MASTER'S THESIS 2024

Improving feature discoverability in continuously deployed software products

Alfred Langerbeck, Love Sjelvgren

ISSN 1650-2884
LU-CS-EX: 2024-13

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-13

Improving feature discoverability in continuously deployed software products

Påverkan av kontinuerlig leverans på synligheten av nya funktioner i mjukvaruprodukter

Alfred Langerbeck, Love Sjelvgren

Improving feature discoverability in continuously deployed software products

Alfred Langerbeck
98alla02@gmail.com

Love Sjelvgren
love.sjelvgren@gmail.com

March 7, 2024

Master's thesis work carried out at Schneider Electric.

Supervisors: Lars Bendix,lars.bendix@cs.lth.se
Carl Serrander, Carl.Serrander@se.com and Åsa Nilsson, Asa.K.Nilsson@se.com

Examiner: Per Andersson, Per.Andersson@cs.lth.se

Abstract

As software teams increasingly adopt methods for continuously integrating and deploying their products, it is relevant to consider which problems can arise as a result of these developments methodologies. Schneider Electric in Lund develops a product called Building Advisor, which is a web-based platform to help maintenance and service buildings. The development team works with continuous integration and continuous deployment. Because of the choice of development methodology, the case company has theorized that users miss the frequent and small updates which are deployed continuously. The subject is not widely studied because the problems arise on the user side and not the development side, making it much harder to research. The primary aim of the thesis is to explore if feature discoverability suffers when working with rapid and small releases. The second goal is to investigate a possible way or guidelines for automating information regarding what features have been released and how they affect the user. Finally, the thesis proposes a way that generated information about features can be displayed to the user in a non-intrusive way. The thesis follows the methodology of a case study and gathers information about how users interact with the program and how the developers could change their workflow to increase the visibility of features. Information is gathered from previous research in a literature study as well as interviews targeting people in specific roles at the company. Observations and literature show users are not using features that are deployed without any marketing or notification. Anonymous data from the users interactions with the website also shows that users interact far more with embedded user information regarding features in the program rather than information the more extensive documentation stored off site. To be able to generate just enough information about a feature, important metrics are established such as accuracy, consistency and timeliness. Information that follows these recommendations should help users find and utilize newly released features. A design proposal to show information generated in this way is presented, which could increase the usage of the features that are deployed continuously. Combining these findings will help companies and organizations that are seeking to reduce release problems when working with small and frequent releases, such as in continuous deployment.

Keywords: CI/CD, Feature discoverability, Release problems, Embedded user information, Visualizing small improvements in software

Acknowledgements

We would primarily like to thank our supervisor Lars Bendix for the extensive time he spent on input and feedback during the entire process of the thesis. We would also like to thank Carl Serrander and Åsa Nilsson, at Schneider, who graciously helped with most of our investigating the work performed at the company. Finally we would like to thank the entire Building Advisor team who agreed to meetings and interviews which gave great input to the resulting in this thesis.

Contents

1 Introduction **7**

2 Background **9**
2.1 The background of the case company . . . 9
2.2 Research questions . . . 12
2.3 Methodology . . . 12
2.3.1 Research question 1 . . . 13
2.3.2 Research question 2 . . . 13
2.3.3 Research question 3 . . . 13
2.3.4 Case study . . . 14
2.4 Theoretical foundation . . . 15
2.4.1 DevOps and CI/CD . . . 15

3 Feature discoverability in continuous deployment (RQ1) **17**
3.1 Data Collection . . . 17
3.1.1 Interviews and meetings . . . 18
3.1.2 AppInsights data . . . 19
3.1.3 Literature . . . 19
3.2 Analysis and Discussion . . . 21
3.3 Result . . . 23

4 Documentation & information in rapid development (RQ2) **25**
4.1 Data Collection . . . 25
4.1.1 Interviews . . . 26
4.1.2 Literature study . . . 28
4.2 Analysis, Discussion and Observations . . . 29
4.3 Results . . . 31

5 Designing a solution to the users' problem of feature discoverability (RQ3) **33**
5.1 Data collection . . . 33

5.1.1 Literature 34
5.1.2 AppInsight 37
5.1.3 Interview 38
5.2 Analysis and Discussion 41
5.3 Design specification 43

6 Discussion and Related Work 45
6.1 Reflection of our work 45
6.2 Threats to Validity 47
6.3 Generalization of results 48
6.3.1 Related work 1: On the journey to continuous deployment: Technical and social challenges along the way 48
6.3.2 Related work 2 : DevDocOps: Enabling continuous documentation in alignment with DevOps 50
6.3.3 Related work 3 : Identifying Characteristics of the Agile Development Process That Impact User Satisfaction 51
6.3.4 Related work 4: Improving Discoverability of New Functionality . 52
6.4 Future Work 53

7 Conclusion 55

Appendix A 63

Chapter 1
Introduction

Schneider Electric is a global company working in many different areas, however the thesis is written in a software development team in Lund. The software team in Lund works with a product called Building Advisor which is used by service technicians of building complexes. Their software solution is cloud based and allows the building administrator to reduce energy consumption, improve occupant's working conditions, helps with maintenance of hardware and detect when and where alarms are triggered. It is able to do all these things through monitoring a wide range of data points and analyzing the result. The product has users all around the world and is used by experienced service technicians and building administrators. Schneider Electric will henceforth be refered to as the case company

The team in Lund working on Building Advisor is heavily committed to working with the development methodology CI/CD. This means that the code base is constantly being updated. This methodology has various benefits for the development team. Making small incremental changes creates a faster feedback loop and reduces conflicts when integrating and deploying software. However, this also creates a problem for the users. When the program is changing daily or weekly with only small and incremental changes, it is hard to notice them.

The case company has noticed that users don't use the features which are implemented, even though ideas for features are gathered from focus groups comprised of end users as well as through forum suggestions. This is the basis for the first research question: To establish why users are not using newly integrated features. If users do not find the features that the development team have released, the value created is lost. Also, users might find it frustrating that the improvements they want in the program never get realized, even though they might already be.

With an indication or understanding of why users do not use the features that are being developed, the focus will shift to how it can be fixed. To be able to inform users of new features, some information text will be required. This text can be displayed to users to show them what has been released, which leads to the second research question: Find a way to continuously create documentation useful for end users. To not harm the speed of development that is a driving factor to working with CI/CD it is important that as many steps as possible

can be automated. The report will give suggestions on how information can be automatically generated with as little added overhead to the developer as possible. It will also provide what the information should include and which attributes are important for the information.

After understanding of how the users interact with the program through research question 1 and generating user facing information in research question 2, these two will be combined in the third research question: Combining the results from the previous goals to implement alternatively suggest a solution to the users problems, or to advise changes in the development process. With the knowledge of why users don't use newly integrated features, and a way to automatically generate information that could help users find the new features, the goal is to present that information in an intriguing way or advise changes in the development process. The combined results of all three research questions will help alleviate the initiating problem.

The thesis will follow the model of a case study because the research method is suitable for private corporations, project oriented cases, and it is investigating the process around a team of software developers. Also, the case study works well with exploratory research, which will be the case for the different research questions.

The thesis is divided into seven chapters, starting with a more thorough Background of the case company leading to the initiating problem. Also in the background section there is a more extensive argumentation for the choice of research method as well as the theoretical foundation which the thesis builds upon. Chapters 3 to 5 will include data collection, results and discussion for each of the three different research questions. After that the sixth chapter will contain a discussion and reflection on the work, threats to validity and as well as some related works. Finally, chapter seven will present the conclusions of the thesis.

Chapter 2

Background

The purpose of this chapter is to describe the background of what the case company works with, how they work, the problem they are facing and what they have noticed in regards to their choice of development methodology. This will lay the basis for understanding the initiating problem, which will lead forward to the research questions we will be investigating. The sections are divided into three parts, starting with the problem at the case company in 2.1. Section 2.3 will cover the methodologies used to research the problem, and section 2.4 will cover the theoretical foundation on which we will expand.

2.1 The background of the case company

The case company develops a web platform for monitoring that leads to analyses which provides insights about the buildings. It also helps service technicians when servicing buildings. The application is called Building Advisor and collects data from various sources within the building, summarizes the data to help service technicians to replace parts that have or will fail soon. The application is also used to create tasks for service technicians on what has to be done with specific buildings. It can also be used to generate reports that show how much money the energy improvements have saved. The product is sold and marketed to owners of big buildings to decrease maintenance and variable costs. The users of the product are specialists in the field of building maintenance.

The application is web-based and figure 2.1 is a screenshot of the interface of the test environment.

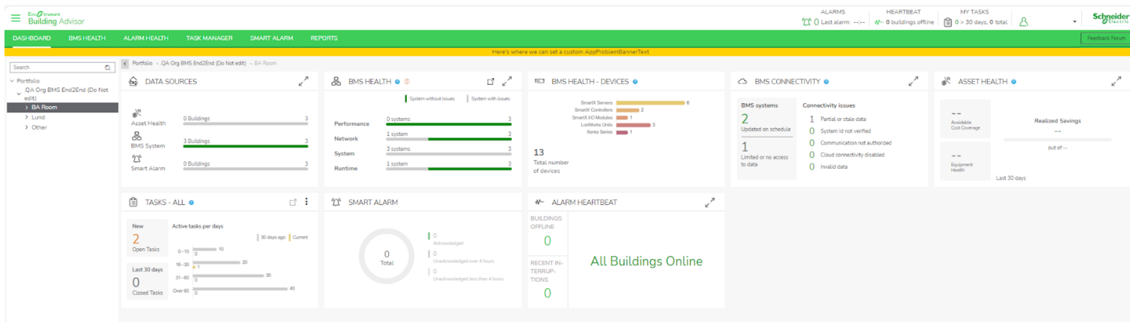


Figure 2.1: The program contains multiple different windows, the one depicted in the image is the home screen.

The case company develops software and uses a DevOps in combination with CI/CD which leads to frequent deployments to the production build every day. The infrastructure that is used to support the pipeline is Microsoft's product Azure DevOps. Microsoft also provides tools for tracking how users click around on the website. This is implemented through creating customEvents, they are simply trackers that count how many times users have interacted with the graphical element associated with the customEvent. They are called customEvents since the developers has to add them manually to track parts of the program that they want data on. When implemented they can track clicks on or hovers over specific elements on the website. This information can then be collected and combined into AppInsights data that can give an understanding of how users interact with the program. While the software product is continuously deployed, there are big biannual launches that get support from the marketing team in order to highlight new features. In the daily deployments, the case company has no way to inform users about the changes which are deployed every day.

The current documentation process is sequential to the development process, with technical writers working on a backlog of features. When the documentation is finished it is released to a help portal that is connected to the website through a hyperlink. Technical writers do not get involved in the process until a feature is already in the production build. There is however some information being released together with the features, this is collected and shown in help bubbles in the application.

The two different alternatives for information and documentation can be seen in the figure 2.2.

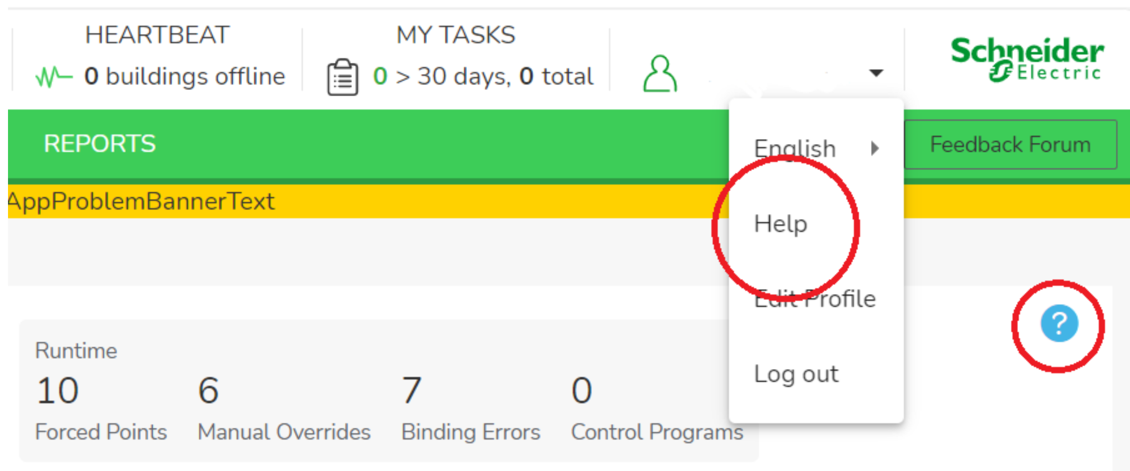


Figure 2.2: A screenshot from the test environment of the program to showcase the two kinds of information available to the users.

When documentation is finished by technical writers, there is a big release and marketing push by the case company. After marketing pushes, utilization of features which are already available, but not advertised, increases according to the team. And it is this problem that the case company has seen and wants to improve on. Showing that there is some kind of disconnect between the user's knowledge, and the frequently deployed features.

The team believes that the high frequency of deployments have numerous benefits. But since changes are not announced until one of the biannual launches every year, the customer receives no perceived improvement in development time from the continuous deployment that the company utilizes. This mix between CI/CD and traditional biannual releases might stem from the fact that the team developing the software wants to work with CI/CD because of the benefits it grants them, however the company as a whole is still stuck in a traditional release schedule.

Another related problem is that the case company has noticed that features only get one marketing push. This strategy of informing users which is very incremental clashes with the iterative process that is used to develop the software. All this means that features which might have been released in a suboptimal state and then continuously improved might never get the users' attention. An example could be that a feature gets released with a marketing push and documentation, but the feature contains bugs that make it flawed for the users. If the feature gets improved and bugs are removed, the users will never find out about it. Another issue is that users of the product are very process oriented, meaning that users often already have a workflow that they are satisfied with. This makes it harder for the users to pick up new features since they are easily stuck in old habits. Users start the program with an intent to perform a specific task or workflow, the action can be time-sensitive which leads to the users just performing it the way that they already know and not exploring the program more than necessary.

This leads to the initiating problem that the case company users are not aware of changes in the product.

The case company is determined to continue to work with CI/CD since they see big benefits from it. So the solution must be applicable to the pipeline and development strategy in use. From the case company's position, the ability to automate integral steps of the develop-

ment process is the main strength of their CI pipeline and the last piece of the puzzle would be to notify users when changes have been deployed ahead of the larger releases (or at least big announcements of features). However, automating this notification of users will probably require more overhead for developers to write usable information regarding the merged feature.

In combination with the implementation of CI/CD the case company also uses agile concepts when describing what should be implemented. This means that things that should be implemented is divided are categorized as epics, stories and tasks. An epic is divided into many stories, and stories are divided into many tasks. Stories can also be classified as user stories if they are something that will be visible for users.

2.2 Research questions

Considering the background and initiating problem, the following research question were constructed:

- RQ1 : **Why are users not using newly integrated features?** We want to determine where case company's problems stem from. The case company has hypothesized that the greater problem stems from the fact that deployed features are not promoted enough and are therefore not being discovered by users. Investigating if this is one of the reasons as well as other potential causes should be the primary goal of this project.

- RQ2 : **How can documentation that is good enough for users to utilize features be continually produced in a CI/CD environment?** During the development process, some user-facing summary or documentation needs to exist in order to notify users of continuously integrated features. To what extent and how this process should work needs to be investigated. By utilizing information in code, literate programming and information from developers to generate documentation in a continuous way in order to reduce the current delay in the process.

- RQ3 : **How can the results from the previous research questions be used to implement or suggest a solution to the user's problems, or at least to advise changes in the development process?** After doing an investigation into deeper causes for the disconnect between the state of the program and user's knowledge of the program, changes should be implemented accordingly. While taking heed to information learned from the initial investigation, changes should be suggested or implemented to the notification, release and/or integration process.

2.3 Methodology

The research questions in this thesis can be generally divided into two parts, an investigation into a problem appearing in the case company and investigation and design of a solution. This necessitates sequential work on the three research questions with different methodology.

2.3.1 Research question 1

To establish why users are not using newly integrated features, a case study is conducted into the product and development/release process of the case company's Building Advisor Team. The case study follows the general guidelines laid out by Runeson and Höst [1] with the objective to answer research question 1.

The study will mainly be exploratory and explanatory, with the later RQ aiming to be improving. The main goal is to provide the necessary information to answer the two later research questions, but also to evaluate general problems with the case, that being the product and the processes that the Building Advisor team works with. The five stages of a case study as described by Runeson and Höst [1] is case study design; preparation for data collection; collecting evidence; analysis of collected data and reporting. The case study design phase is largely overlapping with the planning stage of this thesis and is concluded with the creation of research questions and a general start up interviews at the case company

Data collection for the case study will use most of the sources mentioned in Runeson and Höst [1] these being interviews, observations, archival data and metrics. This means preparing for the data collection consists of finding what data (metrics, archival data) there is to use, planning meetings and interviews with relevant personnel at the case company (interview) and finally explore the processes and system that are interacted with in the team (observation).

Then data is compiled and analyzed with the intention of answering RQ.1 and giving background and information to help answer RQ. 2 & 3.

2.3.2 Research question 2

While RQ.1 is almost entirely explanatory/exploratory, RQ.2 & RQ.3 seeks to propose solutions and/or designs to solve a more specific problem. While methodology included in the case study will be used, early literature studies suggest there has been previous research that has been done strongly relating to the question, therefore options will be considered and discussed in interviews with relevant experts at the case company. Possible process and pipeline changes will be proposed and reviewed in meeting and interview form.

Since both RQ.2 & RQ.3 seek to implement or at least propose a design, one considered approach is research through design. Research through design often used in human computer interface [2] and while this mostly relates to RQ.3 the lessons learned can be used to strengthen the result of study around this research question.

2.3.3 Research question 3

Combining the results from the previous goals and implement a solution to the user's problems or to advise changes in the development process, the methods applicable to the previous questions are of course also relevant to RQ3. To answer RQ.3 literature studies into previous UX studies will mostly be used for design proposals since the goal is not necessarily to research the visual component of a solution.

The case study will once more be used to study what kind of implementations can be made and what changes need to be introduced in the development and release process to solve the initiating problem. The case study will reach a stage where the goal is reaching

toward the improvement of the product. To evaluate the result, feedback from experts as well as other stakeholders will be gathered. The proposed solution will also be analyzed and assessed with what could be done differently and how the results could improve the general process of development in an environment with continuous deployment.

2.3.4 Case study

Case study is considered a suitable research methodology for software engineering according to Runeson and Höst [1]. The methodology might not generate as many conclusive results as controlled empirical study methods, but has shown to deepen understanding of a contemporary phenomenon under study. A case study is especially suitable to software development projects when the following characteristics are met [1]:

1. The study objects are private corporations or units of public agencies developing software, rather than public agencies or private corporations using software systems;

2. Project oriented rather than line or function oriented;

3. The studied work is advanced engineering work conducted by highly educated people rather than routine work.

These characteristics are found in the case of the case company's Building Advisor product. It is developed by a private corporation. It is a project and process oriented case. Finally, it is a software engineering project created by engineers.

While there are other alternatives to the case study that are relevant to consider, such as proper surveys, experiment, and action research, the scope of this thesis makes these methods harder to wholly use in the timeframe. Extensive surveys have two problems. Firstly, the time it would take to conduct the survey exceeds the time allotted at the time of writing. Secondly, the case company did not think it relevant or possible to survey users of the Building Advisor product. Experiment and action research for this software project would require an implementation to be created, deployed to some user, complemented by pre- and post-studies; something that also was not considered possible within the time frame due to release schedule and the unavailability of users.

However, since the goal of the thesis is two-fold, to answer the theoretical research question and secondly and to implement or at least suggest an implementation from the answers to the research questions, parts of the experiment and action research methods are still relevant since an experimental result is viable at a small scale and the work is definitely "change oriented".

Runeson and Höst [1] has stated four different purposes for research based on previous articles: exploratory, descriptive, explanatory and improving, some of which were earlier mentioned in relation to the research questions. Case studies are primarily designed for exploratory purpose [3], which works well with the initial questioning as stated above. When using case study as a method for explanatory research, one has to bear in mind that isolation of factors might become a problem. Finally, Runeson and Höst [1] argues that software case studies are often improving, as is the case in this thesis.

2.4 Theoretical foundation

To establish a theoretical foundation for the research questions in this thesis, some specific concepts should be brought up relating to the fundamentals of Continuous integration and continuous delivery, as well as concepts of the development and release process in general. Several prior works have discovered problems in the implementation of certain parts of CI/CD that seem to apply to the case company.

Establishing the problem stated in the initial research question will be done in the context of the case company and their users and developers, while the second and third can more easily be found in literature in conjunction with information from developers.

2.4.1 DevOps and CI/CD

In the beginning of software development the development strategy was waterfall. The big advantages of waterfall development are that everyone from developer, technical writer and management knows what is going to be done in every step. However, waterfall development comes with big disadvantages as well. Having a rigid process, as shown to decrease product quality when compared to iterative processes [4]. There are many development strategies that help to solve some of these problems, one is DevOps. DevOps is a mentality that tries to bridge the gap between the development side and the operations side. This gap is caused because the two sides do not necessarily share the same goals [5]. The development side wants to continuously improve on the product by releasing new features quickly. The operations side however wants a stable product since new releases might break the product. For the operation side, it's more important that the product is stable and reliable [5].

One way that DevOps can narrow this gap between development and operations side is by allowing developers to continuously improve on software while assuring the operators that it won't compromise the software required for daily operation through thorough automated testing. This in combination with giving the development team full responsibility over the product. This leads to faster and higher quality of development [5].

To break it down Continuous Integration strives to keep the code repository up to date by frequently merging changes in code. Each developer is responsible to continuously integrate the work done. An important step in this process is running the code through a pipeline that runs a suite of tests to uncover any problems that might arise [6].

The other part of CI/CD is Continuous delivery, depending on the application it can also be called continuous deployment. The main concept is that the software should be in such a state that it can be delivered to users at any time. This allows for updates in software to reach customers quickly, reducing time to market [6].

Chapter 3

Feature discoverability in continuous deployment (RQ1)

In this chapter, the first research question will be addressed. To understand and later solve the problem, a deeper understanding of the context and problem statement will be achieved by investigating and analyzing possible causes of why users aren't using features when they are continuously delivered. The purpose of the investigation is to find possible root causes that could stem from the team's adoption of continuous deployment, its implementation thereof or other possible causes. While there can be many reasons for users not to use newly implemented features, the users of the program were hard to get to and therefore finding data and information to find causes of the initial hypotheses of the team and answering RQ.1 is the main focus of this chapter. The result of this chapter will give an understanding of possible causes of the problem, which can later be used to help solve it.

Investigating this initial research question follows a three-pronged approach. Initially, meetings and interviews with stakeholders at the company will be conducted. These will give an understanding of how the development process works and how users experience the program. Anonymous data of the users' interactions with specific features will be collected. This data will be used to give us an indication if features developed with the CI/CD process are less utilized. And finally, a literature study to gain a more general knowledge of what issues might exist in CI/CD environments.

3.1 Data Collection

Data to answer the question about why users aren't using new features was collected through three primary methods: initial meetings and interviews with stakeholders were used to identify and gather thoughts about the initial problem statement, working as a hypothesis generator. Application insight data, containing information about feature use, were used as a secondary data source to strengthen observations made by stakeholders. Finally, literature

will be used in conjunction with the observations, to support the validity of the stakeholders' observation and identify problems that could cause users to not use or discover features, especially relating to continuously deployed features.

3.1.1 Interviews and meetings

It was decided to have meetings with the project owner because of the central position the project owner has in both the development and contact with the users. From these interviews we want to get information about how the users experience the product, how the development cycles work and an overall impression of the product. The meeting with the offer manager was conducted to get a more in depth understanding of the user's requirements and preferences. Also, because the offer manager is in place in between R&D and sales makes him responsible for more general customer relations as well as deciding what features should be developed and why. Developers were interviewed because in the end it's the developers that are working on the features that are being underutilized and might have valuable insight into the state of the program.

The project owner shared experiences from meetings with customers where they weren't aware of functionality that was already available. The understanding of this problem increased when the application was reviewed with the project owner. It was made clear that there was no place for users to find what was new or what had changed for the continuously deployed changes. There had been smaller patch notes in the past, but they were removed when development changed to more continuous methods, due to a perceived notion that they were not used and not worth the effort at the time. Another issue that the project owner saw was that big features only get one big release with marketing. This becomes problematic when features are released, get a big marketing push, and then the feature is continuously improved and expanded. If a user tries to use a feature but encounters difficulties, then there is no way to notify them that issues are fixed or that the functionality they wanted is implemented.

The project owner also shared that the case company previously posted patch notes that were available to the users, but they were manually created and posted on the web help site. These patch notes were discontinued because they didn't fit with the CI/CD process, since there are many small patches. Without any patch notes, users are left with no clear way to find information regarding new features or changes, outside the biannual releases.

The offer manager said that there is no continuous stream of information from the development team and the users. They talked about the three ways of deciding which features were going to be developed. Firstly through community feedback which users could submit in a suggestion forum. Secondly, focus groups of active users where suggestions for new features were tested. And thirdly, if a feature had a high impact business case.

Another issue that was raised in the meetings with the offer manager and project owner is that the users of the product are very process oriented, as mentioned in chapter 2. Which makes it difficult for them to pick up new features into their predetermined workflow. When they learn the product in the beginning, they get used to using certain features, and they are reluctant to pick up new functionalities since what they already have works. Even if new features might improve the workflow.

The lack of a stream of information was something that was also confirmed when developers were asked interview question 1 from Appendix A, they didn't feel like there was a communication channel between them and the users. However, if they would like one the de-

velopers had different answers, some saw it as beneficial while others feared it would hamper productivity. Another issue that the project owner raised was the fact that the documentation was lacking behind the development of features.

In discussion with developers and product owners, it was made clear that the team was very happy with the impact of CI/CD on their development process, but that it did not necessarily integrate well with the company's general release process. This goes back to releases being communicated with bigger marketing pushes and learning material being created in conjunction with bigger releases, which means there needs to be a build up to feature releases. This seems to be due to the release process adopted by the rest of the company. The ability to communicate and integrate small changes well would be "the last piece of the puzzle" in their implementation of CI/CD processes.

3.1.2 AppInsights data

The case company has anonymous data from the user's interaction with the program. By analyzing the data, it's possible to get an understanding of how the users utilize the features of the program. Finding out how users interact with parts of the program can give us hints of what the problem is.

One feature that received a big release at the end of the summer is called "Tasks". Tasks are supposed to help building administrators create specific tasks that can be divided between the building technicians. The second feature is something that has been in the product for a long time and is called "Reports" which is used to generate reports about the building such as the building's energy consumption and what improvements have been performed. These two features were chosen because they are two active actions that aren't done by mistake. This comparison is also done since "Tasks" is a newly integrated feature that is continuously improved upon. The results is shown in 3.1

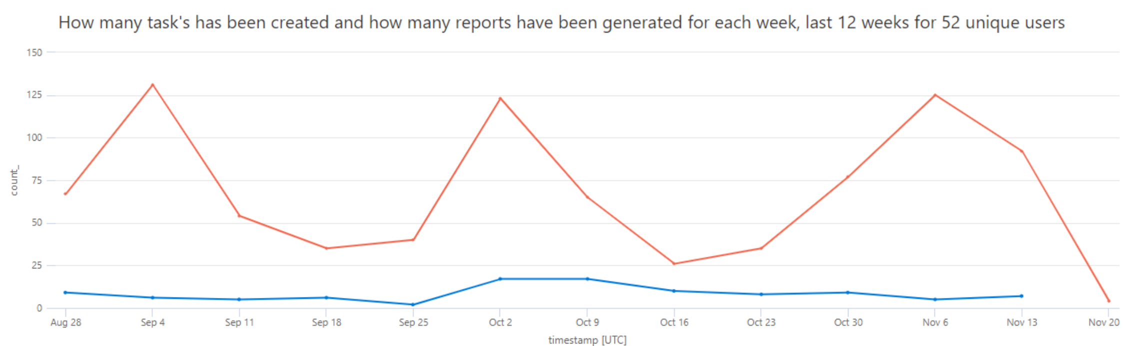


Figure 3.1: Number of tasks created (blue) and reports generated (red). The number of tasks created and reports generated are summarized on a weekly basis. Data gathered from the period 28/08/2023–20/11/2023.

3.1.3 Literature

Finding cases in literature with a similar problem statement is not hard, most companies will likely want to increase the use of features that they believe will improve the workflows of their

user base, and therefore increase "the value" of their product. Eriksson [7] presents a similar case to this one. Spectra is a company developing a Picture Archiving and Communication System (PACS) which is a professional tool used by radiologists. While the tool is not a continously deployed, almost"versionless", program or a web based application, the program has similarities in that it is a tool used in a workflow by professionals. The company which is studied by Eriksson [7] concludes that feature discoverability is low in their product and are looking for ways to increase it. While the study focuses on how to notify features, the initial problem statement shows that the feature discoverability is considered an important problem to solve by other companies.

While the cause of the problem can be due to many reasons, the case company believes it to be due to their rapid delivery of features and lack of notification of said feature. This indicates a belief that one of the causes is their implementation of CI/CD or an inherent problem caused (or at least not solved) by CD. When looking at common problems in CI/CD implementations found by Laukkanen et al. [8], who aggregates data from studies on the implementation of continuous delivery, the following categories are found:

- **Build design problems:** Problems pertaining to complex och inflexible build pipelines and process.

- **System design problems:** Problems pertaining to the architecture, dependencies and database limitations or stability.

- **Integration problems:** Problems often related to version management, integrating code into branches, slow maintenance and approval of changes.

- **Testing Problems:** Problems relating to testing or inability to test code, as well as complexity and sluggishness of testing.

- **Release Problems:** Problems stemming from rapid release of a program from a release perspective. Problems keeping up with rapid deployment from dev, marketing, user and documentation perspectives.

- **Human and Organization problems:** Problems relating to willingness and motivation to perform tasks and practices of CI/CD as well as teams restructuring and coordination.

- **Resource problems:** Problems relating to the initial pressure of setting up continuous delivery, along with insufficient hardware and network resources.

From the initial interviews above, the most relevant category seemed to be Release Problems, since it contained problems which were brought up in the interviews. Release problems brought up by Laukkanen et al. [8] are presented in figure 3.2.

Table 10
Release problems.

Problem	Description
Customer data preservation	Preserving customer data between upgrades.
Documentation	Keeping the documentation in-sync with the released version.
Feature discovery	Users might not discover new features.
Marketing	Marketing versionless system.
More deployed bugs	Frequent releases cause more deployed bugs.
Third party integration	Frequent releases complicate third party integration.
Users do not like updates	Users might not like frequent updates.
Deployment downtime	Downtime cannot be tolerated with frequent releases.

Figure 3.2: Summary of common release problems [8]

Laukkanen et al. [8] does elaborate that it is hard to research release problems since "release problems might be external to the development". It finds that few research papers that have found solutions to these problems, especially feature discoverability. Claps et al. [9] brings up the feature discoverability specifically, stating that "CD enables software products to be constantly updated, but it does not assist in introducing these updates to customers", going as far as stating that it is one of its interviewed team's biggest challenges.

Three of the problems identified seem to be especially relevant to the case company.

- Customer feature discovery, where customers might not notice the newly added features.
- Product marketing: Marketing of CD or versionless products require alternative marketing strategies.
- Documentation, where documentation needs to be in-synch with the released version of the product.

Both of the papers bring up the existence of the problem without providing solutions. Since the papers are mapping studies, this is making it clear that CD's impact on the problem has been identified, but requires further research to solve. One problem that seems relevant is that "Users do not like the content of the updates" this is very contextual to the feature and company. The cause could apply, but is a hard case to investigate without access to users.

Outside the [9] release strategy indicates that to successfully adopt the CD process in an organization, a company-wide effort must be made. If wider management is not directly involved, conflict in direction of team workflows can cause problems. The changes needed to make CD work well touches on many things in the development cycle and are hard to push from a product team.

3.2 Analysis and Discussion

Initial interviews suggest that there can be a problem with rapidly released features not being used since they are not notified as in traditional releases. Literature seems to support that,

feature discoverability is a problem and this type of release process. The App Insights data also points towards newly integrated features getting less usage than older features.

During the initial interviews, both the project owner and the offer manager agreed that there was an issue of users not picking up the new features. Since features came from suggestions or focus groups of users, the features that were developed should reasonably be sought after by the users. With a lack of any kinds of information about new features in the platform, combined with the absence of a communication channel between developers and users. It's not hard to understand why users don't find the new features that get released. If designing a solution to show changes, the forum that the offer manager mentioned might be interesting. The developers concluded that previous efforts to notify users when more traditional development was used would not work with CD. This is attributed to the rapid integration of code and incremental style of development. The topic of documentation and how it is presented will be explored in chapter 4 and 5

After the review of the program with the project owner, it's clear that the application is large, with many sub-pages. Even for a regular user, it can be hard to notice that anything has changed. This is something that need to be further explored when broaching the topic of how to notify users of changes to the program.

The diverging answers of developers concerning if they want a one way communication channel are maybe because some understood the question in a way that meant that users could contact the developers directly. The ones that were negative towards this thought this would hamper production and would lead to users contacting developers directly with requests for new features. When it was made clear that it would be a one way communication channel from developers to users, more developers were interested in having a stream of communication where changes and new features could be communicated. If such a stream of information is implemented, it is important to investigate if it is actually used. After implementation, data could be collected, monitoring feature use of before and after notification or if notified features are used in a higher capacity than features which are not notified. This would further support the hypothesis and increase the relevance of the results, showing that this directly increases use of feature use or if other causes are more relevant for further study.

The fact that users are process oriented and are reluctant to change their workflow increases the importance of showing that new features are available and how they can improve the everyday operations of the users.

Moving on to the analysis of AppInsights data, the figure 3.1 shows the usage of two different functions on the website. It's clear that usage of "tasks", which is a new function, is constantly lower than the old "Generate report" feature. Even though creating tasks should be something that can be done many times a week but generating reports is usually done a few times every month. A reason why "Task" are being less utilized than "Generate Report" can be that the feature was released without functionality that users wanted. When functionality has been added such as sorting the list and widget to show which tasks there are, users won't find it. Unfortunately, the way that AppInsights data is configured at the case company right now only allows data to be stored for 90 days. Therefore, we can't check whether there was an influx of users at the time of release.

Previous literature makes it relatively clear that feature discoverability is a problem that multiple companies want to tackle. It seems that feature discovery can definitely be a problem when implementing continuous delivery, and many of the problems brought up in the initial interviews seem to be problems that can be found in the studies which document common

CD problems. The three main ones which can be found in both interview and literature are customer feature discovery, product marketing and documentation problems. Most of these problems seem to stem from a relatively quick process change, where parts of the process have not caught up. It seems that most of the things brought up by the developers are categorized as release problems. Stakeholders in the team are very happy with the changes done to implement CD and emphasize that a solution like batching, where changes are integrated continuously but released more traditionally, is not relevant to the team, due to the perceived hindrances to the team's development process. The team believe that CI/CD is integral to prevent pressure from building up close to a biannual release, and continuously integrating relieves the importance and therefore stress of the big releases. They also state another value intrinsic of CD, which makes sure that the product is always deployable. Batching to them would, in their mind, negate the usefulness of CI/CD. Other companies do "batching" or "dark features", which in fact the team at the case company also does for larger features by only enabling feature flags when they have the big biannual releases or for early adoption customers. In the current setup, only smaller features and bug fixes are actually continuously deployed.

To successfully adopt CI/CD it seems imperative as the literature seem to indicate that a company-wide effort must be made. While it seems that team experience and willingness to adopt plays an important role, it cross-team collaboration often requires a push from management. Which seems to be a cause for concern in the case company, since it is the part involving multiple units of the company that the team find don't work in the current system.

Some other process problems appearing earlier in the development process that could be possible causes, is decided to be outside the scope of this work either because the team at the case company is content in their process and are not going to change them or because a root cause is hard to investigate, as with customer adoption, when the end customer is not available.

While the problem of feature discoverability is relevant to most products, it seems that CD can definitely worsen the problem and solutions need to be consideration in the context of CD and of the case company's implementation of it.

3.3 Result

The initial hypothesis of the developer team, that users don't use newly deployed features that are continuously deployed, seems to be correct. The possible root causes brought up by stakeholders strongly correlate to the problems one can encounter when adopting continuous deployment. Two are directly mentioned, documentation being out of sync and general feature discovery. Other causes are more alluded to. Observing the process and corresponding literature also identifies that most successful CD implementations require alternate marketing strategies and should be management driven in its adoption. The last cause relates to problems with features themselves and the nature of the rapid releases in CD.

The organizational problems like marketing and adoption on a larger scale would require further study with multiple teams and more work finding strategies for marketing in CD. Similarly, the problem with users not liking the released features would require further research with access to the specific users who use the application. Since users are ostensibly a part of choosing which features are developed, this might bear no further solution, but might

be worth exploring for the team. For users who do not like the rapid release schedule of CD, batching can be implemented. Seemingly not relevant to the team at the case company, this can be of use in a general setting.

The two remaining problems that were identified are of a nature that changes can probably be made at the team level to solve the problem. This makes them viable for further investigation within the scope of this work. How to increase feature discoverability is therefore a problem handled in later chapters. The documentation problem is in itself a problem, but can also have relevance to an increase in feature discoverability. Since some solutions to increase feature discoverability requires documentation or at least some information to be generated, along with the feature. Therefore, the problem relating to documentation and information in CD will be more thoroughly examined in the coming chapter.

Chapter 4

Documentation & information in rapid development (RQ2)

This chapter will further investigate the documentation issues that were found in Chapter 3. The goal is mainly to find solutions to these problems by answering this work's second research question, **"How can documentation that is good enough for users to utilize features be continually produced in a CI/CD environment?"**. This has the goal to initially solve the problem that documentation is lagging behind, due to the current setup with CI/CD. Later, these insights can help tackle the problem that users aren't finding new features or finding, but not using them. Since some solutions to the latter problem might require information about deployed features, this chapter also creates groundwork for solving RQ.3. The research in this chapter will be based on interviews with developers, getting their perspective on possible process changes and the technical writer who is currently responsible for the documentation. A literature study will also be conducted to find possible solutions that have been implemented at other companies, as well as highlighting some important aspects of documentation.

In this chapter, the word documentation is used to describe user-facing documentation currently displayed in the product's web help page. Information is used to describe shorter documentation that adheres to a "just-enough" mindset, containing minimal information to use a feature.

4.1 Data Collection

Since the documentation and information gathering process need to appear in close relation to the development of a feature, interviewing developers and technical writers seems obvious. Developers can present intimate knowledge of the developer process, while technical writers are the de facto documentation writers at the case company. Interviewing both parts will give useful insight into producing documentation/information, where information can be

created in the process, and how the work can be connected between writing documentation and the development of features.

To complement the information obtained from employees at the case company, two other sources of information were used. Literature was used to get insight into previous solutions to similar problems and to further understand what could cause it. Finally, our own observations about the team's development pipeline and process were used.

4.1.1 Interviews

Interviews were performed with seven (N=7) developers in the building advisor team, each developer having a slightly different role. The questions that were asked to developers can be found in Appendix A. In general, developers have a rather small part in the user facing documentation process. They often write some small technical documentation for themselves and other developers, which is sometimes stored in the developer wiki. This information is not at all user facing. After a feature has been completed by a developer, they sometimes check that the technical aspects of the documentation, written later by technical writers, are correct.

It was argued by several developers that for most of the implemented changes, such as backend and database changes, there is no need for documentation and that the documentation would not be read anyway, due to its irrelevance to the user. One part that the developers do partake in is the implementation of embedded help bubbles that explain how some features can be used, while it is often written by technical writers the text is put in place by developers. One of the things highlighted by multiple developers is that the documentation often lags behind. One developer had previous experience writing user documentation in a previous job, where they developed a tool for other developers which they thought worked well since developers generally have a more technical background than the users of Building Advisor. While the developers often know the product well, they have almost no knowledge of the user, which is one of the reasons stated as to why they are not suitable for writing documentation. The technical ability of the application's users are varied, and writing from an outside perspective might be hard for the very involved developers.

When asked if the developers could write a 2-3 sentences long user-facing text with information about a feature, all developers stated that they had enough knowledge to write such text. While working on a feature, the features are often not that complex that a single developer could not understand the feature well enough to write the text. Often when the feature is developed to be user facing, "user stories" are created with enough detail to write this text initially. A problem brought up, relating to the previous statement by another developer, is that some aspects of work items might be interpreted differently by different developers. The only one who knows what is actually implemented might be this developer, having slightly modified the initial intent or design of the feature.

One developer stated that this information is already in commit messages, and it should be possible to recycle it, saying that if the required information is not in the commit messages, you are not writing very good commit messages. There was some contradiction about the relevancy of short informational texts, some developers believed they are not relevant at all, and that most features are intuitive, where users should often easily be able to notice the integrated feature without any information about it.

To collect information during the development process, the biggest challenge for some

developers seems to be building the relevant architecture and solving challenges through process changes. Through the case company's usage of AzureDevOps some of the relevant structure is already there, for example stories relating to user facing changes are at home in user stories. This is where relevant information about a feature could be gathered. There is some challenge in making sure that features are developed at the right level, making sure user facing info is actually in a user story or other "correct" work item. Some developers think that if this kind of information is to be created, it needs to block the process of integrating to make sure it is written before moving on to, for example, QA.

For a non-trivial feature during development, a small "team" is put together. One developer becomes an informal project owner, but it depends on the feature and team. This informal project owner could be responsible for making sure that there is documentation or information when the feature is released. Earlier, this person was also responsible for making sure that the documentation was updated. While some of the interviewees believe that information about the features could be collected inside the development of the feature, some state the necessity of all user facing text going through the technical writer. One of the main reasons given (outside their obvious experience in writing documentation) is that the technical writer has an outside perspective due to having to find out about details of the feature themselves.

Due to the increasing use of large language models (at the time of writing), some developers also see the possibility of AI written information being generated from commit messages and process information such as design documents. Some developers mention previous examples of AI interpreting and explaining code. Others state that the generation of this information or any kind of documentation in the pipeline will never be relevant as a complete solution, but adding manual steps might incur a big overhead.

Regarding what kind of information should be generated, most developers state that the information depends much on the feature specifics. Depending on the level of the feature, according to size and relevance to the user, information about deployed changes might not be relevant at all. The minimum size of information or documentation of a feature differs greatly depending on the questioned developer. Some believe that one to two sentences is enough. Others would say that a major walk-through is required, ideally a tutorial on how to use the feature.

The main things that information created about features should contain is stated to be the following:

- How to find the feature if it is not obvious.
- How to use the feature.
- What makes the feature beneficial to the user.
- What changed if the feature was updated rather than newly deployed.

One problem that is brought up and needs to be considered is that often relevant features consist of multiple smaller changes with their own stories. Sometimes these are relating to larger stories and integrated as a big change, sometimes part of changes made can be integrated step by step and therefore there might not be a central story which encompasses the whole change.

The case company has technical writers who are responsible for producing the supporting documentation. One of the technical writers was interviewed because we wanted to get an understanding of how the documentation process works now, his thoughts on some ideas to solve the problems of creating information and documentation. The technical writer is responsible for both the complete documentation that is placed on the web help page and the information that is placed in the help bubble. These two are treated differently, the help bubble information is released at the same time as the feature, but the complete documentation process is started after the feature is released. Then it's handed over to the technical writer, who's responsible for understanding the feature and then writing the complete documentation. This leads to the complete documentation being severely delayed, which is not helped by the large backlog of features waiting for documentation. Generally, the documentation is not part of the completed features "definition of done"

Some ideas to address the problems with the documentation process were discussed with the technical writer, the idea of shifting some responsibility from the technical writer to the developers. The technical writer believed that it would be possible for developers to create some information regarding features, but mentioned some problems that this might create. The problems are grounded in the fact that many developers are stuck in their developer mindset. This leads to documentation they create being highly technical and not written from a user's perspective and knowledge level. However, the technical writer thought that smaller snippets of information could be created by developers, he did not believe that such information had to go through any technical writer before release to users.

4.1.2 Literature study

As mentioned in chapter 2, the case company works with CI/CD and DevOps since they see big benefits with it. And as mentioned in Chapter 3 their way of using CI/CD has caused documentation to lag behind. This problem is not specific to the case company but has been noticed in literature and ways of dealing with the fact that documentation is often more delayed than software exists. This will be important since documentation lagging behind is a previously documented release problem in CI/CD. If it was possible to decrease the time it took for documentation, this release problem could be removed. If the documentation problem can be solved, the documentation generated can be used to inform users about newly released functionality and therefore help eliminate the problem of feature discoverability.

With the expansion of DevOps and CI/CD there has been research into how documentation can be generated at the same pace as software. Because without supporting documentation, the full value that DevOps promises to bring is compromised [10]. Documentation has three important criteria for it to be considered valuable for users. The documentation should accurately describe characteristics of the software, otherwise it will never be useful for users. It should have Integrity, the documentation can't have conflicts when describing the same functionality in different places. It must also reflect upon the most recent version of the software [10]. Timeliness the documentation for a feature must be available to users in a reasonable time after the release of software [10].

The implementation of DevOps and CI/CD does not inherently solve any of the problems of documentation. Therefore, solutions have been investigated in literature. One such solution is called DevDocOps. The goal of DevDocOps is to address problems in development where features are completed but not their supporting documentation is not [10]. One

method is to use a system called IDoc, to use the system templates have to be created that give the developers an understanding of what is required for different kinds of documentation. With the use of the IDoc tool, developers can create supporting documentation within minutes and get feedback on them through comparison with the templates [10]. This method was tested in 30 software projects and made it possible to create documentation for projects within 1–2 days instead of the 1–2 months previously. The shortened time to delivery is done by shifting the focus of providing documentation from technical writers to developers. With the help of IDoc the developer generates the bulk of the documentation and then the technical writer is responsible for reviewing and improving it [10]. One of the benefits to this approach is that documentation is created in proximity to the development.

Another approach is through CDoc, the thought is that the speed at which documentation is delivered should be the same as for software [11]. The idea is to implement a stage between "Test" and "Release" in a traditional pipeline to generate documentation for what has been released. This concept builds on "Literate programming" where information is stored together with code, an example is "JavaDoc" [11]. CDoc step is responsible for extracting the information from the code, turning it into a MarkDown document, performing checks for spelling, grammatical errors and broken lines and finally turning it into a PDF-file ready to be delivered to the users.

4.2 Analysis, Discussion and Observations

With the case company's current workflow around documentation, it does not fulfil the three important criteria of documentation. Both accuracy and timeliness are hurt by the fact that technical writers have to write all the documentation. This causes documentation to be delayed for up to months. Accuracy could in some cases be hurt by this fact, but there are cases where a feature is updated so regularly that the documentation will not reflect on the current version of the program. Timeliness is more self-explanatory, the documentation is not timely.

In the present, developers are not a major part of the documentation process. This is left to technical writers, which seem to have problems releasing documentation in reasonable time when relating to continuously deployed features. This clearly makes the current process of documentation incompatible with deployments several times a day. Releases being when you make the update available to the user, which is differentiated with deployments, when you install or integrate the feature in a system. Currently, most updates are not released this rapidly but deployed and later toggled with feature flags are ready do be released. Documentation is completed for bigger changes when biannual releases occur, however there is still a backlog of features that are missing documentation. This seems to indicate that the problem mostly relates to the smaller continuously released changes that get less priority than bigger releases, which aligns to the starting premise of the chapter. While some developers believe that comprehensive documentation is not relevant for most features that are delivered continuously, this problem will become more relevant if the team continues its plan to increase the share of features that are delivered in this way. Outside the perspective of CD, the lagging behind of documentation could also become a problem in regular development, as the knowledge of the feature could degrade and rot between the integration of a change and eventual writing of the documentation.

Before considering developers writing any form of user documentation, it could be rel-

evant to look at changing the current documentation process. Information and some small help text about a feature is sometimes integrated in a help bubble, which is presented in chapter 2. This information is written by a technical writer and is included in a feature, therefore required to be completed before the feature is released. This shows that while complete documentation may or may not be feasible in the process, some short and relevant information can be created before the deployment. This also shows that integrating a semi-external technical writer in the change process is possible without slowing the feature down too much. This could suggest that the problem is really related to either prioritization or simply the allocation of resources.

A possible alternative that we considered, is that the work of the developers could include the shorter texts comparable to that of the help bubble. When speaking to the technical writer about the help bubbles, it was understood that sometimes the text is written during the implementation of the feature and only reviewed by them. Developers believe they have the knowledge to write similar text about features they have worked in, but emphasize that the technical writer is more suitable due to writing skill, outside perspective and having relevant tool chain. If developers were involved to a higher extent, the technical writer would probably still need to proofread the text, but this could help increase the effectiveness of the process. This also deals with streamlining the information and making it more homogenous, which might become a problem if different people write these texts.

Depending on how user stories are used, they could be a good place to insert information written by developers. One of the challenges expressed about writing information by developers is locating where the information belongs. Since a feature can have multiple components which all relate to the user facing component, the user story, which "puts the user as the focus for daily work" [12] would seem a good place to gather information being presented to the user from the developers. How the work item is structured is an important part of where the information can and should be written. The structure of a task is somewhat messy today. Epics are the highest level of work item, but what kind of item is directly underneath can change depending on the feature. Looking at the agile work items used by the team right now, the user stories are sometimes used as the "main hub" of feature developments and are sometimes related to an aspect of a feature that has a visual or user facing component. To determine where the information should be written, a task level therefore needs to be decided when planning out development on a larger scale. Identifying one of the higher steps of the task hierarchy, where the scope of the work item is enough to encompass enough of a feature that information exists to write the informational text. Then a person can be assigned the task to write the information just as normal, which according to developers, most who have worked on the related item should be able to.

The idea of integrating a concept similar to CDoc was raised in a meeting with the supervisors at the company. They argued that it wouldn't be a suitable solution for this team's workflow, because they tried to keep the code clean of comments in the code. The reasoning behind this is that code should speak for itself because comments can easily get out dated and then create more confusion than benefit.

Developers brought up some things that need considering. Where should the information reside and where in the process should it be created. The structure of tasks, stories and epics are brought up in many of the interviews and relate to the agile process. The general process of using scrum and agile makes the individual developer more responsible for their part of development, and could ostensibly not have a grasp of en entire feature divided into multiple

work items. While a developer at the case company did not think this would cause problems in their work, it is something that is worth bearing in mind for future works. The company uses AzureDevOps for delegating and keeping track of work items. Here, some possible implementations for collecting information regarding features were discovered. There are free text fields that can be created for each user story, which can also be accessed using Azures API. When creating user stories, it would be possible to mark specific stories as “Important to users” and require the free text field to contain information regarding the feature.

The fact that the case company did not want to start commenting in code because of the real concerns they had with this made the CDoc approach hard to implement. The CDoc approach is probably better suited to be utilized to create technical documentation for specific parts of the program. It wouldn't be impossible to use CDoc, but other solutions probably fit this specific problem better. The Idoc solution that was found in the literature study could be used to solve the problem, since it's not built on the same ideas as CDoc. However, it might be smarter to utilize the software product that the case company already is using, Azure DevOps. Since user stories are already created there, it would be easier to link information to a change. With this approach, it's clear what user stories require documentation or information. Many of the updates to the program are backend changes that improve performance, which is not going to change the workflow for the users and therefore does not require any information or documentation. And when a feature requires the generation of supporting documents, the size and complexity varies depending on what will be released. The user story in Azure DevOps can contain an estimation of what's needed in the way of documentation and information. If an estimation of the information and documentation needed for a feature is determined during sprint planning, it's clear what is required from each feature, circumventing the problem that every feature is unique in its information requirements. Depending on how and where the information is presented, the information could contain the following to be useful for the user: how to find the feature, how to use the feature, what makes the feature beneficial to the user and what changed if the feature was updated rather than newly deployed.

The concept of automatically generated information is alluring. Utilizing large language models to aggregate information and create notification could be a future solution to generate the actual informational texts. The statement that text could be recycled from commit messages runs into two major challenges. Most commit messages are not informative enough to provide the necessary information, and any user facing feature is often an aggregation of multiple commits, many of which bears no relevance to the user. While the two challenges, the concept of automatic generation of the information is deemed outside the scope of this work, the last case still brings up the recurring problem of different parts of features being scattered. This further points towards the information needed to be generated at a higher level than individual code changes, since the necessary scope of the task to decide upon and generate informative texts often needs to be wider. Meaning that information from commits would need to be gathered as part of a bigger work item.

4.3 Results

In order for the case company to address the issues with the current documentation not being accurate and timely, more time and effort has to be devoted to the documentation.

A solution is to give the developers more responsibility over the documentation. To see a feature as done when both the code and user facing information is completed, and in the general case documentation. This would mean a change to the “Definition of Done” of a feature, and comes with both pros and cons. The pros are of course that when the feature is released, so is the supporting documentation and information. The big drawback is that features will take a longer time to reach user’s since there will be more that has to be done before something is allowed to be released. The developers could write their parts in a free text field in Azure DevOps. Then this information is delivered to the technical writer, who’s only responsible for polishing the text and making sure it’s according to company standards. This would incorporate similar ideas to the solutions found in literature but changed to fit into the case company’s already established infrastructure. Since developers seem to believe that they are knowledgeable enough about the feature, this approach should work. This approach to generating documentation can then be used to create information that can be used to notify the users of changes in the product, with the difference that this information probably could go from the developer directly to the users.

Chapter 5

Designing a solution to the users' problem of feature discoverability (RQ3)

Research question 3 of this work is combining the results from the previous goals to implement, alternatively suggest a solution to the user's problems, or to advise changes in the development process. This chapter therefore focuses on finding practical implementation details and requirements for solving the problems found in chapter 3. Together with process changes suggested in chapter 4 the proposed solutions will aim to counteract the perceived lack of feature discoverability and information about changes in the program. To find a solution, data regarding how to present information to users is collected through a literature study and interviews with developers at the company. This data will be analyzed with the primary purpose of finding requirements that are relevant to the context at the case company. These requirements will be used to create proposed designs of a system where information can be created and presented to users

5.1 Data collection

To create a design solution which can be used to increase feature discoverability for rapidly released features, some requirements need to be set in place. Requirements and ideas to solve the problem will be gathered from three primary sources. Literature will be used to find previous instances of similar problems, as well as identifying aspects of previously effective solutions. This is vital to get a base understanding of how information can be presented and what is important when doing so. Interviews will be conducted with the case companies developers and technical writers to find company specific requirements as well as evaluating the possibility of certain implementations. Developers will also be interviewed as general users and stand-ins for users of the program, since those could not be interviewed. To supplement interviews, user navigation data will be analyzed to understand how users interact with the application. The data collected will help build a design specification for a possible

implementation. Possible requirements for such a design proposition will be written in bold text.

5.1.1 Literature

While there is no example of the exact problem that appears at the case company, there are similar problems found in earlier studies. These will be explored and analyzed to find the required component to create a design specification. There are different parts that need to be explored in the implementation process. The initial part that needs to be investigated relates to how updates and new features can be notified. This section will also investigate how information created during the implementation of a feature can be shown to the user effectively. The literature should be a base to interview developers experienced with the program and afterward evaluate their input.

Web based embedded assistance tries to solve the problem of users having insufficient knowledge to utilize some features of a program. According to, DeLoach [13] information has to be available at the point of contact. This lowers the bar for the user to search out information regarding the product. Users are reluctant to exit the application to search out the required information, and therefore **information or documentation should be embedded in the application**. After implementing embedded assistance users are more probable to access the off site documentation and spend time researching the program [13]. This shows the importance of implementing more web based embedded assistance.

Embedded user assistance (EUA) is probably the field most closely associated with the embedded feature notification discussed in this work. Studying it from a human computer interface perspective, methods of EUA are ever evolving, and contemporary solutions are different from those found in earlier programs. Tulaskar [14] presents two lists of examples of early and in their opinion outdated examples of EUA as well as a list of contemporary ones.

The Various contemporary solutions that are brought up by Tulaskar [14], are often inspired by earlier designs. Contemporary solutions generally seem to have some themes in common, and Tulaskar make the following observations. Contemporary solutions tend to contain more **visual cues** and colors to make them stand out from the parent interface. They tend to have **more compound information** as products tend to become more complex. The information, though more numerous, tends to be simplified through contextualizing or breaking the information into smaller parts. Some contemporary EUA solutions can tend to be ignored due to extreme use, still they fill an important part in feature introduction for intermediate users, because even when they don't provide extensive documentation they highlight changes to the program. Important parts of contemporary EUA is **efficiency and interactivity**, they should allow users to focus on particular tasks or workflows while **allowing the user to interact with the product**.

Eriksson [7] also emphasizes the importance of embedded elements but in the context of feature discoverability. The program study is based on a professional picture archiving and communication system (PACS) software tools similar to the one in this work. The article describes a similar case to the case company where they have identified that users don't necessarily pick up new features, with a general release cycle of big releases biannually. Eriksson explores several concepts of how to improve feature use, one of them is "Highlighting new/updated/undiscovered features". The concept explores how to effectively highlight features in a traditionally released program, a picture archiving and communication system,

Positive	**Negative**
Non-intrusive	May only work for features that have a graphical UI component
If more information can be accessed from the "new" feature ,it is an easy way toobtain additional information	In a complex interface with multiple subpages, it is difficult to find where all the "new" indicators are
Smooth and reasonably disruptive way of highlighting things	It can result in a large amount of "new" functionality appearing all at once
Clear visual indication of what is new and/or unused	If the "new" indicator lasts longer than the initial use, it can be disruptive, especially if users need to click on each one to remove them
On demand	Can become unstructured and difficult to get an overview of multiple features that should be used together
Good utilization of technology versus human interaction (letting the "system" filter what is relevant)	

Table 5.1: Summary of relevant bullet points from the PMI evaluation on visual highlighting performed by Eriksson [7]

which is a professionally used tool used by radiologists. It relates to alerting users of new functionality and old functionality which has not been utilized using **visual elements** in the GUI, taking into account previous research about EUA. Of the different concepts explored in the article, it finds that this concept is preferable to implement due to it being viewed as cost-effective, easy to implement, suitable level of intrusiveness, minimal effort of access to the user and least likely to cause new problems. Eriksson gathers impressions and thoughts about the concept of visual notification, a summary of which is gathered in table 5.1.

The table contains relevant bullet points from the study by Eriksson. It highlights the positive aspect, such as often having a **low intrusiveness if pairing small visual cues with information being available in other locations**. A good implementation is a one that lets the user **find features on demand** and does not necessarily force the user to interact with new elements, possibly alienating their image of new features or the new feature specifically. The system should be able to **filter what information is relevant to the user**. The paper also expresses negative thoughts about the concept, also shown in table 5.1. One of the thoughts that seems to relate to this work's context and is brought up in interviews is that the concert only works for features that have a GUI component. Smaller visual elements also often don't provide enough content about a feature themselves, and should therefore be used as a complement to documentation or other information about a feature. In systems with complex GUI, such as multipage interfaces, it can be difficult to find where visual indicators should be

placed. It is important that the visual indicators are well-designed, as they can be perceived as annoying by some users. One therefore needs to be mindful of which features are notified and how long notifications should remain after a change is integrated.

The paper makes it clear that what they refer to as the concept of "Highlighting New, Updated and/or Undiscovered Functionality" is a subtle and established way to increase feature discoverability. In the study, they implement a "dot" which can be added to a tool in the programs' menu, indicating that something has changed. An example of this can be seen in figure 5.1. This shows that something has been changed, but can without a tooltip not inherently show information about the feature. This type of smaller dots or "hotspots" were not seen as intrusive and were appreciated by users. To address users who still think the intrusiveness is too high, the ability to **toggle the highlights** is seen as a way of addressing these concerns.

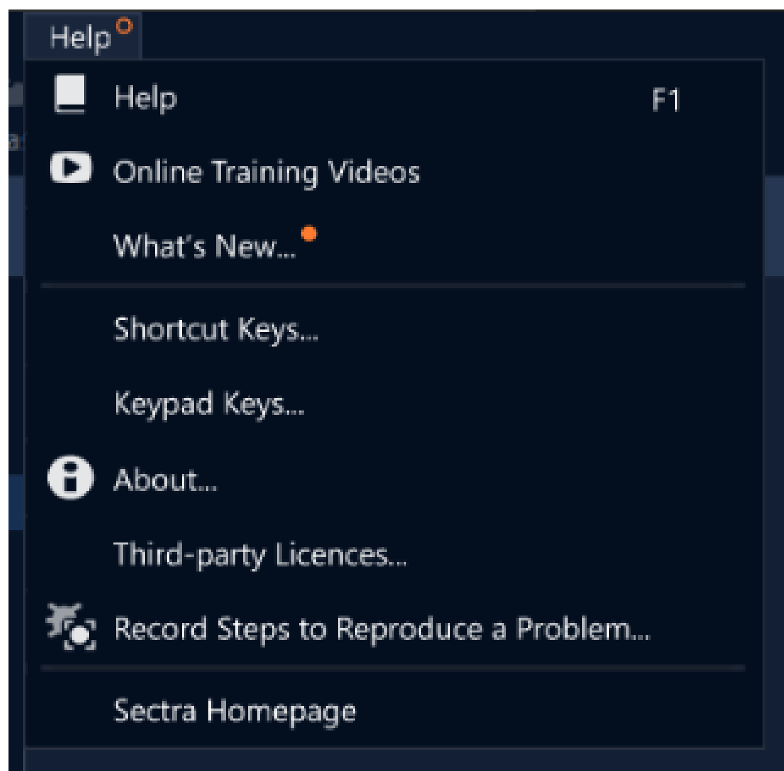


Figure 5.1: An example of visual notification in a menu-driven program shown in Eriksson [7].

It seems users of the PACS program appreciate a **combination of text and visuals**, but with consideration for context and non-intrusiveness. The paper re-iterate that users often appreciate not having to leave the system to find information they are looking for. Both in the PACS study and from this work later interview, users seem to appreciate having access to something similar to release notes and some kind of integrated release notes.

If informational, texts are to be used, it should be gathered together in an information anal flow such as a "What's new" window. Eriksson suggests this should be automatically opened after significant updates to allow the user to find new elements. Which necessitates some thought when applied to a continuously deployed and feature toggled program.

Other concepts found in literature often concern expanding education about the program. Such examples are different kinds of educational walkthroughs or integrated documentation. Education about the building advisor program exists about bigger features when they have bigger launches. To solve, specifically, continuously developed features having low feature discoverability, educational walkthrough or tutorials would most likely slow down the development and is not ideal for solving the problem found in this report. While integrated documentation might harness the benefit of EUA over regular documentation, it would not solve the problems this paper has found with user documentation in CD. This means that it might be a good idea to use integrated documentation, but it has little bearing on feature discoverability before documentation has been created, which has been stated to be the case in the current product.

5.1.2 AppInsight

The literature study can provide us with some general guidelines for how information should be presented to the users. However, it's important to know how the case company's users interact with the software. With information regarding the specific users of the program together with theoretical knowledge, a solution can be designed that fits better than a stock solution. This subchapter is dedicated to using App Insights data, which collects information about how users click around the program. More specifically, how they interact with the closest feature currently implemented that resembles a notification system. This is the help bubbles mentioned in chapter 2 in comparison with the off site documentation "WebHelp".

The number of accesses to the different kinds of information and documentation is shown in the graph 5.2.

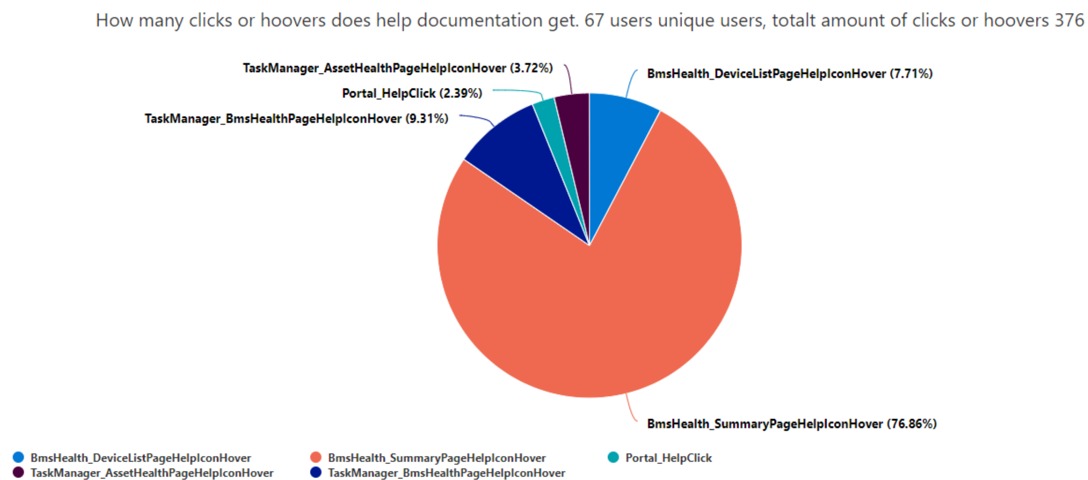


Figure 5.2: A pie chart showing how users interact with help information and documentation

The graph shows the number of hoovers or clicks to the help documentation. Showing that during the last 30 days (2023-10-21 - 2023-11-20) only 2.39% of the times that users accessed help was through the official documentation portal. The graph is divided into pieces since there are different help bubbles for different web pages. The Web Help portal only has one central hyperlink that is used to access it.

In the graph presented in figure 5.2 we can see that most of the users access the help bubble specifically at "BmsHealth_SummeryPageHelpIconHover" much more than any other help bubble. Which leads us to our next query. This graph shows how many times "Portal_BmsHealthTabSelected was accessed, its Help Bubble and the complete Web Help documentation which results in the figure 5.3.

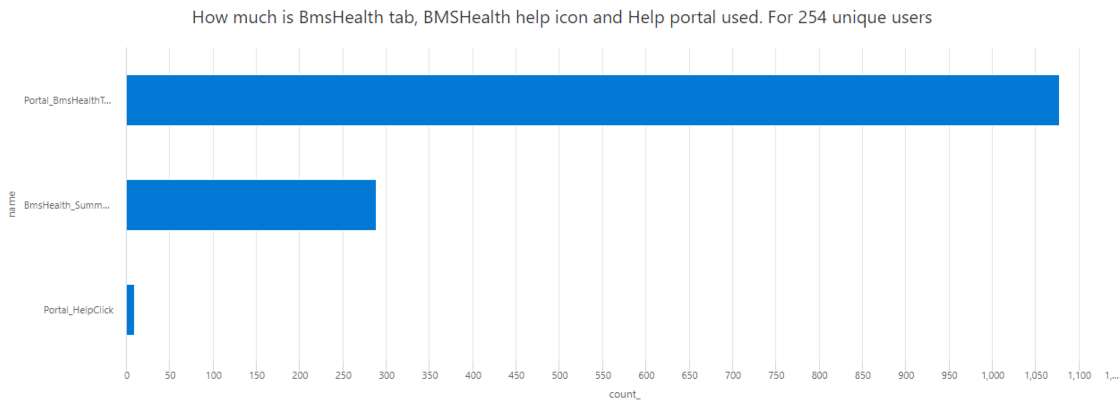


Figure 5.3: A in depth look at the tab BMSHealthTabs help documentation

The first bar shows the number of clicks into BmsHealthTab, this is used as an indicator of how much that specific page is used. The second bar shows how many times users hovered over the "Question mark" icon for help regarding the page, and the third bar shows how many times people accessed the off site help documentation. The quarry looks at the days (2023-10-21 - 2023-11-20). This shows that around 26% of the times people access this specific page they will also check out the embedded user information that is on that page. And very rarely people will access the complete Web Help documentation.

From figure 5.2 and figure 5.3 we see that users are much more likely to access the embedded help information. This is also what is expected after reviewing the relevant literature. DeLoach [13] makes it very clear that users are reluctant to break the flow of work and exit the application to go to another website to get help. **This indicates that the notification system should also be embedded into the website so that users actually find the information.**

5.1.3 Interview

Interviews with developers were conducted with two main purposes. Getting input about what kind of implementation could work for the building advisor ecosystem, and because everyone is interacting with software that is being updated constantly. Getting input on what can and can't work may impose some restrictions on the design proposition, but the developers have much more knowledge about the platform, making this input important. The second purpose is to get an understanding of how users like to be informed about changes in software. This input is important because the design proposition should be designed in a way that users appreciate. The questions asked can be found in Appendix A.

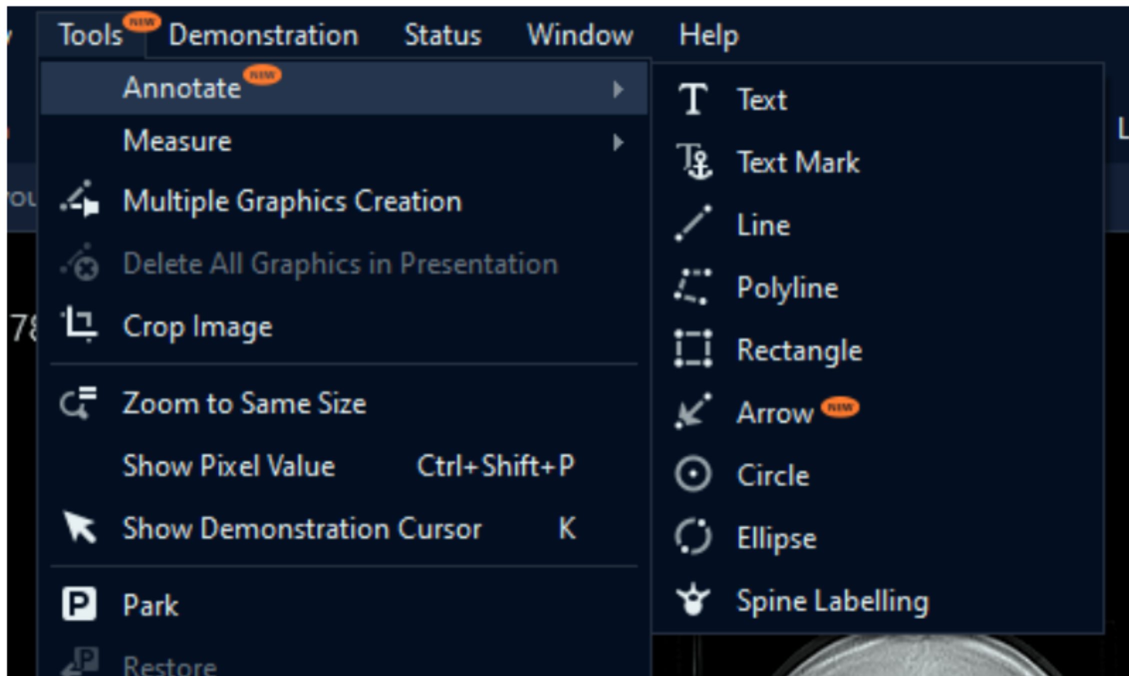


Figure 5.4: Variant of visual embedded informative dots presented in Eriksson [7].

Literature suggests that embedded information is a good strategy, therefore the interviewees were show the example of visual notification shown in figure 5.4. Showing a small bubble with the text "New" in it. Generally, they liked the idea of a **small visual notification as a way to flag for new features**. Most of the developers believe that it would probably be a pretty simple thing to implement. Some developers however bring up the difference between the program in the screenshot and Building Advisor. Building advisor is not a menu-driven program like the PACS from the screenshot, and even the top level interface changes with every page. Some developers raise the problem that this implementation can create a visual overload for the end user, forcing them to click through parts of the program that they don't want to view to get rid of the notifications. Although, some raise the positive aspect that this is less intrusive than other implementations that force users to go through a guide before allowing free access to the program. One developer suggested that this could be expanded to include a tooltip when hovering over the bubble, sort of like the already implemented help bubble previously mentioned.

Since everyone living in the modern world has experience with programs that are regularly updated, the interviewees were asked if there were any programs that informed their users of changes that they liked and disliked. An interesting conclusion from this question was that there is no solution that fits everyone we interviewed. An example of this is Microsoft Teams, some interviewees said that Teams were very intrusive disrupting workflow forcing users to click away popups.

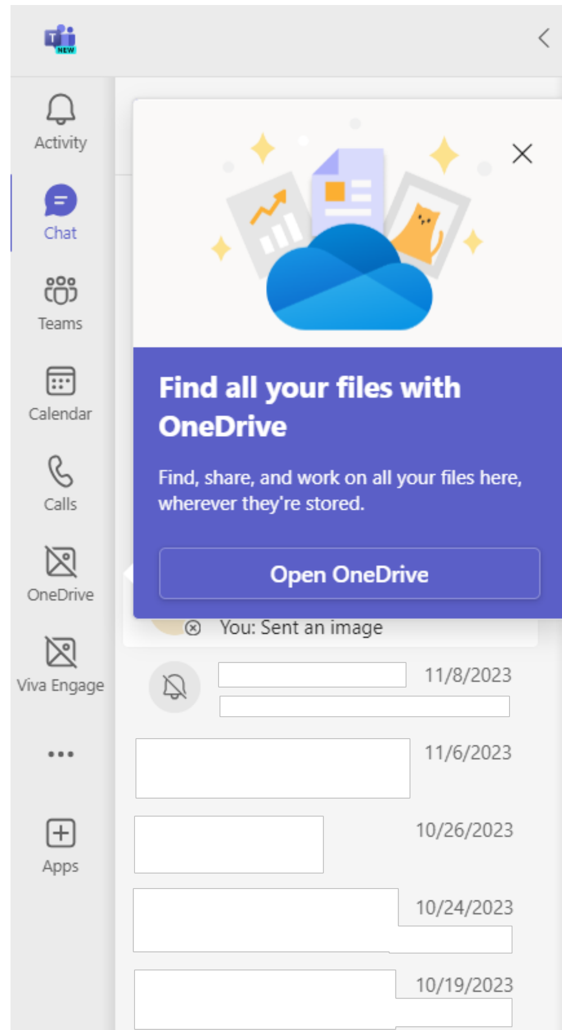


Figure 5.5: Microsoft Teams popup notifications, here forcing the user to click it away before accessing the most recent conversations in Teams.

With a small sample size of interviews, there is no clear best way to implement an information system for new features, because the interviewees preferred different things depending on the program and how they use it. Although, the interview highlights a few important things that most people appreciate. **Choosing the time when they are getting the information and not being forced into it upon starting the program**. Comparing this to the 5.5 where you have to remove the notification before contacting the people that you are most likely to contact will probably lead to users not reading the information and just clicking it away. The interviewees also mentioned that with the Teams'[1] notification system, it's hard to find the information after it's removed because it won't appear again.**Therefore, there should be a clear place where to find the information if a user is interested**.

One program that was generally considered better than Teams was Discord[2], their implementation builds on presenting patch notes when the program is started. These patch notes are also prioritized so that features with high impact are higher up, making it easier

[1]https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
[2]https://discord.com/

to read and understand what's important. Therefore, **prioritizing what is shown seems to be important**. In most cases, the company prioritizes what is important statically. On thought that might be useful is making this prioritization custom to the user, either by highlighting their most used features or letting users decide what to "follow" themselves. Users can't be spammed with information that they don't want because it will only lead to users ignoring information.

One thing that is consistent is that the more time the user is investing in the software, the more likely they are to want to know about changes. Some examples of this are reading the patch notes for the integrated development environment that they are using, since much of the time in their work is done in that program.

Another interesting observation brought up from one developer was GitHub's [3] way of informing users of new features. The thing that was appreciated with git was that they only sent out emails when there were important changes, which leads to a trust relationship, making the users certain that reading the patch notes will not be a waste of time. This strategy is more in line with batching content and making it available to users in big releases. **Batching content until there are important features released to users**.

5.2 Analysis and Discussion

In the data collection part of this chapter, multiple aspects for notifying the user of released changes were found. The aspects need to be considered when designing a solution. We will discuss and analyze ideas and elements that make an effective system for informing and notifying users, which isn't annoying for users to interact with. This analysis will form practical requirements needed to later implement such a system.

The sources brought up in the data collection all agree that an embedded solution is a good way of increasing the user's knowledge of the program they are using. In the context of a professional tool where the user is focused on performing a task, embedded might be the only chance to inform the user. It seems the willingness to read traditional patch notes most often relates to the interest in the application or program. Examples of good ways of presenting update information are most often in applications that are privately used or relating to fields of entertainment or fields where self-improvement is key. This is indicated by examples of people mentioning games, private communication and community as well as IDE:s, like VS Code or eclipse, which are often core to the developer even when not working for their employer. This fact along with the benefits discussed above means that an embedded solution like EUA is an important requirement. Especially since it seems that most users do not interact with other sources of information according to the navigational data. Other ways of presenting updates could be the user forum for the product, where users already can post proposals to the developer. This would be cost-effective in a way since existing assets can be reused, but since once again since most users does not inter act or perhaps even want to interact with external sites, EUA seems like a preferred solution to present the information. Further requirements will assume that a solution is embedded in the program.

Another important quality is non-intrusiveness. Some contemporary EUA have problems with presenting too much information, which can overwhelm the user, only alienating the user to change and further attempts to inform said user. This means that non-intrusiveness is

[3] https://github.com/

an important requirement that can be improved by taking heed to a couple of observations. Mainly, EUA should not prevent the user from interacting with the program, this makes solutions like forced walkthroughs and tutorials a non solution for informing the user of new features. This requirement comes in conjunction with the users' choice to not interact with highlighting elements. If the user does not want EUA or other forms of notification, they should be able to turn it off.

The distinction between information and documentation is mentioned many times in chapter 4. This distinction is also important when presenting the information to the user. While integrated documentation can contain the entire documentation, EUA benefits from compounded information which is just-enough to enable the user to interact with the program. As mentioned in finding answers to RQ.2, proper documentation is often not finished when the feature is integrated. The changes to the process presented in chapter 4 facilitate just this kind of compounded information. One requirement for a solution would be that any textual information contains just-enough compounded information. This for the purpose of informing the user without overwhelming them. This can then relatively easily be implemented, since this kind of information can be created during the development of rapidly integrated features. This means that this requirement helps the specific problem that appears with the continuous release of features. The conclusive requirement is that textual notifications **should contain compounded information rather than longer texts**.

A small visual notification as a way to flag for new features might work for certain features. Developers liked this kind of implementation, but as they said it can work for certain features but not all since the program isn't as menu based as the program from which the screenshot is taken from. Therefore, this is more of a good-to-have requirement that can be used for features where it fits. In the case where changes have no obvious visual elements and still need to be presented, it can still be presented in text instead. Developers seem to like the idea of a "changelog" like list where changes can be represented for a user who wants to get a general overview of the program. This also aligns with the idea that visual elements and cues work best with complimentary text existing in other locations close at hand, making it easy for the user to explore further on their own prerogative. This means a similar requirement is to **pair visual elements and cues with text being implemented in other locations close at hand.**

The interviewees brought up that being forced through a walkthrough of a program upon start or having to click away messages to interact with the program was heavily disliked. Therefore, it's a must-have requirement that the implementation doesn't force users into reading information and clicking away information, but instead lets the users do it when they want. Leading the requirement: **choosing the time when they are getting the information and not being forced into it upon starting the program.**

Going along with the previous requirement, the interviewees mentioned that in some applications it's hard to know where to find information about new features if a notification clicked away. Which leads to the requirement: **A clear place where to find the information if a user is interested**. Since the lack of a clear place to find information seems to annoy users. It's also simple to implement, making this a must-have requirement in the final design proposition.

Prioritizing what is shown. As previously mentioned not all changes need to be notified, such as backend changes that only affect performance. Therefore this is a requirement in the sprint planning process, but it's still important and is a must-have requirement for the

implementation. The process starts during sprint planning, at this stage the project owner or the team as a whole should decide which features are important to notify users about. Only changes visible to users should get this extra step so that users don't feel like they are getting spammed with new updates that only affect performance and such.

The requirement **batching content until there are important features released to users**, is a possible solution to the problem, however this requirement doesn't fit within the CI/CD setup that the case company currently is utilizing. Therefore, this requirement will not be included in the design specification.

There needs to be some system that can filter what notification exists for any given user. This can in versioned programs boil down to what features were added in a recent patch or update but become more complex in an application like building advisor. Two main problems arise. The continuously deployed nature of the program makes showing what was new for the user harder, since you need to keep track of the individual user's interaction with the program. You need to see when the user last interacted with the program as a whole if you want to implement a more changelog-like program, or you need to track the user's interaction with each element if you have a feature by feature notification scheme. The problem becomes even more complex if you take into account feature flagging for different users. This could either be documented in the databases for the product or kept tracked by cookies on the users side. However you want to implement the feature promotion, the **system needs to filter what information is relevant to the user and can't contain static changelogs like versioned programs**. Similarly, users should be able to clear any visual elements without having to interact with them directly.

5.3 Design specification

By applying the requirements that are useful and viable for this specific setup, a design proposition can be derived. The requirements were discovered during data collection and pruned during analysis and discussion. This culminated in this light proposal on how to notify users, solving one of the core issues identified in this work.

Summarizing the specification found by analyzing the data, the following requirements were found:

- Any implementation of feature notification should be embedded into the application.
- The EUA inspired notifications should allow the user to still interact with the application.
- Highlighting or notifying elements should be able to be turned off.
- Any textual information should be brief and compound rather than longer texts.
- For changes with a clear GUI element, there should be visual notifiers.
- Visual notifiers should be paired with textual information found close at hand.
- Information should be presented in a way where users can choose the time when they are getting the information and not being forced into it upon starting the program.

- There needs to be a clear place where to find the information if a user is interested.
- Prioritizing what is shown is necessary as part of the process, not all updates need to be notified to the user.
- The System needs to filter what information is relevant to the user and can't contain static changelogs like versioned programs.

As a visual element, implementing visually informative dots, similar to those in the paper by Eriksson [7], was considered a possibility and general feedback was positive. Building advisor is divided and nestled, in the sense that the program contains several dashboards rather than an always existing menu. For an implementation, this means that visual indicators can have a hard time highlighting features in a sub-page that the user is not currently on without becoming jumbled. This can be considered a good thing to decrease the intrusiveness of visual cues if the visual cue is complemented with a list of changed features that is available on every page of the application. If new pages are added, they should be indicated with visual notifications at their entry point. New features or changes should preferably not be visually notified outside the page they exist on, as not to clutter the interface. How the actual visual cue is implemented would in later stages depend on a specific set of classification for features. As an initial proposal, changes on a page could be aggregated into the currently existing help bubble. This is a good place to start since the help bubble appears on most pages and since the team increasingly updates them. When there exists a change, the help bubble should have an irregular appearance and when hovering over it some brief information about the change would appear, which can be dismissed after reading it. Preventing the user from unknowingly dismissing the notification would need to be considered. This information would be created as a step to develop the feature, and which information would need to be prioritized during development.

Textual information needs to be available both to satisfy the general requirement, but also to complement visual cues and any information presented in places like the proposed help bubble changes. This could consist of a list that can be expanded from all pages where information about changes implemented is available, for instance sorted in implementation order. The information would be created during development would be presented here in addition to textual information on individual pages, like the suggested changes to the help bubble.

The system would require two important layers not visible to the user. Primarily, one would need to deal with what information has not yet been interacted with by the user, as well as which changes have been made since the user last used the application as a whole. Secondly, this system would also need to consider that not all features are available to the users, since features are sometimes toggled for some. This filtering of information should be possible to implement.

Chapter 6
Discussion and Related Work

This chapter will focus on the research method of the thesis. How the work changed from what we intended when writing the goal document before the thesis initiated. How the changes influenced the scope of the thesis and what results were possible to achieve. This will be followed by a discussion of what threats there are to the validity of the results. After that, the generalizability of the results will be presented and discussed. Four papers that relate to the subjects that this thesis covers will also be presented to give a broader context to the results. Finally, there will be suggestions for how this subject can be expanded upon will be presented as future work.

6.1 Reflection of our work

During the work of answering the three research questions in this paper, the initial scope of the questions were somewhat changed. This subchapter will discuss how the initial methods and targets were modified during the course of the thesis. Comparing our thoughts about the work before and after execution, helps reflect upon what could have been done differently and what methods worked well. Together with a broader evaluation of the research methods used in the work, this will strengthen the validity of claims made during the case study at the case company. This will also be an opportunity to discuss which lesson were learned during the project.

The scope for studying all the research questions was generally narrowed quickly, which sometimes left unexplored options. The constriction needed to happen as a result of the time constraint on the thesis but also happened naturally. This was a result of the method of exploration. Often, the exploration led to a certain area or problem that was deemed relevant to explore over other alternatives. While it was necessary to narrow the scope, this leaves the result vulnerable to perhaps obvious pitfalls which could have been found if a more thorough investigation into other options had been conducted. This is hopefully remedied by having done a thorough initial investigation and doing shallow investigations of most of the options

before focusing on what was deemed most important. If conducting a similar study in the future, it would have been useful to narrow the scope earlier to avoid wasting time doing research which did not necessarily contribute to the result.

The initial RQ specifically targeted the behavior of the users of the building advisor program. This meant that the inability to question its user directly changed the initial investigatory method to depend less on interview material and more on secondary data like second hand sources (developers of the program), user data and literature. This means that the initial question, "Why users are not using newly integrated features?", was in a more confirmatory fashion than initially intended. Several possible root causes were established and the validity of these causes investigated. With access to users, the problem could have been more directly investigated, which would lend a stronger case to any solution solving the behavior of real users. Since the changes made the available data more important, there were some things that in hindsight should have been more thoroughly investigated. Having concluded that one of the causes of the adoption problem was release strategy, a more in-depth investigation could have been done to increase the understanding of how the product was released from a user perspective. Similarly, the nature of CI/CD adoption on a company-wide level could also have been further explored. Since these two elements could be said to be more adjacently related to the developer teams work, this could probably be more thoroughly investigated by a management focused study.

The user data used in this study is gathered from a relatively short timespan. The timespan for which there exists data is also outside the most important period for our study, that being the release of a major feature. If the timing of the study had been better, user data could have been more of use and the data would most likely have a higher impact on the conclusion of the study. Initially, there were plans to also generate data on feature use before and after notification with a simple prototype, which would be more directly related to the questions and more conclusive when used for analysis. This idea was also scrapped as other parts of the work took increasingly more of the available time. If doing the study again, planning and making sure the data was available during the study would be the first thing changed.

When constructing research question two, the assumption was that there would need to be some kind of informational text written as part of solving the problem. This turned out to be true. While there was a thorough literature study and interviews, it would have been useful to do a more practical observation of the workflow of the team at the case company to gain a more customized solution to the team, rather than the general answer for anyone using azure dev ops. This is also generally true for most of the conclusions drawn in this thesis. If the thesis study had continued, practical solutions which could be reviewed and tested would have generated feedback and data to strengthen the conclusions. Since there was not enough time to implement a solution, this could be further expanded in a later work, taking into account the design specification presented here.

The thesis generally conforms to the case study method as presented by Runeson and Höst [1]. while the study does not adhere strictly to the format presented in the article, the general research process of the case study was followed. The general outline for generating a hypothesis and then confirming as outlined in the paper were useful and made a structured approach to finding and proposing solutions to the problems at the case company. The methods for analyzing qualitative data were especially insightful.

Generally, most of the lessons learned during the project have to do with planning. The project was timeboxed, but things taking longer than was allotted should have been handled

better as to not disturbed later stages of the work. While there are much that could have been done, differently, according to the things brought up in this subchapter, We are generally satisfied with the work as a whole.

6.2 Threats to Validity

There are many factors that can influence the result of this thesis. These threats to validity will be presented in this subchapter, discussing how severe they are, in order for readers to gauge the reliability. Together with the threats themselves, countermeasures will be brought up when relevant.

Since the company couldn't provide end users for us to interview or major definitive data, most information regarding what users want is at best second source information. This means much information about the user came from the offer manager and the project owner from their meeting with users. This means that the assumptions that lead forward to conclusions in the different chapters might be colored by biases from the people we interviewed and had meetings with.

One source of data that was primary information from users was the AppInsights data. However, the company's current setup only allowed for the last 90 days of this information to be saved. This meant that we couldn't examine the behavior of users over a longer period of time. And since data was only saved for 90 days, we also couldn't examine users behavior to new changes after the big biannual releases. This could probably be mitigated by saving the retiring data ourselves, but when the 90-day period was discovered it was too late. There was also the issue that the number of customEvents that were used to track how users interacted with the software was lower than we first expected. Many small features released lacked customEvents. This made comparisons between the continuously released features and the biannual released features hard, since there was less data for continuously released features.

Because of the time limitation of this thesis, and earlier parts of the work taking longer than anticipated, it was decided to only present a design proposal rather than implement a solution. Therefore, everything is based on very theoretical information. There is a possibility that there are going to be issues if the solutions that are presented in previous chapters are implemented due to things missed. There could be implementation problems in the software that make certain propositions more time-consuming than what it may look like at first glance. There can also be issues with how the affected people react to the changes, an example of this could be the move towards developers writing some supporting documentation. In the interviews, developers were generally positive about this, however that might change when they actually have done it. Therefore, a prototype setup should be investigated before committing fully to the results in this thesis.

In qualitative research there can easily be bias when conclusions are drawn with a low number of researchers, this can be mitigated by performing common analysis and strengthening observations with other academic literature. With a low number of authors of this thesis, this bias should be taken into mind. In terms of quantitative data, this study is quite bare. Navigational and user data exists, but with its low n-value as well as its short timespan, its use is limited. While it's used in this work to strengthen hypotheses created during the work, the quantitative data alone is probably not enough to draw any major conclusions.

The threats to internal validity should also be considered. Due to the scope of this work

being narrowed quite early, there are generally factors which are not considered which leads to the users not using the features which case company's team develops. The initial premise frames the question out of a CI/CD adoption perspective, which leads to some otherwise relevant factors being left for future works to explore.

Since many of the individual data points are not enough, we have strengthened our vulnerabilities by increasing many sources of data, which all point to similar conclusions. While the necessity of discussing the threads to validity is key, the considerations made for them means that the problem has been investigated from different angles and threats to validity are reasonably minor. The strongest claims of the paper are those confirming problems which earlier papers have not found suitable evidence.

6.3 Generalization of results

Most of the result of the study seems to be relevant to companies or teams with a similar product to the case company, some of the more general info can be used for versionless products or to software processes as a whole. Since CI/CD is expanding in the software industry, there are many parts that are generalizable. Since much of the information came from literature, there are aspects in every chapter that can be used in a general context. An example is that the usage of embedded user assistance is more used than off site documentation.

For research question 3 we weren't able to contact the specific users of the company and had to resort to interviewing the developers. This leads to the answers being far more in line with those of general users than if we had interviewed the actual users of the program. This increases the generalizability of the results in the chapter, but lessened the use for the case company.

The problem with the feature discoverability is found in at least two other software cooperation brought up in the course of this work, this indicates that the problem is regularly occurring. One of the companies, Atlassian, seems to have a very similar situation and by performing this study and finding similar problems, the generalizability of both studies seem to increase.

6.3.1 Related work 1: On the journey to continuous deployment: Technical and social challenges along the way

On the journey to continuous deployment: Technical and social challenges along the way by Claps et al. [9] identifies that the Agile development methodology is increasingly used among software companies. Even further, the need for rapid development cycles has made certain major companies adopt continuous deployment to gain its benefits, one of which the paper states to be "lowering the risk in any release". As continuous deployment becomes more popular, it also becomes clear that there are technical and cultural challenges that need to be identified. The paper seeks to identify these challenges as well as find methods to mitigate them, helping to increase the likelihood of a successful CD adoption.

To identify the challenges when adopting CD, the paper uses two methods, an initial literature study as well as a qualitative exploratory case study at the software company At-

lassian Software Systems. The company's teams have adopted a number of different software development strategies, one of which is CD. The authors interview employees at Atlassian using questions modified from Vavpotic and Bajec [15].

The paper finds a total of 20 CD adoption challenges, of which 9 are considered technical and 11 considered social challenges. The paper lists the found challenges along with possible mitigation strategies for each one. The challenges are further classified into 5 categories of discussion relating to the challenges and their mitigation strategies found by studying Atlassian.

The need to be lean: Companies need to become lean by working with smaller batches and breaking tasks into small tasks and software parts which can be delivered to customers faster. By using feature toggles and canary releases, the risk of the fast release cycles can be mitigated. Management-driven adoption: CD adoption must be a company-wide effort and "requires involvement of different organizational units in order to fully succeed".

Changing responsibilities: In CD software developers also become the deployers of code, with less oversight from quality assurance, resulting in more pressure on the developers. This can be mitigated by increasing communication and introducing management programs for adopting CD. The risks of adopting CD: Frequently it seems that CD is not suitable for some types of software. One example is enterprise software, since it is easier to introduce bugs into continuously deployed software, bigger enterprises are sometimes hesitant to purchase such software. It is also pointed out that customer feature discovery often suffers along with the ability to integrate with external API's and products since there are frequent changes.

The paper concludes that a company adopting CD needs to be prepared to face the challenges that come with CD and be prepared that the mitigation strategies presented in their work may produce additional challenges.

The transparency level of the interview process means that not all challenges are directly motivated. This makes it hard to predict if the challenge and mitigation strategy is relevant to the cases at the case company, and decreases the usefulness to this thesis. This could perhaps be remedied by including important parts of the interviews or stating the found symptoms of each challenge along with the description and mitigation strategy. If the interview questions were more clearly stated, this could have helped this study to identify if the same problems could be found at the case company.

According to the paper, Atlassian's process "lacks the true attributes of CD". This seems to hint at challenges which have yet to be solved, which could be very relevant to this thesis. Investigating this could have been especially helpful since the case company also has a similar type of semicontinuous deployment where some features are released as before.

Feature discoverability is partially investigated in the paper. One interviewee says, *"I think, our biggest challenge right now for customers"*. This is one of the initiating sentences that inspired the investigation done in this thesis. It seems that Atlassian is in a similar situation as the case company and by comparing results can make the result from both this thesis and the study performed at Atlassian more generalizable.

6.3.2 Related work 2 : DevDocOps: Enabling continuous documentation in alignment with DevOps

In recent years, DevOps has been deployed to many projects and organizations to increase the speed of development of software. Many companies have seen great value in adopting this software development methodology, however since DevOps is mostly focused on code artifacts supporting documentation is often looked over which leads to outdated documentation. To try to address this problem, the paper [16] investigates if parts of the documentation process can be changed. This is important since supporting documentation is often needed for the users of the program to be able to use the program.

The two traditional industry ways of creating documentation are either through technical writers or open source documentation solutions. Technical writers often employ DITA (Darwin Information Typing Architecture). DITA offers a lot of value, one example is that different documents can inherit properties from each other, such as user manuals. These advantages come with the disadvantage that these systems are complex and require the specific knowledge of technical writers to operate. Instead, in the open source community documentation is treated in the same way as source code, projects usually don't have technical writers, so developers have to shoulder these responsibilities themselves. Because of this structure within open source projects, documentation can easily become redundant and faces the risk that documentation is not consistent to the software.

DevDocOps tries to solve the problem of supporting documentation lacking behind by utilizing the strengths of the two traditional ways of documentation. The core problem is that documentation can't be developed at the same rate as software. The idea to solve this is through shifting the responsibility to the development team of writing documentation for the features that they are working on. To help with this, a pipeline can be built to automate some parts of the work. The big benefits of this approach are that when a product is ready to release to customers, there is also supporting documentation ready to be deployed. Technical writers that were previously in charge of writing documentation now get a role of improving and curating the support documentation that developers generate.

The report found that both the quality and speed of supporting documentation increased because the responsible developer knows what's been implemented and what needs to be documented. Also, a fast feedback process between developers and technical writers allowed problems with the documentation to be fixed quickly.

This paper proposes a solution to decrease the time between finishing the development of software and its supporting documentation. The fact it was a solution implemented in a real project and managed to reduce the time for documentation to be completed shows that it is possible to solve the documentation problem in CI/CD with technical solutions. Even if we did not build the solution presented in this thesis on the same platform as the paper, the ideas are the same, such as technical writers taking a more supervisory role over documentation.

6.3.3 Related work 3 : Identifying Characteristics of the Agile Development Process That Impact User Satisfaction

User satisfaction is an important aspect when developing software, empirical studies have shown that user satisfaction is increased by using agile development.There has however been little research into statistically proving what parts of agile development affect user satisfaction, therefore the paper [17] investigates this. By analyzing reviews from Google Play Store, corresponding development metrics that are important for agile development.

Previous research regarding user satisfaction suggests that release notes correlate with a higher rating of mobile apps. Other research suggests that there is no correlation between code quality and how highly an app was rated. And finally through interviews with users one study conducted that reliability is a significant factor, once the app is reliable the things that are correlated to a higher score are capability, usability and performance. However, these studies did not examine how the development process affected users' experience of the application.

To be able to rate user satisfaction, reviews are taken from Google Play Store and the text is analyzed by a sentiment analysis that outputs a numerical value in the range minus one to one. Where minus one is classified as a negative and one is a positive review. The principles of agile that were decided on were chosen because of the impact they could have on users satisfaction and aspects being easy to collect from public git repositories. There are some criteria that were used to filter which software project's/Apps to analyze. Such as being under development for more than 2.5 years and at least an average of 40 reviews per year. Criteria were put in place because to reduce short term noise in the time series analysis and getting an ample amount of data.

After applying all the different criteria, there were 35 apps that were left and could be analyzed. This showed that there were two criteria that affected user satisfaction, merge duration and remaining pull requests lifetime. Both these had negative correlation, meaning a shorter merge duration usually led to higher user satisfaction. However, shorter merge times can often be attributed to many factors, such as the software team's motivation and how complex the code base is.

Some factors in agile such as lead time and number of merged pull requests had no clear correlation with user satisfaction. This indicates that release frequency does not affect user satisfaction, neither positive nor negative. A trend between remaining issues' lifetime was also correlated with negative user satisfaction.

The result of this study seems to suggest that it is not negative for user satisfaction to batch content, since there is no correlation between release frequency and user satisfaction. Therefore, it would be interesting to investigate if a simpler solution to solve the problems with feature discoverability would be to batch more content and release it when there is a sufficient amount of content that is noticeable by users. Batching content is something that was outside the scope of this thesis, but is an interesting solution.

6.3.4 Related work 4: Improving Discoverability of New Functionality

“Improving Discoverability of New Functionality : Evaluating User Onboarding Elements and Embedded User Assistance for Highlighting New Features in a PACS.”[7] is mainly focused on the improving of feature discoverability and onboarding efficiency for picture archiving and communication system used by radiologists. The main questions that the paper wants to answer is “can PACS users find and learn about functionality in the product with the help of user onboarding elements and embedded user assistance”. The Author indicates that Sectra, the company that produces the PACS program, has identified that users seem to struggle to find added functionality when the tool is updated to new versions. From the initial research questions, a number of smaller scope questions are asked which relates to how new functionality can be indicated as well as persuasiveness and intrusiveness of the design.

The author finds a number of previously realized problems that they want to find solutions to, mainly: people often don’t want to leave or interrupt their workflow to go to an external place for help or documentation; How to get users to change their workflow, to what you think is the better process, i.e. how to solve the “Motivational paradox” from a development perspective; and avoiding psychological pitfalls such as presented above, as well as overflow of information. The solutions to the questions asked are used to create a design proposal for embedded feature notifications and onboarding elements.

The paper uses two main methodologies, Case study and Research through design, to answer its research questions. To create a concept, the author performed interviews with various Sectra staff members. Four concepts are generated, Educational Walkthroughs (in a Test System), hints about Locked Features, Integrated UserDoc, Highlighting New, Updated and/or Undiscovered Functionality. After evaluation, concept 4 was chosen (Highlighting New, Updated and/or Undiscovered Functionality) and implemented as the scope of the paper. To determine how concept 4 should be implemented, different visuals were tested as to which association they brought to respondents. The final prototype contains a visual cue together with textual information explaining what the visual cue indicates.

As a conclusion, the paper clearly emphasizes that embedded visual indicators and tooltips were found useful, as well as not having to leave the program. One main point brought up is to focus on just enough information in tooltips and onboarding as not to overwhelm the user. The design of embedded cues should be smaller and not attention grabbing as not to interrupt a user. The general idea of embedded onboarding and user assistance holds potential for improving feature discoverability and usability.

Since the PACS tool is a purchased tool used by professionals, it seemed that the conclusion drawn in the thesis is at least in one way applicable to the product studied in our thesis.

The author is quick to lay out the research questions that will be answered in the paper, there is no practical evidence presented of the cause of the problem other than that Sectra has identified the presence of it. This was similar to our case and presented us the opportunity to complement the author’s work.

The initial quandary is interesting, and one we also pursue answers to, though due to different reasons, there are some parts that I think the paper could expand upon. The paper often seeks to move away from the “old” information system, the one that is separate from

the program itself; how it works today is not explored much. Taking inspiration from this paper, it seemed relevant to explore what changes need to be made to the development and documentation process to implement more embedded user onboarding and tooltips. This also works into the Just-enough-information approach and how a feature without a sufficient scaffold of information and documentation can fall out of favor. This prompted this thesis to continue to explore phenomena in a rapid release environment at the case company.

6.4 Future Work

With a thesis limited to one study period, there were many aspects that had to be left unstudied. In this chapter, we will cover how this subject can be expanded. Both at the case company and more generally for other companies or organizations.

To get a better understanding of how the automatic notification system presented in chapter 5 should be implemented, the users of the product should be interviewed. Right now, the design proposal is based on more general users. To make it as effective and appreciated as possible, it should be designed around the specific user's wants and needs.

The current proposed implementation in chapter 4 adds overhead to the developers to create documentation and information. An interesting solution to this is to use large language models to generate this information instead. It could be done by using aggregated commit messages to produce a text of what is included in the feature. This would be a good solution since it doesn't increase the workload of developers and still makes sure that documentation and information is done when the feature is fully developed.

Implementing a prototype of the design specification that is suggested in chapter 5 is a very logical way to move forward with this concept. This would test how users interact with the more current information regarding features. It would also test if it's viable for developers to write user facing documentation directly, or if it has to go through a technical writer.

One way that this thesis subject could be expanded upon at the specific case company is investigating if there is a possibility to expand the currently implemented forum, and use that to notify users. We only found out about the forum and this possibity in the later work of the thesis, and didn't have time to investigate this possible solution.

Chapter 7
Conclusion

From our research, it is clear that feature discoverability can suffer when a team works with CI/CD. Because of the incremental improvement that is the core concept in CI/CD. Therefore, how users are supposed to find features should be one aspect to conciser when moving to this development methodology.

The design proposal that is suggested in chapter 5 is designed to work with the setup at the case company. However, there are general guidelines that should be followed in both the case companies setup and the general setup. The information presented should be embedded into the software to help users find the information. There is evidence that embedded user information is better in both literature and through our data collection.

If the embedded user information is going to be more than visual, it is important that the text can be generated in a continuous way. Since an important factor when adopting CI/CD is that you want to reduce lead times in the development, features can't be severely delayed because of missing texts. Therefore, there might also be some possible process changes that will be required, because developers will have to take some responsibility for producing texts. This will add some overhead to the developers work but will allow features to be released quickly.

Finally, the design proposition is purely theoretical. There was no time to build, test and evaluate it. Which means that we can't be sure if the design proposition solves the initiating problem, but since it's build on relevant data, it should at least alleviate it.

References

[1] Per Runeson and Martin Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical software engineering* 14 (2009), pp. 131–164.

[2] John Zimmerman, Erik Stolterman, and Jodi Forlizzi. “An Analysis and Critique of Research through Design: Towards a Formalization of a Research Approach”. In: *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. DIS ’10. Aarhus, Denmark: Association for Computing Machinery, 2010, pp. 310–319. ISBN: 9781450301039. DOI: 10.1145/1858171.1858228. URL: https://doi-org.ludwig.lub.lu.se/10.1145/1858171.1858228.

[3] Bent Flyvbjerg. “Five misunderstandings about case-study research”. In: *Qualitative inquiry* 12.2 (2006), pp. 219–245.

[4] J.A. Osorio, M.R.V. Chaudron, and W. Heijstek. “Moving from Waterfall to Iterative Development: An Empirical Evaluation of Advantages, Disadvantages and Risks of RUP.” In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (2011), pp. 453–460. ISSN: 978-1-4577-1027-8. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.6068383&site=eds-live&scope=site.

[5] Abrar Mohammad Mowad, Hamed Fawareh, and Mohammad A. Hassan. “Effect of Using Continuous Integration (CI) and Continuous Delivery (CD) Deployment in DevOps to reduce the Gap between Developer and Operation.” In: *2022 International Arab Conference on Information Technology (ACIT), Information Technology (ACIT), 2022 International Arab Conference on* (2022), pp. 1–8. ISSN: 979-8-3503-2024-4. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.9994139&site=eds-live&scope=site.

[6] Swaralee Nandgaonkar and Vaibhav Khatavkar. "CI-CD Pipeline For Content Releases." In: *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Advancement in Technology (GCAT), 2022 IEEE 3rd Global Conference for* (2022), pp. 1–4. ISSN: 978-1-6654-6853-4. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.9972129&site=eds-live&scope=site.

[7] Rebecca Eriksson. "Improving Discoverability of New Functionality : Evaluating User Onboarding Elements and Embedded User Assistance for Highlighting New Features in a PACS." In: (2023). URL: https://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A1780185&dswid=8256.

[8] Eero Laukkanen, Juha Itkonen, and Casper Lassenius. "Problems, causes and solutions when adopting continuous delivery—A systematic literature review". In: *Information and Software Technology* 82 (2017), pp. 55–79. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2016.10.001. URL: https://www.sciencedirect.com/science/article/pii/S0950584916302324.

[9] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. "On the journey to continuous deployment: Technical and social challenges along the way". In: *Information and Software Technology* 57 (2015), pp. 21–31. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2014.07.009. URL: https://www.sciencedirect.com/science/article/pii/S0950584914001694.

[10] Guoping Rong et al. "DevDocOps: Towards Automated Documentation for DevOps." In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019 IEEE/ACM 41st International Conference on, ICSE-SEIP* (2019), pp. 243–252. ISSN: 978-1-7281-1760-7. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.8804428&site=eds-live&scope=site.

[11] Bill Andel. "Continuous Documentation: Automating Document Preparation with your DevSecOps Pipeline." In: *2022 IEEE 29th Annual Software Technology Conference (STC), Software Technology Conference (STC), 2022 IEEE 29th Annual, STC* (2022), pp. 156–165. ISSN: 978-1-6654-8864-8. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.9951025&site=eds-live&scope=site.

[12] Max Rehkopf. *User stories: Examples and template*. URL: https://www.atlassian.com/agile/project-management/user-stories.

[13] S. DeLoach. "Designing, localizing, and customizing Web-based embedded assistance." In: *IPCC 2005. Proceedings. International Professional Communication Conference, 2005., Professional Communication Conference, 2005. IPCC 2005. Proceedings. International, Professional Communication Conference* (2005), pp. 176–180. ISSN: 0-7803-9027-X. URL: https://www.researchgate.net/publication/4167888_Designing_localizing_and_customizing_Web-based_embedded_assistance.

[14] Rucha Tulaskar. "Evolution of Embedded User Assistance: Considering Usability and Aesthetics". In: *Proceedings of the 2018 ACM Companion International Conference on Interactive Surfaces and Spaces*. 2018, pp. 69–76.

[15] Damjan Vavpotic and Marko Bajec. "An approach for concurrent evaluation of technical and social aspects of software development methodologies". In: *Information and software technology* 51.2 (2009), pp. 528–545.

[16] Rong Guoping et al. "DevDocOps: Enabling continuous documentation in alignment with DevOps." In: *Software: Practice and Experience* 50.3 (2020), pp. 210–226. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=inh&AN=20739565&site=eds-live&scope=site.

[17] Minshun Yang et al. "Identifying Characteristics of the Agile Development Process That Impact User Satisfaction." In: (2023). URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsarx&AN=edsarx.2306.03483&site=eds-live&scope=site.

Appendices

Appendix A

Question for final interview with developers The questions were asked in either Swedish or English, depending on the interviewee's preferred language.

- **Which role do you have in the building advisor team?** Vilken roll har du i building advisor teamet?
- **Do you feel that there is a communication channel from you as a developer which you can use to communicate changes?** Känner du att det finns en kommunikationskanal från dig till användarna där du kan kommunicera förändringar?
- **If yes, how?** Om ja, hur ser den ut?
- **If not, would you want one?** Om nej, skulle du vilja det?
- **What part of the documentation process do you have today?** Vilken del har du i dokumentationsprocessen i nuläget?
- **When you deploy a function or feature to production that is targeted to users, how often do you feel that you could write 1-2 sentences that describe the functionality of what was released.** När du deployar en funktion eller feature till PROD som riktar sig mot användare, hur ofta känner du att du skulle kunna skriva 1-2 meningar som förklarar vad som släppts?
- **In an ideal world the information regarding the features would be automatically generated, however in the end there will probably be more overhead for the developers, since information regarding the released features must come from someone. How do you feel about that?** I en ideal värld hade informationen genereras automatiskt i pipelinen men tyvärr kommer det nog bli mer overhead för utvecklarna, då informationen kring vad som blivit släppt hur det kan användas måste komma från någon. Hur ser du på det?

- **Regarding the added overhead for developers, how would you like this information to be collected?** Angående frågan över hur skulle du vilja att denna information skulle samlas ihop?
- **What kind of information do you think is required to create "just-enough" information regarding a feature?** Vilken typ av information tror du skulle behövas för att göra "just-enough" information om en feature?
- **One thing we have found in our literature study is this example for notifying users, what do you think of this implementation?** Vad tror du om att implementera något liknande detta förslag som vi har hittat i vår litteraturstudie. (The figure shown was figure 5.4)
- **Do you have an example of a program you regularly use that notifies its users in a good or bad way?** Har du exempel på program du använder ofta som notifierar användare på bra eller dåligt sätt?
- **Do you have something more to add?** Har du något mer att tillägga?

EXAMENSARBETE Improving feature discoverability in continuously deployed software products
STUDENTER Love Sjelvgren, Alfred Langerbeck
HANDLEDARE Lars Bendix (LTH), Carl Serrander (Schneider Electric), Åsa Nilsson (Schneider Electric)
EXAMINATOR Per Andersson (LTH)

Påverkan av kontinuerlig leverans på synligheten av nya funktioner i mjukvaruprodukter

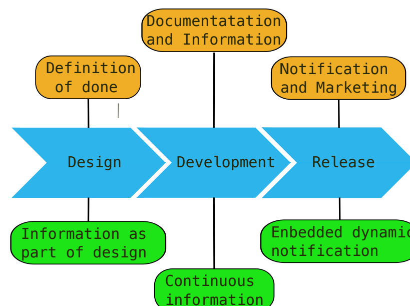
POPULÄRVETENSKAPLIG SAMMANFATTNING **Love Sjelvgren, Alfred Langerbeck**

Många företag strävar efter att utnyttja utvecklingsmetoder som grundas i kontinuerlig utveckling. Detta kan orsaka problem med att användaren inte blir medveten om ny funktionalitet och att dokumentationen uppdateras inte i takt med keverans, vilket detta arbete syftar att lindra.

Vid användning av utvecklingsmetodiken "continous deployment" släpps funktioner och uppdateringar kontinuerligt till användare. Detta kan i vissa fall leda till en rad problem som kan behöva kompenseras för. Det finns två huvudsakliga problem som vi utredde. De två stora problem som tas upp i arbetet är hur dokumentation och notifiering av förändringar i programmet bör hanteras.

Det första problemet som uppstår när organisationer jobbar med små kontinuerliga uppdateringar är att dokumentation blir bortprioriterat. Eftersom att målet med utvecklingmetodiken är att minska ledtiden så släpps funktionerna när de är klara, men det betyder inte att dokumentationen är klar. Därför så måste lösningar som kan fungera med moderna utvecklingmetodiker och generera dokumentation i samma takt som mjukvara utvecklas.

Det andra problemet handlar om notifiering av nya funktioner och uppstår också utifrån valet av utvecklingsmetodik. Detta påverkar användare och inte utvecklare vilket leder till att det ett område som inte har undersökts så mycket. Men eftersom det påverkar användare är det viktigt att undersöka. Ifall inte användare vet att programmet har fått nya funktioner så kommer inte användare använda funktioner. Vilket leder till att värdet som utvecklare skapar också går förlorat.



Problem i olika delar av processen i orange och lösningar i grönt

Resultatet av studien visar att majoriteten av användare behöver hjälp få kännedom om nyligen tillagda funktioner. Klassiska lösningar för att informera användare på så som patch-notes krockar med moderna utvecklingsstrategier där det inte är stora patches utan istället många små uppdateringar. Därför krävs det nya lösningar. Rapporten hanterar även strategier för att skapa dokumentation och information i CI/CD flöden. Det föreslås också processändringar som skulle möjligöra att dokumentationen kan vara klar samtidigt som mjukvaran.