondents think that the tools are slightly more utilized. However, the respondents seem to think that they are much better at offering valuable knowledge and that they are more up-to-date. Three free-form answers were given: one of them thought that the tools are useful if you are savvy with them, one thought that they have been improved but that they could be improved even more, and one thought that Azure DevOps is much better than TeamCity and Octopus and that it is the industry standard.

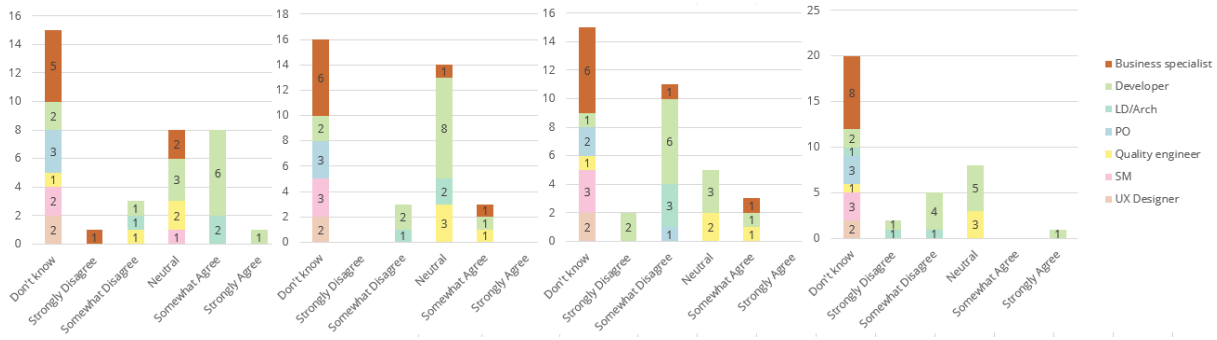

Figure 43. Pre-migration survey results for "I think our Release and Deployment Management tools are..." matrix. Left "...easy to use". Second from Left "...offer valuable knowledge". Second from right "...up to date". Right "...fully utilized"

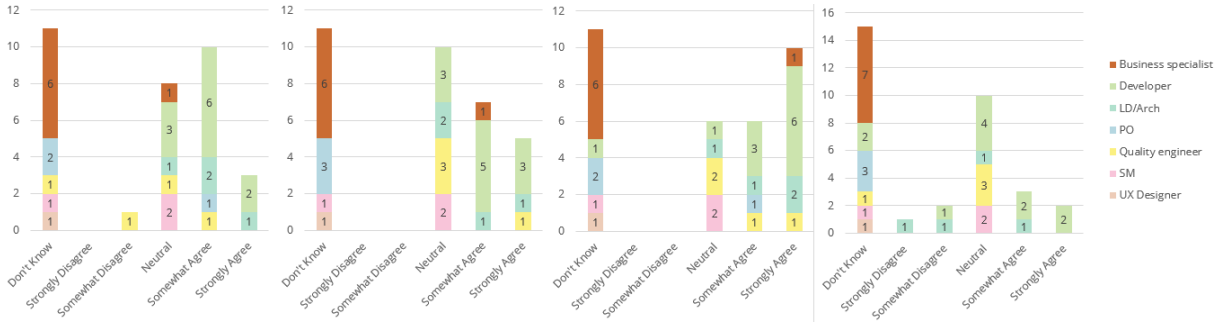

Figure 44. Post-migration survey results for "I think our Release and Deployment Management tools are..." matrix. Left "...easy to use". Second from Left "...offer valuable knowledge". Second from right "...up to date". Right "...fully utilized"

The respondents were somewhat divided on whether there are good levels of automation in place in the pre-migration process or not, as can be seen in the left-side chart in Figure 45. The free-form answers reveal that the respondents are not entirely satisfied with the levels of automation and that it could be improved. All of the six answers given acknowledge that automation exists, but also that there could be more of it or that the automation is currently being improved. Three out of these respondents are talking about test automation two would like to see more testing automation and one would like the fix the existing flaky tests. Three respondents commented about the automation not working or that it is not verified if it works properly or that the automation is immature because of bottlenecks.

The automation can be said to have improved in the post-migration process, as there are fewer people disagreeing with the statement and more agreeing than before, this can be seen by comparing the charts in Figure 45. This could be attributed to the fixes in the feature tests since in the pre-migration survey most of the free-form answers were about test automation, but in the post-migration survey, none were. Instead, Five out of six respondents thought the automation should be improved or that it should be utilized even more. However, one respondent thinks that automation has a downside, because "…automation often leads into people not learning what and why things are happening."

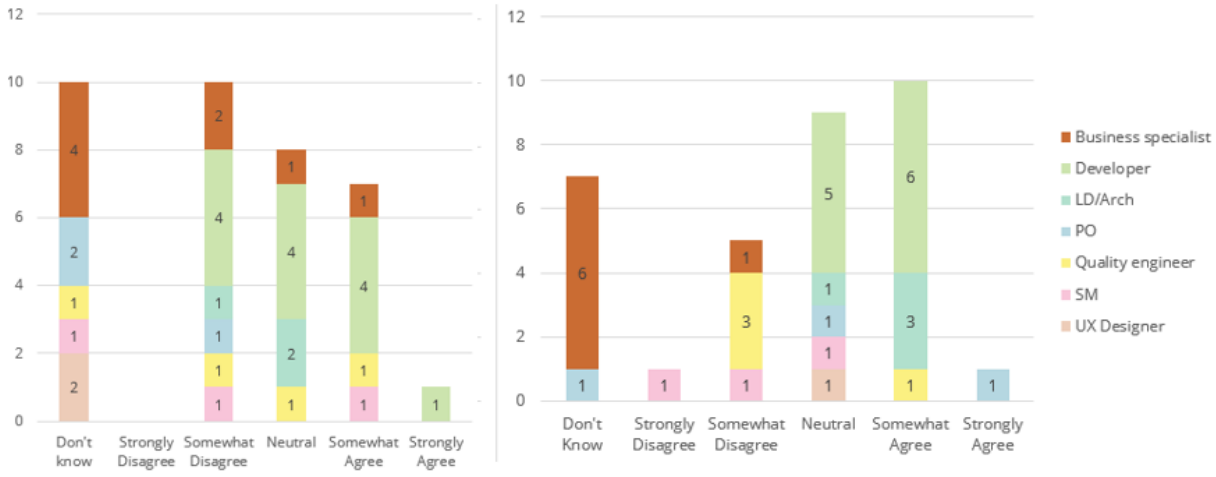

Figure 45. "I think we have good levels of automation." Pre-migration results are on the left and post-migration results are on the right.

The respondents are mostly neutral on the topic of automation reliability in the pre-migration system, even apart from the neutral questions the respondents are divided almost evenly between agreeing and disagreeing with eight respondents agreeing and seven disagreeing, as seen in the left-side chart in Figure 46. Five out of eight free-form answers mention flaky feature tests and one respondent mentions failing quality gates, this would indicate that the biggest problem with the quality of the automation is the feature tests that are flaking. Additionally, one respondent calls the automation

unreliable and that it hinders the ability to deploy. There was one positive comment that was about the production downtime being low.

The responses were quite similar in the post-migration survey with the same number of people agreeing with the statement overall, but there are fewer people disagreeing with the statement. Most responses in the free-form field are steps in which the automation does not work, this includes steps such as content lifting is sometimes unreliable, test automation problems still exist, and Azure DevOps board automation has some issues. Other thoughts were that more monitoring of failures would be beneficial, feature tests should be mandatory to run, and automated processes are done wrong.

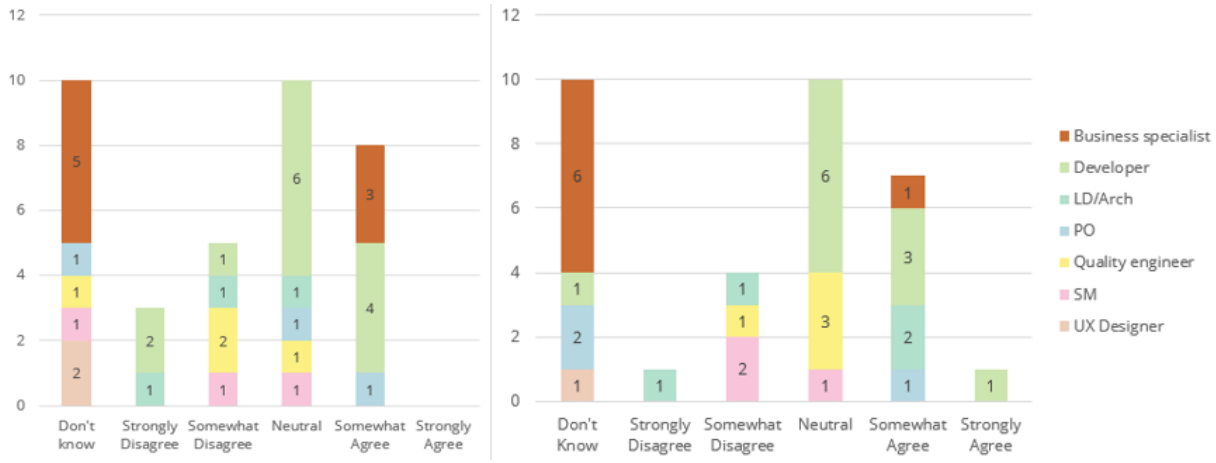

Figure 46."I think our automation is reliable." Pre-migration results are on the left and post-migration results are on the right.

8.9 Opinions on the Release and Deployment Management Systems

Eleven persons out of 36 somewhat agree that the pre-migration Release and Deployment Management process should be modified and another eleven strongly agree with the statement, which can be seen in the left-side chart in Figure 47. A pattern can be seen in Figure 47. "We should modify our release and deployment management process." the chart, as all of the people in the group LD/Arch strongly agreed with the statement and the majority of the Developers would like to change the process as well. The six persons who answered neutral could be considered as being content with the pre-migration process, which means that it could be said that 16,67 per cent of the respondents would not change the pre-migration process. Changes the respondents would like to see and reasons for their stance on the topic can be found in the free-form answers. Only people who would like to see the process changed gave a free-form answer with all the answers thinking that there are problems with the pre-migration process and that it could be improved. The most popular answer apart from that the process should be improved is that they would like to see more automation, this answer was given by

two persons out of eight respondents who gave a free-form answer. Other wished changes include fixing the master branch, being able to do country-specific releases, smaller deploys, and pipelines.

In the post-migration survey, 13 out of 33 respondents somewhat disagree or were neutral about the statement if the process should be changed, which means that 39,39 per cent of the respondents are content with the current process. However, 10 persons still disagree with the statement. Therefore, 30,30 per cent of the respondents would still like to see the process changed. Six out of 10 free-form answers were about how the process should always be improved and one person thought that it is hard to say because the process is still so new. Improvements that were mentioned include less manual work, creating the release branch should include less manual work and analysis, manual review steps should be enforced or removed, no more morning releases and there could be more test automation.

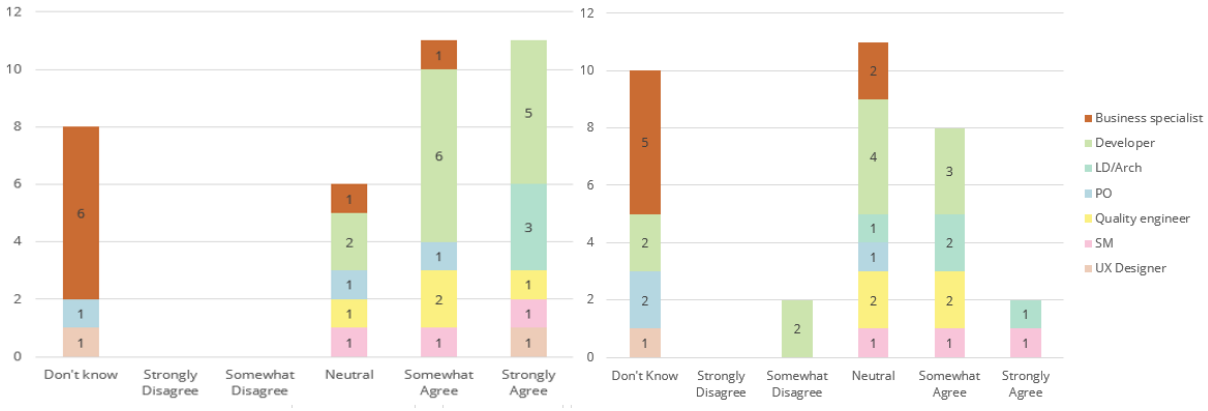

Figure 47. "We should modify our release and deployment management process." Pre-migration results are on the left and post-migration results are on the right.

Once again all of the respondents in the group LD/Arch thought negatively of the pre-migration process as all of them strongly disagreed with the statement that they would recommend the Release and Deployment Management process, as can be seen in the left-side chart in Figure 48. A similar negativity is seen with Developers because the majority of them would not recommend the process either. However, the general tone seems to be somewhat more neutral as nine answered with “Don’t Know” and twelve were neutral. Seven free-form answers were given and out of those seven answers two could recommend parts of it, and two answered that the previous answers were the reasons for their answer to this question. The majority answered that they would not recommend the process.

The negativity has completely disappeared from the results of the post-migration survey as there is not anyone who would disagree with the statement that they would recommend the process, as seen in Figure 48. Some explanations can be found in the free-form answers to which five respondents

answered that using Azure DevOps has been nice, it is good for legacy systems, it is safe and transparent, it could be improved, but it works well and some parts could be recommended.

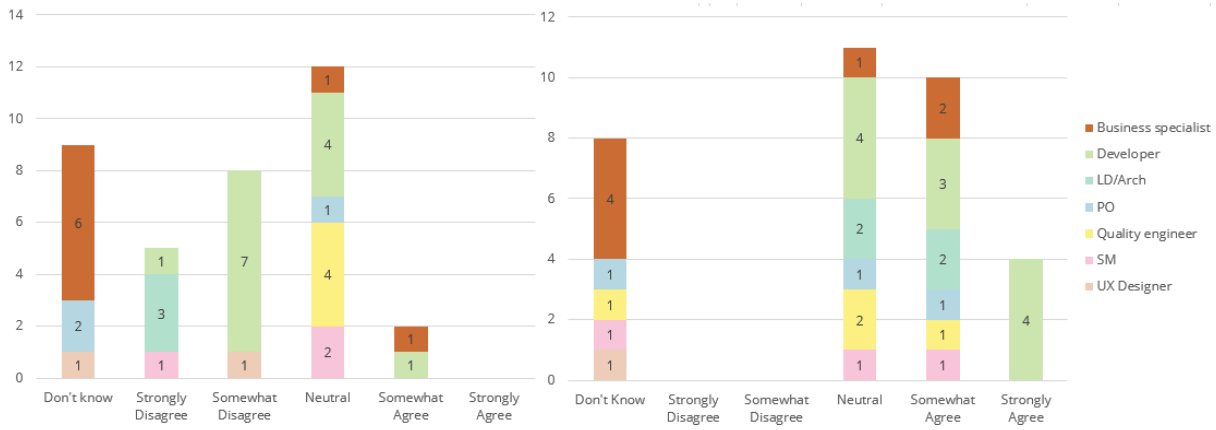

Figure 48. I would recommend our release and deployment management process. Pre-migration results are on the left and post-migration results are on the right.

"I think our team relies too much on a few persons to have knowledge about our Release and Deployment Management." is a statement that was inspired by the interviews, and the majority of the respondents agreed with the statement. The statement is somewhat similar to what was seen in The 2021 State of DevOps report [7], in which failure in sharing best practices was seen as a problem in some DevOps teams. As can be seen in the left-side chart in Figure 49. 22 respondents out of 36 agreed with the statement, out of these 16 somewhat agreed and five strongly agreed. The most popular reasons given in the free-form answers were that most people know how to use it, but only a few know how to manage or configure it and that there is a lack of documentation. Both of these were answered by two out of eight respondents the latter answer was expanded on by one of the respondents who thought that the lack of documentation means that people learn through their personal experience, which means that they only learn what is relevant to their work and not process as a whole. Some other respondents thought that as there have been quite a lot of new people joining and they have not been introduced to the process properly and that people don't know how the system is actually built, which is akin to the answer that people know how to use it but not how to configure it.

Less people seem to think that the team relies too much on a few persons in the post-migration system, as can be seen in Figure 49. Reasons for this shift in mentality can be found in the free-form answers, which have seven responses. Three out of those seven responses were some kind of variant of "at the moment the people who were working on the changes know about it, but it should be changed in the future". Two respondents thought that better documentation should be created to combat the issue. However, one person thought that documentation does not help in transferring expertise in maintaining

and developing. Moreover, another person thought that automation causes people to learn less about the automated aspects as they can then be ignored.

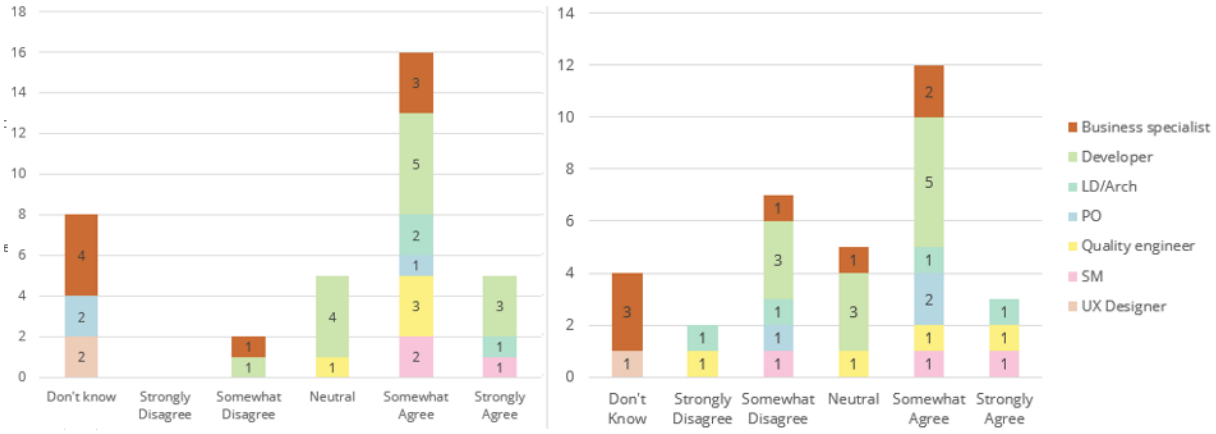

Figure 49. "I think our team relies too much on a few persons to have knowledge about our Release and Deployment Management." Pre-migration results are on the left and post-migration results are on the right.

It can be seen from the left-side chart in Figure 50. that in the pre-migration system, there is a clear lack of documentation as the most popular answer to the questions was somewhat disagree. Out of the 14 free-form answers given seven were about how documentation is lacking. The answers described several places where more should be documented such as failures and configuration. Some mentioned that documentation exists in bits and pieces, but no overarching documentation exists. Other negative aspects of documentation include out-of-date documentation, it is hard to find, and the processes are taught instead of documented. However, one respondent mentioned that the production release process is well documented.

The responses are more positive in the post-migration survey results with nine more agreeing answers to the statements as seen in Figure 50. However, this positivity is not reflected in the free-form answers in which the majority (six out of eleven) thought that the levels of documentation are not adequate. This includes such answers as non-technical and overarching documentation is still needed and that the documentation is non-existent. Three respondents thought that the levels of documentation are better now than before. Moreover, someone thought that you just have to know how to find it and that it is not the cure for everything as it can get out of date and it requires maintenance.

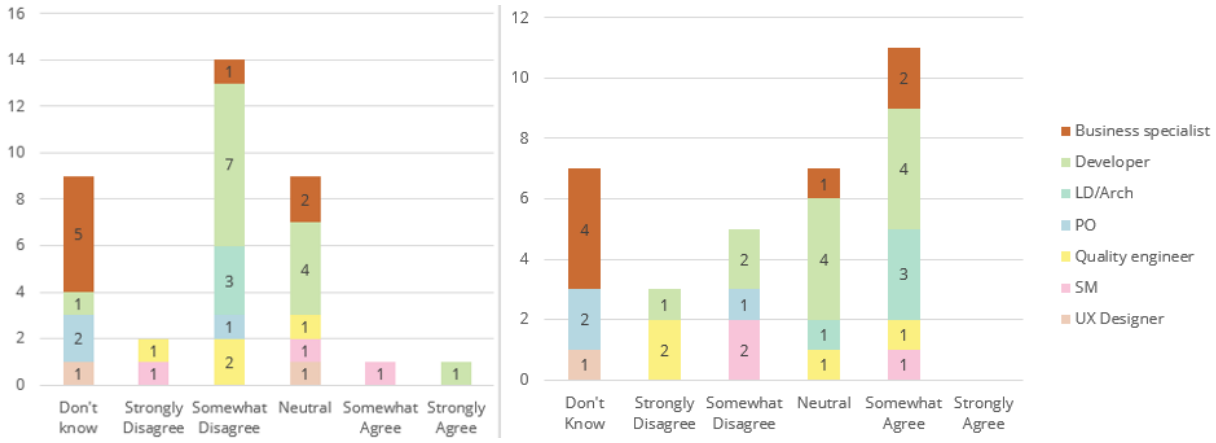

Figure 50. "I think we have adequate levels of documentation about our release and deployment management." Pre-migration results are on the left and post-migration results are on the right.

The left-side chart in Figure 51. shows that 15 out of 36 respondents agree with the statement that more resources should be allocated to Release and Deployment Management in the pre-migration survey. Eight free-form answers were given on the statement and three of them pointed out that the system that should be striven for does not require a lot of effort once it is in place and one answerer thought that a good enough solution should be adequate. Some other respondents wished to allocate more resources for documentation and that the process should be modernized.

The post-migration survey clearly shows that there have been enough resources allocated to Release and Deployment Management, as five respondents agree with the statement even though 15 agreed with it in the pre-migration survey. The free-form answers can be used to shed light on this change of heart to which 5 respondents answered. Three of them thought that a lot of resources have been invested into Release and Deployment Management already in the short term. Another respondent thought that through automation the amount of resources needed can be reduced and another one thought that more resources can be allocated to it if there are more resources available.

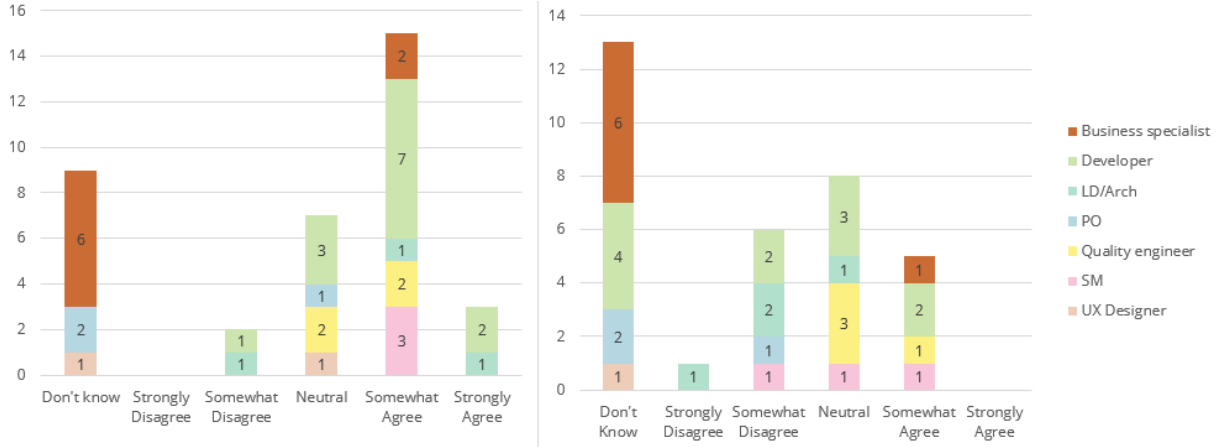

Figure 51. "I think we should allocate more resources to our Release and Deployment." Pre-migration results are on the left and post-migration results are on the right.

“Is there else regarding release and deployment management, that you would like to address that is not covered in this survey?” had three answers in the pre-migration survey with one Business Specialist answering that they do not know a lot about Release and Deployment Management even though the explanation given at the beginning helped a bit. A Developer thought that Feature tests are a time sink as they are run for so long. In the post-migration survey, a Quality Engineer thought that it was hard to scope what is included in Release and Deployment Management.

“Comments/feedback" had seven answers in the pre-migration survey from which two gave feedback on questions in the survey. One was about the statement “How much time do you spend on release and deployment management per day?” to which a Business Specialist answered that they only spend time with Release and Deployment Management during days with production releases. The other feedback was that one Quality Engineer was not sure was manual testing included in this Release and Deployment Management process. In the post-migration survey, two persons answered that the process has improved, and one Business Specialist thought that the business side is not that involved with Release and Deployment Management.

8.10 Summary of the Surveys Based on the Research Questions

Table 5 summarizes the results of the surveys based on the research questions. The table is further used together with Table 3 to derive the conclusions of this thesis.

Table 5. Summary of the surveys based on the research questions.

Research Question	Summary
RQ1. How does Release and Deployment Management support software development?	It is not always clear to people working in Release and Deployment Management systems in which ways it supports them. This can be seen from the statements regarding Experience in different fields of Release and Deployment Management because the respondents did not always realize they had expertise in something they actually had the expertise of. Additionally, the time spent a day per respondent could arguably be much more.

	The perceived way how Release and Deployment Management supports software development differs from role to role. This can be concluded by examining how different roles answered the questions in the surveys. There is a pattern that respondents from certain roles answered more neutral or “Don’t know” answers than others. The roles that had comparatively a lot more of these answers include UX Designer, Business Specialist and Product Owner. Thus, it could be said that people in these roles are the least affected by Release and Deployment Management, and to a degree feel like they are working outside of it. There was some change in this between the surveys, as for example, in the pre-migration survey two Product Owners answered that they are not directly working with Release and Deployment Management to the statement "I think our release and deployment management supports my way of working.". However, in the post-migration survey, two Product Owners thought that they would want to see more flexibility in releasing. Hence, this could indicate that it is more that the persons in these roles do not know in what ways they are affected. Additionally, the different roles seem to have different opinions on what changes they liked, which means that Release and Deployment Management supports the roles differently as well. The post-migration system was thought to be better in all of the attributes that were attached to the Release and Deployment Management in the literature, which would support the claim that the way Release and Deployment Management supports software development is by increasing the quality of the end product and it lowers the amount of manual work needed with automation. Moreover, it makes delivering software quicker, more transparent, predictable, replicable, flexible, smoother, safer and more stable. Additionally, it helps in recovering from failures, tracking errors and changes and it increases system integrity. All these factors increase the software development team’s productivity and save time.

RQ2. Do Release and Deployment Management tools make it easier for a scrum team to follow defined workflows?	The pre-migration system was thought of as a hindrance to agility, because of bottlenecks and the slowness of the process. In contrast, the post-migration system was thought more positively in regard to supporting working in Scrum and that it is easier to follow Scrum in the new system. However, a Quality Engineer thought that the testing process was still more waterfall-like, which means that the Release and Deployment Management process could still be waterfall-like even though the teams are working in Scrum. Therefore, it is essential for the Release and Deployment Management system to promote agility or it might end up hindering it.
RQ3. What aspects of Release and Deployment Management tools are most valued by a Scrum team?	Releasing taking a long time and that it is a big thing to do was seen as negative, thus, Release and Deployment Management tools should make releasing easy, quick and as stress-free as possible. These negative comments disappeared after the migration, which could suggest that the possibility of releasing without Maintenance Pages was seen as positive. Moreover, there is a clear want for more releases now that they are more seamless for the customer. Additionally, releases that the customers don't notice were seen as positive, thus the tools should offer the possibility of seamless releases. Tracking releases should be easy for everyone, not just the technically oriented members of the teams. This means that the information provided by the tools should be understandable to everyone. It is also beneficial that the tools are available in one place. The tools should enable a smooth flow from development to testing and to the release deployments, as it was identified as a negative by Quality Engineers that it takes long for them to get their hands on the new software. The post-migration process had been in use for 91 days when the post-migration survey was sent out, but some of the answers were about that it's still hard to tell what can be improved. Therefore, getting used to new tools takes, which means that they should be easy to learn. Moreover, it is rare that the process is perfect right away, which means that the tools should offer configurability and the possibility to improve them iteratively.

	Feature toggles are seen as important for the teams' Release and Deployment Management process. This emphasizes that the process for managing feature toggles is important, as it can quickly turn into a process that generates a lot of technical debt. Using Feature Toggles allows for coordination with dependencies. The amount of documentation was criticised in both pre-and -post-migration surveys. Especially documentation that would cover the whole process and not just bits and pieces and non-technical documentation. Documentation was thought to remedy the issue that too few persons have all the knowledge about the process. Documentation Was said to have downsides as well, because it cannot transfer expertise in maintaining and developing and that it requires effort to keep it up to date. The pre-migration system was thought to have a lot of points of failures and that there are a lot of things that could go wrong, this would suggest that it is important there are features that would enable the teams to recover from failures, such as logging and monitoring.
RQ4. How much of the deployment can and should be automated by using modern CI/CD solutions?	Negative feedback on the pre-migration and on the post-migration process included that there should be more automation and better test automation. Some respondents want to get rid of all manual steps and that there should never be morning releases. The reason why the level of automation should be added is because it reduces the amount of manual labour. However, automation was also seen as something that could reduce learning as "…automation often leads into people not learning what and why things are happening." Unreliable automation is seen as a negative factor, hence if automation cannot be implemented in a way that it is trustworthy it could be argued that it should not be implemented at all. Additionally, fixing broken automation should be a priority task and monitoring for the automation should be implemented. Content management is acknowledged as a hindrance to automation.

9 Discussion

This chapter begins by covering factors which may have affected the outcome of the research. Following that the chapter discusses the shortcomings of the survey and the reasoning behind the choices made in the survey. After that, the validity of measuring a Release and Deployment Management System using attributes from the literature is presented. Then the chapter continues into reflecting the research done in this thesis with previous research in the field. The reflection is done first on the basis of what differences this research has with previous research and then the findings of this research are compared with previous research. After that the limitations are discussed and the chapter ends by proposing possibilities for future research.

The opinions that people have about Release and Deployment Management could have been affected by talks that have been happening related to that topic. This includes updates on what has been done, what the plans are for the future and why such an endeavour is even necessary. This can be seen in the interviews, especially when the interviewees were asked if they think enough attention is paid to Release and Deployment Management, to which some interviewees responded that yes, because of the work done in this thesis. An issue with Release and Deployment Management can be noticed in the interviews and the surveys is that people do not think about it that much and it is more something that is in the background. This means that people did not always have the most thought-out opinions on the matter.

Initially in the interviews and later in the surveys, it became clear that the term Release and Deployment Management was not communicated clearly enough as a holistic process. This is especially noticeable in the answers of people who are not directly working with technology, such as Business Specialists, Product Owners and UX Designers. It is a worrying trend to notice that some non-technical-oriented people who are working tightly together in the Release and Deployment Management process see it as too technical to be relevant for them, even though essentially everything that the teams are working on is technical. Planned features rarely come across as too technical to understand. Rather the features are something concrete and understandable, although they are made with technology and their implementation is technical. Therefore, efforts should be made to make technical processes more available for the non-technical roles, because in the end, such processes should support everyone.

In the survey results a common group is created for the Lead Developers and the architect. This is done because otherwise, it would be clear what the opinion of the architect was in every question as there is only one architect. The survey is quite long, as it takes 20-30 minutes or longer to complete it, depending on how many open-questions are answered. Although the survey was long, the expectation

was that the respondents would take the necessary time to complete it, as it was answered during work hours. However, it can be noticed that there are more open-questions answered at the beginning of the survey than at the end of it.

Measuring Release and Deployment Management systems using a SUS-like scoring but with attributes that are attached to Release and Deployment Management systems seemed to have been more understandable, especially by the non-technical respondents. This can be seen from the data as in the Pre-migration survey the SUS matrix had “Don’t know” answers 68% of the time from POs, SMs, UX Designers and Business Specialists, but in this newer matrix “Don’t know” was answered 52% of the time. This means that it could yield better results than the SUS score when measuring Release and Deployment Management Systems.

Another question is, how reliable is it to compare different attributes with one another, as this is not done in the SUS scoring systems? Comparing the scorings of statements using Table 6, a general trend can be noticed; the higher the score of the statement, the more positive answers and less negative answers it has. However, the scoring is not perfect and leaves some room for interpretation. For example, “...be more productive” and ” ...keep our quality of work high” have the same exact score, because the latter has more positive answers but more negative answers as well. Therefore, the individual scoring’s purpose is not to give an absolute answer on what way the Release and Deployment Management System is the best, but rather it gives direction and guidance. Additionally, Table 6 offers valuable information on which attributes are well-understood and which are not. For example, “…deliver value in a replicable way” has 15 “Don’t know” answers, which is almost 50% of all the respondents. This suggests that the statement could be more informative, for example, “deliver value in the same manner each time”, or it could have an additional explanation informing that replicability means that value can be delivered using the same processes and practices every time with the same end results.

Table 6. Count of each answer in the Release and Deployment Management attribute matrix in the pre-migration survey.

Statement	Don't know	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree	Pre-migration score
...work in a transparent way	8	0	1	12	13	2	3,44
...do less manual work	10	3	4	3	9	7	3,36
...be more productive	7	1	6	11	5	6	3,25

...keep our quality of work high	5	1	7	9	10	4	3,25
...easily track the states and environments our tasks currently are in	4	2	7	8	12	3	3,19
...deliver value in a predictable way	13	2	4	7	7	3	3,14
...easily keep track of changes	8	3	5	7	10	3	3,14
...deliver value in a safe and stable way	9	1	7	10	6	3	3,08
...deliver value in a replicable way	15	2	4	8	5	2	3,03
...flexibility when delivering value to our customers	9	1	10	5	9	2	3,03
...have a smooth flow of work	10	2	8	7	7	2	2,97
...easily track down error cases	9	4	7	5	9	2	2,94
...quickly deliver value to our customers	7	2	12	6	5	4	2,92
...have higher system integrity	14	2	10	6	5	0	2,78

9.1 Reflection on Previous Research

The basis of this thesis is on the current IT literature. It is then applied in real-life use by using that theory to build a Release and Deployment Management system and to gather data on what Release and Deployment is in real-life scenarios. Most of the material used for this thesis covers the topic of Release and Deployment Management from a narrower viewpoint. For example, sources such as [7], [30], [35], [36], [65] were used to write the DevOps chapter, but they covered only a part of Release and Deployment Management. The same applies to sources such as [3], [52], [53], [54], and [55], which were used for the Chapter about CI/CD. Therefore, this thesis tries to bring all of these under the same umbrella of Release and Deployment Management as the different viewpoints do not exist in a vacuum, but rather they affect each other, and they are often also used together. The thesis also combines the literature background to establish a set of attributes through which Release and Deployment Management Systems can be rated. The list of all the attributes attached to Release and Deployment Management can be found in Table 1.

This thesis offers a unique viewpoint into the world of Release and Deployment Management as it is conducted from within a development team. This gives the advantage that the author knows the inner

workings of said team. Additionally, some of the research [4], [5], [6], [7] in the literature background only had one survey or an interview, but in this thesis, there is first an interview which is then used as the basis for the survey, which is done longitudinally. Moreover, the longitudinal survey creates additional value as the teams are using different Release and Deployment Management tools in the surveys, which means that the tools can be isolated and better compared.

When the results of this research are compared with previous research in the same field, similarities can be found but new findings were made as well. This could be explained by the unique viewpoint used in the thesis since respondents and interviewees were motivated to give quality feedback because that feedback will be used to improve the Release and Deployment Management process that they use daily in their work. In the research done by Olsson, et al. [6], it was identified that for an organization to adopt Continuous Deployments it is important for product management to be fully involved in the product delivery process. Therefore, the results in this thesis present a difficult dilemma as the Product Owners and Business Specialists feel alienated from the process due to its technical nature. This would imply that getting everyone on board in the Release and Deployment Management process is crucial to unlock Continuous Deployments. This is further supported by the study done by Neely & Stolt [47] in which it was found that adopting Continuous Delivery means that everyone will be affected and the process should be clearly communicated. Neely & Stolt [47] also made a point that code is not made safer by longer periods between deployments, but quite the opposite was mentioned in the interviews. Therefore, it is possible that if the code becomes safer with time, the automated test coverage may not be up to standard for Continuous Delivery.

More automation and Continuous Deployments were seen by the majority of interviewees as a goal for the Release and Deployment Management system. This contradicts the findings of a study [4] about nine problems in Release Management, which includes a high distribution rate as a problem. This raises the question of whether the high distribution rate of releases, which is achieved by Continuous Deployments, is something that the customers want or is this just something that modern development methodologies aim for because value should be delivered quickly to the customer.

Qualitative analysis [3] done by Mazrae, et al. on CI/CD listed valuable features for CI tools. The findings of the study by Mazrae, et al. had some of the same results with this thesis, namely *ease of use, customizability, speed and documentation*. Findings that were not included in this study include *good integration with hosting platform, support specific architecture/OS, good free tier support, popularity/familiarity, good plugin support, self-hosting abilities* and *security*. Arguably, it could be said that *good integration with hosting platform*, *support specific architecture/OS* and *popularity/familiarity* are reasons why it was seen as desirable to migrate to Azure DevOps. This is because everything else used for making the product is made by Microsoft except the tools that were

migrated. On the other hand, this thesis included features such as the *possibility for iterative improvements, automation, effortless and seamless releases, feature toggling* and the idea that *information provided by the tools should be understandable to everyone*. Some of the release-specific features (*effortless and seamless releases and feature toggling*) were not found in the research by Mazrae, et al. because that research focused on CI tools. The reason why the *possibility of iterative improvements* was most likely only found in this study is because it was from the migration perspective. Furthermore, the *information provided by the tools should be understandable to everyone* was likely absent because the study by Mazrae, et al. only interviewed Developers, whereas this research included non-Developers as well. *Automation* could have been thought of as given in the study by Mazrae, et al., and therefore it was not listed as a feature.

This study concludes that automation such as Continuous Deployments should only be done if the team thinks they are comfortable with that process. A similar point is made in the Puppet Labs 2021 State of DevOps report [7] which states that “DevOps is not just automation”. The same report expands that statement by making it clear that, even though highly evolved DevOps organizations have high levels of automation that is not the only reason why they are highly evolved. The research in this thesis supports the statement by Puppet Labs that automation alone is not the solution for good Release and Deployment Management.

Section 2.2.5 Release and Deployment Management in Scrum reaches the conclusion that for Scrum to embody the Agile values, production releases should not be done on a Sprint basis, but rather value should be delivered to the customer as fast as possible. These conclusions are supported by research done in Continuous Scrum [29]. The research in this thesis confirms the thinking behind the development of Continuous Scrum was correct; Scrum is only as Agile as the Release and Deployment Management process allows. Therefore, not only does Release and Deployment Management make it easier to follow the pre-defined Scrum workflows, but it is also the key to enabling Agile workflows.

9.2 Limitations

The main limitation of this thesis is time, which in this case means that not everything that was planned was completed during the scope of this thesis. Thus, the implemented pipeline cannot be used to do production releases without maintenance pages in every situation, because there is no staging slot database. However, deployments without maintenance pages are possible when there are no breaking UI changes. The work to implement the staging slot database is going to continue after this thesis. The work to create pipelines is rather simple, at least now in retrospect, and the problems arose

mainly from the legacy aspects of the project just as is mentioned in [47]. Additional problems came from an ongoing .NET migration and other infrastructure changes that sometimes interfered with the pipeline work. Thus, in the future, it would be advisable to do one infrastructure change at a time. However, in this case, as so many things are already in need of updating it was not always possible. Some aspects of the Release and Deployment Management process are done differently in the .NET version that the solution is being migrated into, which means that some aspects should be done only after the .NET migration or they would have to be done twice.

There is also a limitation on how much can be changed in the processes in one go, especially when adopting new tools. It is not regarded as feasible at the moment to move away from the time-based release model or to adopt a continuous deployment process. This is because it takes practice to manage a new process and with the teams doing so many other infrastructure projects it is not a priority at the moment.

9.3 Future Research

There are two different avenues identified for the topic of future research: (1) In the surveys and interviews it became apparent that non-technical members of the team thought that they were working outside of Release and Deployment Management. Therefore, a possible subject could be bridging the gap between technical-sounding processes such as the Release and Deployment Management process and non-technical members of the team. (2) Some team members want more automation, while some thought that too much automation could be harmful for learning what and why something is done. This is interesting, as automation is generally thought of as making processes more efficient by reducing the amount of manual labour, but does it also have the side-effect of making Developers less proficient? This presents the opportunity to research whether automation in fact does more harm than good.

10 Conclusions

The purpose of this thesis was to migrate the Release and Deployment Management tools from one platform to another. Additionally, the thesis aimed to further improve the existing Release and Deployment process to allow for a more seamless customer experience during production releases by removing the necessity of maintenance pages. The introduction of maintenance page free deployments means that the deployments would not have to be done during the early hours of the day. Furthermore, they are in a key role in enabling a true Continuous Integration / Continuous Deployment process. The tools were successfully migrated during this study, and it was proven that maintenance pages are not needed during deployments that do not have breaking UI changes.

The academic purpose of this thesis was to follow the migration process and to collect data before and after the migration process. The data that was collected before the migration was collected firstly by interviews in which a representative of each role was interviewed to gather information about Release and Deployment Management in general. Secondly, a pre-migration survey was sent out which all the team members used to rate the pre-migration Release and Deployment Management process. Post-migration data was collected by sending out the same exact survey that was used to rate the pre-migration process. The data collected in the interviews and surveys was used together with the current IT literature to answer four research questions. The similarities between the conclusions made in this thesis and the results of previous research are reflected in section 9.1.

The first research question was *RQ1. How does Release and Deployment Management support software development?* The importance of Release and Deployment Management is derived from necessity. It mostly affects in the background of software development, and it is rarely the main focus. In the interviews, the representatives of the roles of UX Designer and Scrum Master did not think it was important to their roles. In the survey, there were Business Specialists, UX Designers, and Product Owners who thought that they were not working with Release and Deployment Management. Therefore, it could be said that Developers, Quality Engineers, Lead Developers and Architects are supported by Release and Deployment Management the most, while others are not as much. However, this may be caused by Release and Deployment Management being the most visible to Developers, Quality Engineers, Lead Developers and Architects. Therefore, Release and Deployment Management is more important to the more technical roles than to others, and it is perceived to support those roles more as well. Additionally, the different roles have different opinions on changes, which means that they are supported in different ways.

Release and Deployment Management supports software development by increasing the quality of the end product and the amount of manual work is lowered through automation. Moreover, it makes

delivering software quicker, more transparent, predictable, replicable, flexible, smoother, safer and more stable. Additionally, it helps in recovering from failures, tracking errors and changes and increasing system integrity. All these factors increase the software development team's productivity and it saves time.

The second research question was *RQ2. Do Release and Deployment Management tools make it easier for a scrum team to follow defined workflows?* Ineffective Release and Deployment Management systems are seen as a hindrance to agility, while effective ones are seen as supporting Agile development. Therefore, it is closely intertwined with Agile software development, and thus, it is thought of as the enabler for Agile software delivery. It is not enough to develop the software in an Agile manner, but the Release and Deployment Management system must support it. Scrum was seen as having a similar relationship with Release and Deployment Management. Furthermore, all the interviewees thought that workflows in Scrum can be made easier to follow with Release and Deployment Management and these views are further backed up by the surveys, in which the post-migration system was thought to support the Scrum process more.

The third research question was *RQ3. What aspects of Release and Deployment Management tools are most valued by a Scrum team?* There are as many possible Release and Deployment Management processes as there are software projects, which means that the tools used must be configurable in order to be tailored to the exact needs of the project. Moreover, as the process may be complex, they must allow for iterative improvements, because all the requirements are rarely met on the first try. It also takes a considerable amount of time to get used to new Release and Deployment Management tools, as was seen in the post-migration survey. Even though the tools had been in use for 91 days already before the post-migration survey was sent out, some respondents thought that they had not been in use for long enough to find improvements. Additionally, this emphasises that it should be easy to use the tools.

The single most important aspect that was mentioned in the interviews was automation and this was confirmed in the surveys as well, where the post-migration level automation did not satisfy all the respondents. In addition to automation, documentation is seen as an important aspect, as it can help to remedy the problem that knowledge is concentrated on a selected few.

Releasing is seen as something that is always stressful for the people who are responsible for it. Therefore, the tools should offer ways to mitigate the stressfulness and they should allow for fail-safes, logging, monitoring and other ways that can be used in cases of failure. It was thought to be positive that the deployments do not have to be done in the early hours of the morning. The problem with early releases is that most of the team members are not working yet and cannot support the

release. Releasing should be effortless so that small releases can be afforded, because the fewer the changes the fewer things can go wrong. Releasing with fewer changes and more frequently means that the tools should offer the possibility to release without disturbance to the customers.

The Release and Deployment Management tools should offer information that can be understandable and usable by every team member and not just technically oriented team members. That information should not be spread out across multiple platforms, as one of the benefits of the post-migration system is that the tools are all in Azure DevOps instead of in three different platforms. Another project-specific requirement for the tools was the ability to turn features on and off in the production environment. This enables the release of features before the content managers have set the content for them or before dependencies are ready. However, managing feature toggles should be made easy as it can quickly generate technical debt if old feature toggles are not removed.

The fourth research question was *RQ4. How much of the deployment can and should be automated by using modern CI/CD solutions?* The pipeline that was implemented in this thesis has the capability to automate the whole Release and Deployment Management process, but it would require the use of feature toggles for all features. However, the teams do not consider themselves good at handling feature toggles, and therefore automating the whole process is not feasible. Therefore, the automation should only be implemented if the team is comfortable with it. Additionally, some respondents thought that implementing higher levels of automation would make the process more complex.

Even though a continuous deployment process could not be implemented at this time, respondents in both the interviews and surveys requested more automation. Nevertheless, when implementing automation, it should always be considered that the automation is reliable because unreliable automation was identified as one of the most common defects in the pre-migration system. This would suggest that broken automation is as good as no automation at all. A negative aspect attached to automation by some of the answers in the interviews and the survey was that automation was said to sometimes lead to people not learning what and why something is done.

The findings of this thesis had similarities but also differences with previous research conducted in the same field. Regarding RQ1 the attributes attached to Release and Deployment Management were verified in this study. Similarly, in regards to RQ2, there have been findings in previous research in Continuous Scrum [29] that indicate that Release and Deployment Management is needed in Scrum to enable Agile workflows. Similar findings related to RQ3 have been done in research by Mazrae, et al. [3]. This includes aspects such as *ease of use, customizability, speed and documentation.* However, new aspects that are valued in Release and Deployment Management tools were found in this study, too. This includes *possibility for iterative improvements, automation, effortless and seamless releases,*

feature toggling and the idea that *information provided by the tools should be understandable to everyone.* In regards to RQ4, it is concluded that the automation should only be implemented if the team is comfortable with it. The conclusion is similar to the point made in the Puppet Labs 2021 State of DevOps report [7] which states that “DevOps is not just automation”.

References

[1] R. K. Yin, *Case Study Research: Design and Methods*. SAGE, 2003.

[2] F. Shull, J. Singer, and D. I. K. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Springer, 2008.

[3] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan, "On the usage, co-usage and migration of CI/CD tools: A qualitative analysis," *Empir. Softw. Eng.*, vol. 28, no. 2, p. 52, Mar. 2023, doi: 10.1007/s10664-022-10285-5.

[4] A. Lahtela and M. Jäntti, "Challenges and problems in release management process: A case study," in *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, Jul. 2011, pp. 10–13. doi: 10.1109/ICSESS.2011.5982242.

[5] K. Jokela and M. Jäntti, "Challenges and problems in product portfolio release and deployment management: A case study," in *ICSSSM12*, Jul. 2012, pp. 138–141. doi: 10.1109/ICSSSM.2012.6252208.

[6] H. H. Olsson, H. Alahyari, and J. 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*, Sep. 2012, pp. 392–399. doi: 10.1109/SEAA.2012.54.

[7] Puppet Labs, "2021 State of DevOps Report." 2021. Accessed: Sep. 16, 2023. [Online]. Available: https://www.puppet.com/resources/history-of-devops-reports#2021

[8] R. B. Johnson and A. J. Onwuegbuzie, "Mixed Methods Research: A Research Paradigm Whose Time Has Come," *Educ. Res.*, vol. 33, no. 7, pp. 14–26, Oct. 2004, doi: 10.3102/0013189X033007014.

[9] A. Dearle, "Software Deployment, Past, Present and Future," in *Future of Software Engineering (FOSE '07)*, May 2007, pp. 269–284. doi: 10.1109/FOSE.2007.20.

[10] A. van der Hoek, R. S. Hall, D. Heimbrigner, and A. L. Wolf, "Software release management".

[11] A. Gordon and S. Hernandez, *The Official (ISC)2 Guide to the SSCP CBK*. John Wiley & Sons, 2016.

[12] A. Limited, *ITIL® 4: Digital and IT Strategy*. London, UNITED KINGDOM: The Stationery Office Ltd, 2020. Accessed: Jun. 15, 2023. [Online]. Available: http://ebookcentral.proquest.com/lib/kutu/detail.action?docID=6367312

[13] G. B. O. of G. Commerce, *Service operation*. The Stationery Office, 2007.

[14] G. Poo-Caamaño, "Release Management in Free and Open Source Software Ecosystems," 1994.

[15] H. K. Wright, "Release Engineering Processes, Their Faults and Failures".

[16] J. Teixeira, "Release Early, Release Often and Release on Time. An Empirical Case Study of Release Management," in *Open Source Systems: Towards Robust Practices*.

[17] M. Michlmayr, "Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management," 2005.

[18] M. Michlmayr, B. Fitzgerald, and K.-J. Stol, "Why and How Should Open Source Projects Adopt Time-Based Releases?," *IEEE Softw.*, vol. 32, no. 2, pp. 55–63, Mar. 2015, doi: 10.1109/MS.2015.55.

[19] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, Inc., 2001.

[20] K. Schwaber, *Agile Project Management with Scrum*. Microsoft Press, 2004.

[21] "What Is Scrum? by Ken Schwaber." Accessed: Aug. 06, 2023. [Online]. Available: https://www.volaroint.com/wp-content/uploads/dlm_uploads/2014/03/DC-VOLARO-Training-Scrum-What_Is_Scrum.pdf
[22] "Agile Software Development: The Business of Innovation." Accessed: Aug. 06, 2023. [Online]. Available: http://sunnyday.mit.edu/16.355/highsmith-agile.pdf
[23] pmadmin, "What is Scrum? | The Agile Journey with PM-Partners," PM Partners. Accessed: Aug. 11, 2023. [Online]. Available: https://www.pm-partners.com.au/the-agile-journey-a-scrum-overview/
[24] R. Mokhtar and M. Khayyat, "A Comparative Case Study of Waterfall and Agile Management," *SAR J. - Sci. Res.*, pp. 52–62, Mar. 2022, doi: 10.18421/SAR51-07.
[25] K. Petersen, C. Wohlin, and D. Baca, "The Waterfall Model in Large-Scale Development," in *Product-Focused Software Process Improvement*.
[26] S. Sachdeva, "Scrum Methodology," *Int. J. Eng. Comput. Sci.*, Jun. 2016, doi: 10.18535/ijecs/v5i6.11.
[27] N. B. Moe, T. Dingsyr, and O. Kvangardsnes, "Understanding Shared Leadership in Agile Development: A Case Study," in *2009 42nd Hawaii International Conference on System Sciences*, Jan. 2009, pp. 1–10. doi: 10.1109/HICSS.2009.480.
[28] A. Begel and N. Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Sep. 2007, pp. 255–264. doi: 10.1109/ESEM.2007.12.
[29] S. S. Samarawickrama and I. Perera, "Continuous scrum: A framework to enhance scrum with DevOps," in *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, Sep. 2017, pp. 1–7. doi: 10.1109/ICTER.2017.8257808.
[30] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.
[31] R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices," in *Proceedings of the Scientific Workshop Proceedings of XP2016*, Edinburgh Scotland UK: ACM, May 2016, pp. 1–11. doi: 10.1145/2962695.2962707.
[32] Puppet Labs & IT Revolution Press, "2013 State of DevOps Report." 2013. Accessed: Sep. 05, 2023. [Online]. Available: https://www.puppet.com/resources/history-of-devops-reports#2013
[33] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.
[34] Sushma T N and RV College of engineering,Bangalore, "Automation of Software Development using DevOps and its Benefits," *Int. J. Eng. Res.*, vol. V9, no. 06, p. IJERTV9IS060369, Jun. 2020, doi: 10.17577/IJERTV9IS060369.
[35] A. Katal, V. Bajoria, and S. Dahiya, "DevOps: Bridging the gap between Development and Operations," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, Mar. 2019, pp. 1–7. doi: 10.1109/ICCMC.2019.8819631.
[36] S. M. Mohammad, "DevOps Automation Advances I.T. Sectors with the Strategy of Release Management," *Int. J. Comput. Trends Technol.*, vol. 67, no. 12, pp. 82–88, Dec. 2019, doi: 10.14445/22312803/IJCTT-V67I12P114.
[37] F. Bildiri and Ö. Akdemir, "From Agile to DevOps, Holistic Approach for Faster and Efficient Software Product Release Management".
[38] J. Humble and J. Molesky, "Why Enterprises Must Adopt Devops to Enable Continuous Delivery," *Cut. IT J.*, vol. 24, no. 8, pp. 6–12, Aug. 2011.

[39] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.
[40] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.
[41] T. Fitz, "Continuous Deployment." Accessed: Sep. 17, 2023. [Online]. Available: http://timothyfitz.com/2009/02/08/continuous-deployment/
[42] M. Leppänen *et al.*, "The highways and country roads to continuous deployment," *IEEE Softw.*, vol. 32, no. 2, pp. 64–72, Mar. 2015, doi: 10.1109/MS.2015.50.
[43] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *J. Syst. Softw.*, vol. 123, pp. 176–189, Jan. 2017, doi: 10.1016/j.jss.2015.06.063.
[44] F. A. Abdul and M. C. S. Fhang, "Implementing Continuous Integration towards rapid application development," in *2012 International Conference on Innovation Management and Technology Research*, May 2012, pp. 118–123. doi: 10.1109/ICIMTR.2012.6236372.
[45] E. Soares, G. Sizilio, J. Santos, D. Costa, and U. Kulesza, "The effects of continuous integration on software development: a systematic literature review," *Empir. Softw. Eng.*, vol. 27, May 2022, doi: 10.1007/s10664-021-10114-1.
[46] P. Swartout, *Continuous Delivery and DevOps – A Quickstart Guide: Start your journey to successful adoption of CD and DevOps, 3rd Edition*. Packt Publishing Ltd, 2018.
[47] S. Neely and S. Stolt, "Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)," in *2013 Agile Conference*, Aug. 2013, pp. 121–128. doi: 10.1109/AGILE.2013.17.
[48] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
[49] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at Facebook and OANDA," May 2016, pp. 21–30. doi: 10.1145/2889160.2889223.
[50] I. Weber, S. Nepal, and L. Zhu, "Developing Dependable and Secure Cloud Applications," *IEEE Internet Comput.*, vol. 20, no. 3, pp. 74–79, May 2016, doi: 10.1109/MIC.2016.67.
[51] J. Humble, "Continuous Delivery vs Continuous Deployment - Continuous Delivery," Continuous Delivery. Accessed: Sep. 17, 2023. [Online]. Available: https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/
[52] D. Medvedev and K. Aksyonov, "The Development of a Simulation Model for Assessing the CI/CD Pipeline Quality in the Development of Information Systems Based on a Multi-Agent Approach," in *MATEC Web of Conferences*, Les Ulis, France: EDP Sciences, 2021. doi: 10.1051/matecconf/202134603095.
[53] M. R. Pratama and D. Sulistiyo Kusumo, "Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, Aug. 2021, pp. 230–235. doi: 10.1109/ICoICT52021.2021.9527496.
[54] S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, May 2018, pp. 156–161. doi: 10.1109/MERCon.2018.8421965.
[55] M. Soni, "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery," in *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Nov. 2015, pp. 85–89. doi: 10.1109/CCEM.2015.29.

[56] D. Barker, *Web Content Management: Systems, Features, and Best Practices*. O'Reilly Media, Inc., 2016.
[57] J.-M. Martinez-Caro, A.-J. Aledo-Hernandez, A. Guillen-Perez, R. Sanchez-Iborra, and M.-D. Cano, "A Comparative Study of Web Content Management Systems," *Information*, vol. 9, no. 2, Art. no. 2, Feb. 2018, doi: 10.3390/info9020027.
[58] M. T. Rahman, L.-P. Querel, P. C. Rigby, and B. Adams, "Feature toggles: practitioner practices and a case study," in *Proceedings of the 13th International Conference on Mining Software Repositories*, in MSR '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 201–211. doi: 10.1145/2901739.2901745.
[59] TeamCity, "Continuous Integration with TeamCity." Accessed: Mar. 11, 2023. [Online]. Available: https://www.jetbrains.com/help/teamcity/2023.05/continuous-integration-with-teamcity.html#Basic+CI+Workflow+in+TeamCity
[60] Octopus Deploy, "Getting started." Accessed: Mar. 11, 2023. [Online]. Available: https://octopus.com/docs/getting-started
[61] Microsoft, "Tutorial: Create a multistage pipeline with Azure DevOps." Accessed: Apr. 11, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/process/create-multistage-pipeline?view=azure-devops
[62] Microsoft, "Set up staging environments in Azure App Service." Accessed: Mar. 03, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/deploy-staging-slots?tabs=portal
[63] "System Usability Scale (SUS)." Accessed: Jan. 05, 2024. [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html
[64] J. Brooke, "SUS: A quick and dirty usability scale," *Usability Eval Ind*, vol. 189, Nov. 1995.
[65] L. Zhu, L. Bass, and G. Champlin-Scharff, "DevOps and Its Practices," *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, May 2016, doi: 10.1109/MS.2016.81.

Appendices

Appendix 1

Background

This is an interview for my master's thesis that is about Release and Deployment Management

1. What is your position in the team/company?
2. How long have you been working in the team?
3. How much experience do you have in the field you are working in?
4. Working in a Scrum team?

5. How would you describe Release and Deployment Management?
(*Release and Deployment Management contain all the activities that happen to the software after the new version of it has been created by the Developer and before the software is used by the end customer. This would include such activities as building and packaging the software, running automated tests, deployments to test environments and finally deployment of the release to the live environment. On top of this logging of all steps and tracking of dependencies, changes etc.*
Conversation about how our definitions differ)
6. How much experience do you have in Release and Deployment Management?
7. Are you interested in Release and Deployment Management?

Importance of Release and Deployment Management

8. Is Release and Deployment Management part of your work? (In what way?)
9. Is Release and Deployment Management important to your role? (How?)
10. Do you think that your team pays enough attention to Release and Deployment Management? (Why?)
11. Do you think it's worthwhile to spend time in Release and Deployment Management? (Why?)

Of Release and Deployment Management

12. What benefits can be achieved with Release and Deployment Management? (More value quicker to the customer, safer more stable, predictability, replicability, releases require less effort, easier to recover from failures, code quality, higher system integrity, less manual work and more productivity?)
13. What aspects of Release and Deployment Management are hard?
14. What problems can be solved with Release and Deployment Management?
15. What do you think are Release and Deployment Management best practices?

Agile and Release and Deployment Management

16. What is the relation between Agile and Release and Deployment Management?
17. Is Release and Deployment Management part of the agile processes?
18. Do you think Scrum influences Release and Deployment Management?
19. How do Release and Deployment Management tools influence the Scrum process?

Release and Deployment Management features

20. What features are in your opinion important in Release and Deployment Management? (Why are they important?)
21. What is currently done well regarding Release and Deployment Management in your team? (Why?)
22. What should be done differently? (Why?)
23. What do you think of DevOps in Release and Deployment Management?
24. What do you think of CI/CD in Release and Deployment Management?

Release and Deployment Management Strategies

25. Which one do you think is better a feature or a time-based Release and Deployment Management strategy? (Current DRM strategy or something else?) and why?
26. How much do you think should be automated? What parts? (*building and packaging the software, running automated tests, deployments to test environments and finally deployment of the release to the live environment)*
27. Do you prefer continuous deployment or continuous delivery? Why?
(in Continuous Delivery the process leading up to the release of the software is automatic, but the release is not. In contrast in Continuous Deployment, the whole process is automated
28. Do you think feature toggles are important in Release and Deployment Management? (Why?)
29. Does Content Management have an effect on Release and Deployment Management? (How?)

Appendix 2

Release and Deployment Management survey

Please complete the survey below.

Thank you!

What is Deployment and Release Management?
It contains all the activities that happen to the software after the new version of it has been created by the developer, and before the software is used by the end customer. On a practical level, this could be thought of as the way our changes flow from the developer's computer through all the different test environments all the way to the production environment.
On a more technical level, this includes such activities as building and packaging the software, running automated tests, deployments to test environments and finally deployment of the release to the live environment. On top of this, monitoring, logging of all steps, tracking dependencies and changes etc.

1) What is your position in the team?

- ◯ Scrum Master
- ◯ Product Owner
- ◯ Business specialist
- ◯ UX Designer
- ◯ Developer
- ◯ Quality engineer
- ◯ Lead Developer
- ◯ Architect
- ◯ Other

2) If other, what?

_____

3) How much experience do you have in the field you are currently working in?

_____

4) How long have you been working with the MyPages project?

_____

5) What fields of release and deployment management do you have experience from?

- ☐ using them
- ☐ building them
- ☐ configuring them
- ☐ process management
- ☐ test automation
- ☐ statistics
- ☐ version control
- ☐ business verification
- ☐ creating documentation
- ☐ monitoring
- ☐ DevOps
- ☐ CI/CD

6) What fields of release and deployment management do you have experience from in your team?

- ☐ using them
- ☐ building them
- ☐ configuring them
- ☐ process management
- ☐ test automation
- ☐ statistics
- ☐ version control
- ☐ business verification
- ☐ creating documentation
- ☐ monitoring
- ☐ DevOps
- ☐ CI/CD

		Don't Know	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
7)	I like using our release and deployment management system.	○	○	○	○	○	○
8)	I find our release and deployment management system unnecessarily complex.	○	○	○	○	○	○
9)	I think our release and deployment management system is easy to use.	○	○	○	○	○	○
10)	I think I need the help of a technical person to understand our release and deployment management system.	○	○	○	○	○	○
11)	I find that the various functions in our release and deployment management system are well integrated.	○	○	○	○	○	○
12)	I think there are too many inconsistencies in our release and deployment management system.	○	○	○	○	○	○
13)	I think most people learn quickly how to use our release and deployment management system.	○	○	○	○	○	○
14)	I find our release and deployment management system cumbersome to use.	○	○	○	○	○	○
15)	I feel confident in our release and deployment management system.	○	○	○	○	○	○
16)							

I needed to learn a lot of things before I could get going with our release and deployment management system.	○	○	○	○	○	○

I think our release and deployment management enables us to...

	Don't Know	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
17) ...keep our quality of work high	○	○	○	○	○	○
18) ...easily keep track of changes	○	○	○	○	○	○
19) ...easily track down error cases	○	○	○	○	○	○
20) ...have higher system integrity	○	○	○	○	○	○
21) ...flexibility when delivering value to our customers	○	○	○	○	○	○
22) ...do less manual work	○	○	○	○	○	○
23) ...be more productive	○	○	○	○	○	○
24) ...have a smooth flow of work	○	○	○	○	○	○
25) ...work in a transparent way	○	○	○	○	○	○
26) ...easily track the states and environments our tasks currently are in	○	○	○	○	○	○
27) ...quickly deliver value to our customers	○	○	○	○	○	○
28) ...deliver value in a safe and stable way	○	○	○	○	○	○
29) ...deliver value in a predictable way	○	○	○	○	○	○
30) ...deliver value in a replicable way	○	○	○	○	○	○

31) Additional comments?

_____

32) I think our release and deployment management supports my way of working

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

33) Why / Why not?

_____

34) I think we have adequate levels of documentation about our release and deployment management.

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

35) Why / Why not?

_____

Production releases

36) I like the way we do our production releases.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

37) What do you like/dislike?

_____

38) The amount of production releases we have is sensible.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

39) Why / Why not?

_____

40) I think we are good at planning what features we will have in our releases.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

41) Why / Why not?

_____

42) I think it is easy and reliable to find out what our releases contain

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

43) Why / Why not?

_____

44) The way we do production releases makes me rush my work

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

45) Why / Why not?

_____

46) We rarely have problems with our releases.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

47) Why / Why not?

_____

48) Our production releases are stressful.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

49) Why / Why not?

_____

50) It is easy to recover from failures when doing production releases.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

51) Why / Why not?

_____

52) It does not require a lot of effort to do production releases

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

53) Why / why not?

_____

54) Our production releases are seamless for our customers.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

55) Why / Why not?

_____

Our release and deployment management process...

		Don't Know	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
56)	...supports our way of working in Scrum	○	○	○	○	○	○
57)	...makes it easier for me to follow Scrum	○	○	○	○	○	○
58)	...is well integrated into our agile way of working.	○	○	○	○	○	○

59) Thoughts?

_____

60) I think we are good at using feature toggles

- ○ Don't Know
- ○ Strongly Disagree
- ○ Somewhat Disagree
- ○ Neutral
- ○ Somewhat Agree
- ○ Strongly Agree

61) Why / Why not?

_____

62) I think feature toggles are an integral part of our Release and Deployment Management

- ○ Don't Know
- ○ Strongly Disagree
- ○ Somewhat Disagree
- ○ Neutral
- ○ Somewhat Agree
- ○ Strongly Agree

63) Why / Why not?

_____

64) I think Content Management hinders our Release and Deployment Management

- ○ Don't Know
- ○ Strongly Disagree
- ○ Somewhat Disagree
- ○ Neutral
- ○ Somewhat Agree
- ○ Strongly Agree

65) Why / Why not?

_____

I think our Release and Deployment management tools are..

		Don't Know	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
66)	...easy to use	○	○	○	○	○	○
67)	...offer valuable knowledge	○	○	○	○	○	○
68)	...up to date	○	○	○	○	○	○
69)	...fully utilized	○	○	○	○	○	○

70) Thoughts?

_____

71) I think we have good levels of automation.

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

72) Why / Why not?

_____

73) I think our automation is reliable.

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

74) Why / Why not?

_____

75) We should modify our release and deployment management process.

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

76) Why / Why not?

_____

Closing comments

77) I think we are mature in release and deployment management.

○ Don't Know
○ Strongly Disagree
○ Somewhat Disagree
○ Neutral
○ Somewhat Agree
○ Strongly Agree

78) Why / Why not?

_____

79) I would recommend our release and deployment management process.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

80) Why / Why not?

_____

81) I think I am experienced in release and deployment management

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

82) Why / Why not?

_____

83) I think our team relies too much on a few persons to have knowledge about our Release and Deployment Management

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

84) Why / Why not?

_____

85) I think we should allocate more resources to our release and deployment.

- ◯ Don't Know
- ◯ Strongly Disagree
- ◯ Somewhat Disagree
- ◯ Neutral
- ◯ Somewhat Agree
- ◯ Strongly Agree

86) Why and where? / Why not?

_____

87) How much time do you spend on release and deployment management per day?

- ◯ less than 30 minutes
- ◯ 30-60 minutes
- ◯ 1-2 hours
- ◯ 2-4 hours
- ◯ 5-6 hours
- ◯ 7-8 hours

88) Is there else regarding release and deployment management, that you would like to address that is not covered in this survey?

_____

89) Comments / feedback