MASTER'S THESIS 2021

Establishing Feedback in Continuous Delivery – Benefits and Approaches

Emanuel Eriksson, Keiwan Mosaddegh

ISSN 1650-2884
LU-CS-EX: 2021-02

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-02

Establishing Feedback in Continuous Delivery – Benefits and Approaches

Emanuel Eriksson, Keiwan Mosaddegh

Establishing Feedback in Continuous Delivery – Benefits and Approaches

Emanuel Eriksson
mat14ee1@student.lu.se

Keiwan Mosaddegh
ke2476mo-s@student.lu.se

February 15, 2021

Master's thesis work carried out at Verisure Innovation AB.

Supervisors: Jon Nessmar, jon.nessmar@verisure.com
Lars Bendix, lars.bendix@cs.lth.se

Examiner: Emelie Engström, emelie.engstrom@cs.lth.se

Abstract

It is generally understood that feedback is the oxygen of continuous software development systems. When software velocity is high, continuous feedback can ensure that quality is maintained without sacrificing throughput. However, the precise impact and value of feedback in modern development paradigms like CI/CD is not thoroughly explored. Distinctions between different types of feedback, as well as descriptions of how a feedback loop is to be established, are incomplete in literature.

Based on literature analysis and data gathered from a company transitioning to CD, we have attempted to categorize and evaluate the urgency and utility of different types of feedback within CD – specifically distinguishing between process and product feedback. We also explored the value of feedback by analyzing feedback-related problems at the company. To address a subset of these problems, a number of approaches to feedback design in practice were evaluated.

Overall, our results show that any generalizable feedback system, even a rudimentary one, is an extreme necessity to achieve sustainable Continuous Delivery. This is especially true when multiple teams cooperate, as one improvised ad hoc solution per team is likely to hinder comprehension across teams. In practice, this system should be centralized but tailorable after specific team needs.

Keywords: Continuous Delivery, Continuous Integration, Feedback, Feedback System, Continuous Testing, DevOps, Software Engineering

Acknowledgements

First of all, we would like to thank our academic supervisor Lars Bendix for his invaluable assistance, not to mention his extraordinary patience with us, throughout the course of this thesis. His knowledge, guidance, and Danish sense of humor all served us well in the past months. Most importantly, his method of using ridiculous metaphors and his mysterious hints will not be easily forgotten.

We would also like to defy explicit instructions and – one last time – thank our friends at Verisure for letting us take so much of their invaluable time. Thanks to their support and expertise, completing this thesis went from dream to reality. Jon, Mattias and Olof really did their absolute best to make us feel at home and welcome, and also treated us with the utmost professionalism. Thank you for making us feel like colleagues, and good luck in your future endeavors!

Contents

Chapter 1

Introduction

Continuous Delivery (CD) is a software development methodology used by practitioners with the aim to increase the speed and frequency in which software is released, all while maintaining or even increasing its reliability [6]. It allows developers to write code, commit changes, receive feedback, and reach a production-ready state, asynchronously and independently of other stakeholders and teams.

However, an increase in software development speed also requires other aspects of the system, such as feedback, to achieve sufficiently high quality and speed. This is important in order to fulfill the information needs of a faster moving work environment. Nonetheless, the exact function and value of feedback in paradigms like Continuous Delivery has not been thoroughly explored yet. As a consequence, some companies transitioning towards CD risk underrating the impacts of mismanaged feedback.

Verisure Innovation AB (Verisure) is a provider of home security solutions. All Verisure products and services are developed at their Innovation Center in Malmö, where around 400 employees work. Previously, the company consisted of approximately 20 applications that each month were assembled together into a monolith-like infrastructure, which was thereafter deployed to production. Today, Verisure have expanded in scope to over 120 applications. However, their work processes have not evolved accordingly, but have rather remained stagnant throughout the years. Some of these practices are considered hindering or obsolete by employees. Especially the release processes at Verisure have with time become disreputable, as the assembly phase at the end of each month has grown into an inconvenient, overwhelming, and error-prone event.

Verisure has realized and acknowledged the issues of their current synchronized releases, and have initiated the journey towards adopting the software development discipline Continuous Delivery (CD). They are currently investigating what aspects of CD to prioritize, in order to achieve a long-term sustainable solution.

1.1 Problem Statement

With Continuous Delivery in development at Verisure, the question regarding feedback in CD becomes pertinent. It is generally understood that feedback plays some important role in software development, but how one should proceed from this conjecture is usually unclear. This is especially the case in fledgling paradigms, as the actual impact and value of feedback in modern development systems such as CI/CD is not thoroughly explored. Without the underlying layers of motivation for what value and impact feedback brings, an adoption roadmap with the appropriate implementation steps becomes significantly more difficult to construct.

Distinctions between different types of feedback, as well as descriptions of how a feedback loop is to be established, are fragmented and incoherent in literature. If different types of feedback were documented, companies could analyze the applicability of these types to their context, and scrutinize their own information systems for deficiencies. The current situation makes it difficult for companies such as Verisure to utilize past experiences and lessons learned from the success stories of the industry, forcing companies to have to reinvent the wheel when trekking down the path of Continuous Delivery. This in turn stagnates advancements within this domain in the literature, as the compounding effect of building upon one's past experiences does not occur. Some companies do harbor success stories, but feedback insights that lead to triumph rarely make their way into academia. This last statement applies to the value of feedback, the lack of distinctions, and the absence of well-documented feedback approaches.

Together, these three overarching problem domains form the point of entry into this thesis:

- The precise value and purpose of feedback in CD is not known.
- There are no clearly usable distinctions between different types of feedback.
- Practical approaches for feedback in CD are inadequate and fragmented.

1.2 Research Questions

Around these problem domains, three main research questions were formed. The goal of this master thesis was to investigate and discuss these questions:

- **RQ1**: What is the purpose and value of feedback in CD?
  - **RQ1a**: How are Verisure's feedback systems and practices insufficient in a CD context?
- **RQ2**: What are some factors that are relevant when categorizing feedback in CD?
  - **RQ2a**: What differentiates process and product feedback within CD?
- **RQ3**: How could feedback design be approached for CD?

The process in which we attempt to answer these RQs involves two phases; the problem analysis phase, and the design phase. The problem analysis phase set out to explore and answer RQ1 and RQ2, using data from academia and industry. This information was gathered through literature studies and interviews with the various stakeholders within Verisure. Once collected the data was analyzed and turned into results for the first two research questions. From that, the design phase tackled RQ3, utilizing some results from the previous phase. In this phase, data was collected using a second round of literature studies and interviews focused on design approaches. Finally, data was analyzed and compounded into a set of results for RQ3.

1.3 Thesis Report Disposition

The next chapter of this paper provides a more in-depth look into the background and context of this thesis; it gives acquaintance with central theory, and the steps taken in the methodology. Next, the problem analysis chapter describes what data was gathered for problem analysis, how analysis of that data was done, and what the results were. In the design chapter, the results of the problem analysis phase are utilized together with more data from academia and interviews, to answer RQ3. This is followed by a discussion of threats of validity, related works, and future work. Lastly, the conclusions to the research questions stated in the problem formulation are presented.

Chapter 2
Background & Context

In this chapter we aim to provide a better understanding of the background and context of the report, including related information about the case company, central theory, and what method was used. This is important to establish in order to become familiar with the case company in which the steps were taken and the results stem from. Furthermore, this clarifies the central theory which the report is based on, and makes sure a common perception of the definition of terminology and concepts are communicated clearly. First, central theory is presented, followed by a description of the case company Verisure, and their relation to Continuous Delivery. Lastly, the report methodology is presented and motivated.

2.1 Theory

The motivation behind the Theory section is to become familiar with the underlying theory which the thesis work is based on. More importantly, this section helps understand the interpretation of what the terminology means specifically in the context of this thesis work, and – not to mention – to see that the reader's understanding of the theory is aligned with how we interpret it. Concepts such as Continuous Delivery, Feedback, Monitoring, and Software Analytics will be covered. We are aware that there are many different opinions on the definitions of some of the mentioned concepts, and we will therefore provide a motivating description about how we interpret them in the context of this thesis. First, modern software development processes will be presented, including descriptions of Continuous Integration, Continuous Delivery, and DevOps. Finally, in Information and Feedback in CD, concepts such as Continuous Feedback and Continuous Monitoring will be explained.

2.1.1 Modern Software Development Processes

Software development is a constantly evolving cooperation between people, tools and processes, with the end goal of delivering a cohesive and functioning product to users. In the

grand scheme of things, the field is very young and immature – there are often few well-established best practices that have stood the test of time. The advantage of this, however, is that the way software is created can be continually improved. Software engineering is not held back by tradition or convention, and often is not limited by aspects like material costs and transportation like other industries are. The consequence of these characteristics is that unlike other fields that deliver products, software development is free to experiment with new strategies and optimize at a comparatively incredible pace.

In the recent decades, software paradigms have quickly shifted from the traditional waterfall model, to agile principles, and finally towards the "continuous" craze of the past years. This transition can be explained in a multitude of ways, but for the purposes of this thesis, the most interesting factor to consider is the speed of changes propagating through the product delivery process. Common to all processes is the sequential set of procedures that turn raw materials into a complete product, often known as an assembly line or pipeline. In the context of software, the velocity of this pipeline has steadily increased, reducing development cycles from years, to months, to weeks. Software products, updates and changes now reach customers faster than ever before, with some systems having cycles as short as minutes or hours.

The first part of the continuous puzzle is often called *Continuous Integration* (CI). This model has many interpretations, but for the context of this thesis, CI is a system of cooperation where individual developers commit their code changes to a version control system that contains the code base of the entire team [3]. In this first half of the pipeline, small changes are integrated, built and tested by the tool infrastructure, preferably automatically. Another characteristic of CI is the fact that developers are essentially free to integrate their code whenever they want to. CI has quickly propagated throughout the software industry, as its benefits are broad – most are not within the context of this thesis.

The speed of these pipelines is usually bottlenecked by the slowest or the least frequently performed step. Even if the majority of the CI software pipeline is fast – development, building artifacts, and testing – releasing to customers at a slower pace will impact the velocity of the entire system, and the quality of the product. This is especially true in cases where multiple teams are contributing to a complex system of cooperating parts. If teams merge their changes together in a synchronized manner at longer intervals, much time is spent on ensuring those changes combine peacefully. The fact that this is troublesome is somewhat counter-intuitive – if all changes need to be tested together eventually, why not dedicate time before a scheduled release to ensure that the product works? The issue is not the idea of quality assurance, it is the method.

In a company where developers are bottlenecked by a slow release process, they often contribute their changes fast and move on to implementing new changes. This means that if issues occur before release, significant time has passed. Developers then have to context switch back to the world of weeks ago to try and address any problems, and when changes from many weeks of work are combined late, issues are plentiful. This delayed feedback can complicate an already intricate development process. Essentially, short development cycles can deliver great value to customers and developers alike, but are difficult to maintain in practice. The entire process of short iterations can be bogged down and jeopardized if one part of the pipeline slows down the others.

The natural step forward is to extend the continuous mindset of CI even further – if developers can integrate their code with their own team freely, why do not teams integrate

their changes with other teams freely as well? As long as quality is ensured, a single line change from a developer could technically travel from their computer, through the pipeline, all the way to customers in an asynchronous manner. This second part of the continuous puzzle is called *Continuous Delivery/Deployment*, and is the coveted and highly desired destination of many software companies today [1]. In the context of this thesis, the focus will be on *Continuous Delivery* (CD). The distinction between delivery and deployment is miniscule, as deployment implies acceptable changes can reach customers without any human intervention at all. Continuous Delivery, however, involves that a quality assured change needs one manual stamp of approval before it finally leaves the pipeline.

Reaching CD in the context of a single team at a start-up company is comparatively straight-forward, due to the simple nature of the code base. If the entire product is contained in one version control repository, extending CI to CD is just a matter of implementing the appropriate deployment infrastructure. At a mid-to-large-sized company, however, sustainable CD is much harder to achieve. A complex code architecture managed by many teams is exponentially more complicated, as sub-components need to be tested together before releasing to production, meaning that a straight line towards deployment is sometimes impossible. Certain architectural approaches exist to mitigate the effects of this complexity – simulated system environments being one. Another strategy that usually combines well with CD is a microservice architecture. Microservices are modular, isolated subcomponents that are loosely tied to other parts of the system, meaning they can be updated, changed or even removed without impacting the rest of the code. Many software companies combine CD and microservices, allowing for the independent deployment of each service.

Continuous Delivery is not only a technical challenge – it is also a challenge of human cooperation. The tight tolerances of CD require stringent engineering and streamlined collaboration between teams and roles. These characteristics are key elements of the summation of practices and philosophies known as *DevOps*. DevOps looks at the problems of this thesis from another perspective, focusing on the human collaboration first, with CD and other things coming as a consequence. Essentially, the goal of DevOps is to further collaboration and unity between developers and operations personnel [2]. The purpose of this is to make the development, deployment and release processes coherent with each other. In DevOps, developers take responsibility for their code changes all the way to production, and operations personnel continuously support and facilitate incoming changes from multiple teams. Exploring the relationship between DevOps and CD is not within the scope of this thesis. However, the characteristics of DevOps are not to be overlooked when attempting to deep-dive into the nuances of CD.

2.1.2 Information and Feedback in CD

Central aspects of both CD and DevOps are information transmission and the existence of feedback loops. Just like the speed of artifacts is substantially faster in CD than in other development processes, the speed of information needs to be just as fast. A direct consequence of CD are the numerous tools that together make up the infrastructure, managing version control, builds, testing and deployment. These tools constantly produce information; collecting, managing and utilizing that information is often discussed as an important aspect of CD. Information also comes from the people utilizing the tools. A natural method of managing this information is pointing it backwards in the pipeline, a concept that is often

called feedback. In general terms, feedback loops ensure that the system reacts to changes, and transmit some information about those changes to relevant stakeholders. In the context of CD, this is sometimes called *Continuous Feedback* [8]. Exploring the exact value and effect of feedback in CD is the central purpose of this thesis.

Other terms exist that cover the transfer of information in CD. One is called *Software Analytics*, and deals with the gathering and analysis of data and metrics [19]. Software Analytics enables companies to gather general insights throughout a software product's life-cycle, using data analysis. Another term, *Continuous Monitoring*, utilizes similar methods to detect issues and perform risk assessment and management [17]. Both terms are highly relevant to CD, as managing and utilizing information is extremely important in high-velocity systems.

2.2 Verisure & CD

In this section we aim to provide a description about the case company of this thesis, and how their context relates to the one of our work. It is important to become familiar with the case company as it helps better understand the environment and circumstances in which the research steps were taken and the results stem from. The contents of the section are an introduction about Verisure, a presentation about their current practices and future plans, and lastly an motivation for why Verisure fit well within the context of our thesis.

2.2.1 Verisure

The aim of this subsection is to attain an initial familiarization with the case company Verisure. This is done by presenting some signifying characteristics about the company, and their work.

Verisure Innovation AB (Verisure) is a medium-to-large sized home security solutions company. Verisure offer sets of products and services, which together compose kits of complete alarm systems, to detect intrusion, fire, water leakage, etc. All of Verisure's products, services, and support systems are developed in-house, at their Innovation Center located in Malmö and Linköping. Approximately 400 employees work (directly or indirectly) with the development of the company's products and services.

2.2.2 Current Practices

This subsection presents the current software development practices of Verisure, with the aim of further building upon the context of this thesis. Having knowledge of the past and current situation of Verisure's software development, facilitates understanding their problems, decisions, and motivations going forward.

Previously, the company developed and maintained approximately 20 applications that each month were assembled together into a monolith-like infrastructure, which was thereafter deployed to production. Today, Verisure have expanded in scope to over 120 applications. However, their work processes have not evolved accordingly, but have rather remained stagnant throughout the years. Some of these practices are considered hindering or obsolete by employees. Especially the release processes at Verisure have with time become disrep-

utable, as the assembly phase at the end of each month has grown into an inconvenient, overwhelming, and error-prone event.

During each monthly release cycle, changes from different areas, teams, and projects throughout the organization, line up within the release pipeline, ready to be assembled as the cycle reaches its end. The hopes of the assembly phase is to seamlessly and effortlessly merge the changes from the more than 120 applications, without encountering conflicts of any kind. This is rarely the case, and due to the magnitude of the changeset, the troubleshooting is not an easy nor convenient task.

Verisure has realized and acknowledged the issues of their current synchronized releases, and have initiated the journey towards adopting the software development discipline Continuous Delivery (CD). They are currently investigating what aspects of CD to prioritize, in order to achieve a long-term sustainable solution.

2.2.3 Verisure's Transition to CD

In this subsection Verisure's transition towards Continuous Delivery is presented. This is done to further develop the context of this thesis, by elaborating on the domain within the company that best relates to the context of our work.

Verisure have recently initiated a transition towards Continuous Delivery, and are still in the early stages of this journey. Up until now change management has been in focus, with a couple of initial meetings, discussions, and workshops conducted, with the aims of clearly communicating the paradigm shift, and receiving feedback about potential concerns that different stakeholders may have. There is currently no CD pipeline within the company, and no such development has been initiated nor planned so far.

The transition towards Continuous Delivery is brought forward by an assigned team (the CD project team). The members of the CD project team have great experience and competence within fields such as configuration management, release management, build chain architecture, and it operations architecture. However, none of the members have any particular prior practical experience within the domain of CD.

2.2.4 Context for Thesis

The aim of this subsection is to closely relate Verisure and the context of the company, to the context of the thesis. This is important to establish, as it explains how certain aspects and challenges arising out of Verisure's current context and upcoming goals, make them a viable case company for our work.

With Verisure transitioning to Continuous Delivery, the topic of Continuous Feedback becomes evident. It is generally understood that feedback plays an important role in CD, but how one should proceed from this conjecture is usually unclear. This is particularly so, as the actual impact and value of feedback in modern development paradigms such as CI/CD is not thoroughly explored. Without fully understanding the impact and value of feedback in CD, it reasonably becomes significantly more difficult to figure out how to approach the topic of feedback, what problems to address, and what solutions to implement.

This scenario is consistent with that of Verisure. Verisure realize that feedback is an essential piece of Continuous Delivery, but lack the understanding of the actual value it can

bring, and what aspects contribute to that value. Without an understanding of such decisive information, it becomes significantly more difficult to delve into the distinct subtopics of Continuous Feedback in practice. What different means of categorizing feedback could apply in CD, and should they be treated differently? Should different types of feedback be directed to different types of stakeholders? Should the way they are – and when they are delivered – vary? These are all questions that become a lot harder to answer when the overarching questions remain unanswered, and leaves Verisure unsure how to approach the feedback situation.

As these areas of research in literature are scarce and fragmented, it becomes difficult for companies such as Verisure, to get help from past experiences and lessons learned from the success stories from the industry. This forces companies to reinvent the wheel when trekking down the path of Continuous Delivery. This in turn stagnates advancements within this domain in the literature, as the compounding effect of building upon one's past experiences does not occur.

This is why we consider Verisure to be a relevant and interesting case company for our research. As Verisure has just entered the phase where the lack of research on feedback in CD becomes apparent, it is possible to explore and investigate their questions and problems as they arise. By examining feedback problems related to CD, at a company who are currently battling against those very problems, we believe we have a greater opportunity to truly understand and address those problems.

To assist Verisure with their journey, this thesis aims to explore three research questions in the context of Verisure. The assumption is that investigating these questions together with Verisure will address the general academic problems surrounding feedback, but also resolve or improve the situation at the company.

- **RQ1**: What is the purpose and value of feedback in CD?
  - **RQ1a**: How are Verisure's feedback systems and practices insufficient in a CD context?
- **RQ2**: What are some factors that are relevant when categorizing feedback in CD?
  - **RQ2a**: What differentiates process and product feedback within CD?
- **RQ3**: How could feedback design be approached for CD?

2.3 Methodology

This section explains what steps were taken in the methodology, with motivations based on the problems and research questions of the thesis. It is important to understand the reasoning behind the steps that were taken, in order to discern how the objectives of the thesis were approached. This section might also be of interest for reproducibility or validation purposes. First, the general structuring of the thesis is briefly described and connected to the problems to be explored. Then, a more thorough explanation of each step with more motivations is discussed.

Overall, the thesis was divided into two phases – a problem analysis phase and a design phase. This division is a consequence of the natural ordering of the research questions, as

the exploratory and investigative results of RQ1 and RQ2 were needed to tackle RQ3. To approach the design of feedback solutions with confidence, the purpose of feedback needed to be probed first. The problem analysis phase also served as a great opportunity to gather insights into the procedures of the case company, clearly grounding the thesis in an industry context.

The goal of the problem analysis phase was centered around exploring RQ1 and RQ2, with valuable insights regarding RQ3 coming as a byproduct. To study the value and types of feedback in CD, a literature study was performed. Further, RQ1a was directly approached using a set of interviews at the case company. RQ2a was explored using the initial literature study, and validated during the interviews. The following design phase was entirely dedicated to exploring RQ3, but required the results of the previous phase to be adequately re-scoped. The design phase was performed in an iterative manner, combining a solution-oriented literature study and interviews with a design process. A visual representation of the complete process of the method can be seen in figure 2.1.

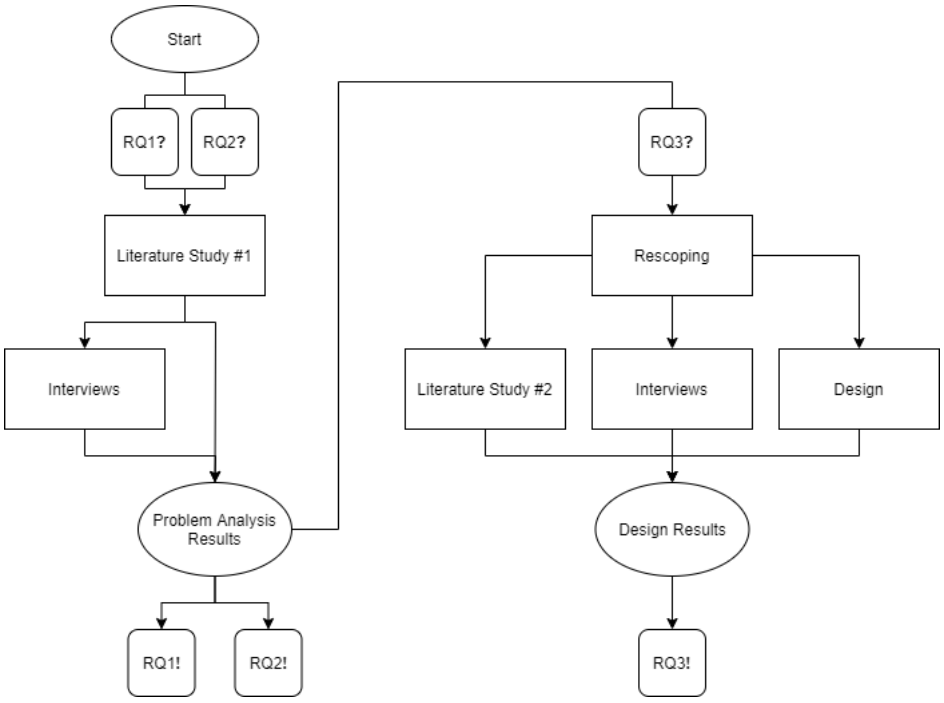

Figure 2.1: A flowchart overview of the thesis work methodology.

2.3.1 Problem Analysis Phase

The overarching goal of the problem analysis phase was to fully answer RQ1 and RQ2. By answering the research questions, parts of the purpose behind the thesis would be fulfilled, and the understanding and insights required to enter the design phase from a better position to answer RQ3, would be achieved. In order to successfully answer the former research questions, a series of steps consisting of data gathering activities had to be performed. This was necessary in order to (1) get a deeper understanding of the current state of research within Continuous Delivery, feedback, and the relation between the two; and to (2) become familiar with the case company, their current feedback systems and practices, and what problems

exist regarding CD related feedback within the company. This data was gathered through a literature study, and interviews. Lastly, to be able to form concluding insights and attempt to answer RQ1 and RQ2, the data collected from the literature study and interviews were then related and analyzed.

Literature Study 1

The reason behind the literature study was to explore RQ1, RQ2, and RQ2a from an academic perspective. The context of the research questions involve understanding the value and purpose of feedback in a CD setting (RQ1), investigating what different types of feedback categorizations that apply well to CD (RQ2), and where and how product and process feedback fit within CD (RQ2a). RQ2a also includes the actual part of defining the meaning behind *product* and *process* in the context of this thesis, as a common definition could not be identified in literature. Part of the outcomes of the literature study were utilized as preparation for the following interview session, as it provided a deeper understanding of the topic of the interview, as well as acted as the backbone of the interview guide.

With the objective of the literature study in place, an approach to find interesting and relevant literature was planned and executed. The idea behind the chosen approach was to combine independently and collectively performed exercises, in order to significantly utilize the collective competence and judgement when identifying valuable resources.

First, collectively, keywords and search terms that characterize the problem domain were defined and listed. Next, the exact same set of keyword searches were made independently by the two of us, through *Google Scholar*. The motivation for choosing this search engine was to ensure the quality and trustworthiness of the collected references. Continuing independently, the articles of the search results were then filtered by if they seemed interesting and relevant for the work, determined by the title. The independently filtered lists of articles were then combined into one, common, longer list. At this point, yet another filtering process followed. Independently, the articles of the combined list were filtered by their abstract, and resulted in two separate lists of interesting and relevant references. The union and the intersection of the two lists were determined, where their intersection functioned as highly relevant references to be studied, and their union functioned as references to study in case their topics increased in relevancy, or the focus of the thesis shifted. In the case of the literature study of the problem analysis phase, the resources within the intersection were of particular interest, as they were perceived as sufficient in order to attain the required understanding about the problem domain and industry examples. The intersected articles were then studied, analyzed, and discussed, collectively. Rather than studying every article completely, only parts relevant to problem collection were analyzed. Most articles, even the ones closest to the problem domain, did not intersect with the thesis context entirely. As such, there was no need to study them extensively.

From the articles, a number of patterns were discovered. These patterns each related to a problem domain, and formed the basis for the interview guide.

Interviews 1

The reason behind the interviews were to primarily gather data regarding RQ1a, RQ2, and RQ2a. The focus of RQ1a is to understand how the feedback systems and practices of the case

company are insufficient in a CD setting. In order to answer this and the previously introduced RQs (RQ2, RQ2a), the interview supporting results of the conducted literature study were utilized to prime the interview structure and topics, so that the desired information could be better extracted from the interview candidates.

For the selection of interview candidates, the aim was to identify employees at Verisure who collectively and rightfully represent stakeholders that show a first-hand feedback interest about the events and processes of the CD pipeline. To effectively identify these individuals, help was provided from the CD project team, who proposed a selection of relevant teams and key individuals within them. Interviews were booked with software developers, scrum masters, release controllers, and first line supporters, from 5 different teams.

The overall focus of the interviews was identifying and synthesizing problems related to feedback. The stakeholders were asked to relate their thoughts on terms like *essential*, *valuable*, *timely*, and *practical*, to better understand what feedback meant in the context of their role.

Semi-structured in-depth interviews were conducted in order to collect in-depth data describing problems and desires at Verisure. An overly open-ended approach would result in incohesive and superficial discussions, and surveys were deemed too strict to be of use. Time-constrained semi-structured interviews seemed like the natural approach to finding this balance. The initial time estimate for the interviews was 60 minutes, and this was trialed during the first interview. Throughout the rest of the interviews, the one hour benchmark appeared to allow just enough time for adequate data collection without an overflow of irrelevant information. None of the conducted interviews strayed noticeably from this benchmark, and at no point was more time required to properly explore the discussed subjects.

Furthermore, the first interview also helped establish a coherent order of the interview questions, and an effective elaboration-inviting phrasing of these questions. Also, the division of responsibilities between the interviewers was naturally formed from the first interview, where one primarily acted as the interviewer, while the other acted as the secretary; taking notes, occasionally adding questions when necessary.

Due to the time constraints of the thesis work, the amount of conducted interviews was limited to 7. However, 7 interviews were sufficient to adequately collect information to answer the related research questions.

Lastly, in order to extract valuable information from the conducted interviews all interviews were transcribed, coded and important insights and patterns were identified and compiled.

Problem Analysis Results

The overarching goal of the problem analysis phase was to fully answer RQ1 and RQ2, including their respective subquestions. With all the requested data collected from the literature study and interviews, an attempt to reach the goal of the problem analysis phase could be performed. RQ1a was best answered by utilizing the case company specific information collected through the interviews. RQ1 as a whole partly relied on the answer of its subquestion, and partly relied on the findings of the literature study, in order to be considered fully answered. RQ2 and RQ2a were mainly answered by analyzing the findings of the literature study, but certain interview answers helped validate or further investigate the formed assumptions.

Thorough analysis of these problems was followed by a validation stage together with Verisure. The thesis advisor at the company, together with other representatives, gave their

input on what problem descriptions seemed accurate to them. Finally, a revised version of the problem analysis results was delivered to all parties.

2.3.2 Design Phase

The design phase had the overarching goal of exploring RQ3, deep-diving into the realization of feedback in the company context. This presented an opportunity to shift tone from the focus on problems to the focus on solutions. However, doing this required correctly taking the results of RQ1 and RQ2 into account. This was achieved, in part, by performing an initial re-scoping. To incorporate company input continuously, a literature study, set of interviews and design process was performed iteratively. Finally, the resulting design was validated with the company, and the results were analyzed in relation to RQ3.

Re-scoping

The purpose of the re-scoping was two-fold; the primary goal was to identify a useful and interesting subset of problems from the problem analysis phase to focus the remaining thesis resources on. The second goal was to coherently transition the results of RQ1 and RQ2 into the exploration of RQ3. To find the appropriate level of depth, the design phase size and scope was limited. By utilizing a problem dependency chart and supervisor feedback, an intersection of problems from the problem analysis phase was identified. The intersection was perceived to cover problems that were both critical to CD, but also not already being investigated by the case company. Thus, the return on investment of the selected problems was high.

This intersection, which became the scope of the design phase going forward, also fulfilled the second goal of the re-scoping – the transition from RQ1 and RQ2 into the exploration of RQ3 was well-established.

Literature Study 2

The purpose of Literature Study 2 was to gather data from a feedback solutions perspective, to answer RQ3. In practice, the method for this process was similar to the method of Literature Study 1.

The greatest difference from the first academic study was the set of keywords used. For the second study, keywords came from the re-scoping, as well as from the exploratory literature study itself. Once an initial set of keywords was used, more keywords were gathered. Another difference was the fact that two academic databases were used, *LUBSearch* and Google Scholar. The filtering process was otherwise identical to Literature Study 1.

Interviews 2

Once data focused on solutions and implementations were gathered, it needed to be validated and reconciled with the context of Verisure. Also, more data specific to Verisure was needed – how should feedback be approached in their context, to be valuable to them, specifically? To do this, a set of interviews were conducted with key members of the CD transition team at Verisure. As the nature of the interviews was more explorative, free-form interviews of about

1 hour were chosen as the method. A total of four interviews were conducted; two concerned the general implementation approach, and two were focused on specific aspects of the design. As the design was iteratively improved, the interviews were distributed throughout the design phase.

Design Process

In order to be able to answer RQ3, data collected during the design phase was utilized to create a proposed design solution. The approach to the design process primarily consisted of a series of iterations of design with subsequent validation. The newly acquired insights from each validation step was then used to reiterate on the design. The design reiteration was either solely based on the previous validation, or encouraged further literature studies and interviews, to form a required ground for the new approach. The designs were continuously validated against the feedback type findings of RQ2, in addition to interviews with relevant stakeholders at the case company.

Design Result and Validation

To gather input, feedback, and validate the contributions of the design phase, a one-hour meeting with Configuration Management and CD enthusiasts and practitioners from the industry was conducted. The meeting participants primarily communicated opinions to validate and expand on RQ3, but did also share their thoughts on the answers to the two former RQs.

Chapter 3
Problem Analysis

In this chapter, the first of the problems and research questions are approached (RQ1 and RQ2). They are approached by utilizing a literature study to gather useful information on feedback from academia, and then performing interviews. Finally, the two data sets are reconciled with each other, to identify where Verisure are lacking in regards to CD feedback. Doing so led to some interesting results concerning RQ1, by giving us an understanding of the value of feedback in CD. Also, the viability of using certain factors (process/product, stakeholders, urgency) as feedback distinguishers was investigated. First, useful data from the literature study is discussed. Then, the data gathered through interviews is analyzed and discussed. Finally, the two sets of data are compiled into a final set of results.

3.1 Literature Study 1

In this section, key pieces of useful information from academia that influenced the thesis are discussed. This data led to some preliminary results regarding feedback types (RQ2), and was also needed to create the company interview guide. What important aspects of feedback are missing at Verisure? It contains a description of typical CD pipelines, and some initial feedback loops within. The definitions of product and process feedback are explored, as well as the distinction between them. Stakeholder interests are discussed as a feedback distinguisher. Finally, typical feedback problem domains are discussed.

3.1.1 The Pipeline Model

The vast majority of software products at Verisure are developed continuously according to CI principles, but released or deployed on a monthly schedule. Understanding what consequences this fact would have on feedback, and data collection, was difficult – there was no clear point of reference for what an ideal CD environment looked like. To address this issue, a model for a general CD pipeline was procured, based on communication with Verisure and

industry literature (Figure 3.1). This model describes what subcomponents a complete CD system should contain, what their responsibility is, and how they fit together.

The pipeline could be divided by the role responsible for each part, developer, tester or operations manager. In practice however, too much overlap exists for this distinction to be useful when modeling pipelines. Instead, the pipeline is divided by subcomponent purpose. This proved to be very useful in identifying what teams to base our interviews around, because some parts of the model pipeline exist as data sources, without necessarily existing in a complete CD system.

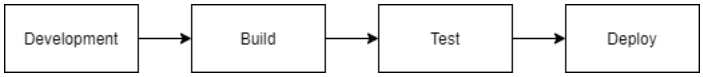

Figure 3.1: A simple representation of the composition of a CD pipeline

3.1.2 Process and Product Feedback

The term feedback encompasses many different aspects of communication, a trend that was noticed in the literature. According to Krancher, et al. feedback is information about actions returned to the source of the actions [8]. To Krancher, et al. the focus lies on the transmission of information backwards. Webster's New World Dictionary calls feedback "a process in which the factors that produce a result are themselves modified, corrected, strengthened, etc. by that result" [11]. For the purposes of this study, we used the following definition:

> *Feedback is the delivery of information to those who need it, with the purpose of enabling them to perform informed and appropriate actions.*

This definition was used and validated throughout the later interviews. At first, it was not clear if all types of feedback are critical within CD. To mitigate this, we distinguish between what we call product and process feedback. Product feedback is insight into how "one specific change" moves through its life cycle. A change in this context can be a single commit to a version control repository, or an artifact being bundled into a complete application. Examples of product feedback is a developer learning that a test failed, or an operations employee learning that an issue exists in production. Both relate to one specific change or changeset.

Process feedback is an overview of many changes over time. Examples of this are graphs of build times during the past sprint, or subjective employee feedback about "the process" itself. This type of information is generally more interesting to stakeholders with oversight, such as managers. This distinction between product and process feedback was necessary because it allowed us to categorize the feedback needs we collected.

3.1.3 Stakeholder Interests

Scaling continuous practices to fit the size of the software system, and cover the needs of all possible roles within that system is not an obvious task according to Rogers and Owen [12]. The difficulties behind such scaling are primarily related to the domain of test activities, and one of the most pressing questions within that domain is what test activities that best

serve the interests of the stakeholders within/around the continuous integration and delivery pipeline. By identifying the different stakeholder interests, and then satisfying them with an adequate test activity, the needs of the stakeholders can be met. However, it is not particularly the test activities themselves that help support the needs of the stakeholders, but rather the information that can be extracted from the different quality gates throughout the CD pipeline; i.e. the feedback about the test activities status.

The reason why categorizing feedback based on stakeholder needs is desirable, is because of its versatility. Feedback based on the stakeholder need ensures that (1) all stakeholders revolving the CD pipeline are included and taken into account when designing feedback solutions, and (2) that the information behind the feedback, and the manner in which it is received, is adjusted to fit each specific stakeholder need.

Mårtensson et al. have designed the Test Activity Stakeholders (TAS) model – a model which shows how it is possible to design the continuous integration and delivery pipeline to include test activities that satisfy all of the different stakeholder interests in the organization [9]. The authors have helped narrow down the different needs expressed by stakeholders throughout 25 interviews, into four comprehensive categories. The reason for making an abstraction into stakeholder needs, rather than categorizing actual stakeholder roles, is because it was identified that different stakeholder roles might have similar and overlapping feedback interests. The four identified stakeholder interests were:

1. Check quality and correctness of software changes
2. Secure stability and integrity in the system during development
3. Measure project progress
4. Verify compliance with requirements or user scenarios

The need for (1) was primarily communicated by developers; (2) came from a variety of stakeholders, but primarily test and QA-related roles; (3) was mainly expressed by project managers; and (4) arose from a set of manager roles, e.g. product owner, project manager, technical manager, etc.

3.1.4 Feedback Problem Domains

The primary focus of this subsection was identifying what type of feedback problems plague complete CD systems, and defining concrete problem domains. Academia has discussed the topic of problems in CD thoroughly, and most research briefly touches on poor feedback as one of them [14] [15] [7]. Data originating from the study was compared to the CD model in Figure 1, and the following problem sources were identified as typical. Once the domains were identified, they were used to build a proper foundation for the interviews. This was done by structuring part of the interview guide based on the domains, with the direct objective being exploring RQ1a.

In-component Problems

This sub-class of problems relates directly to a single continuous subcomponent, its outputs, and its inputs. For example, someone working within the development subcomponent who

lacks feedback from others working within the same subcomponent, or from the development tool directly. Essentially, the feedback is produced within the same subcomponent, and needs to be delivered back to that subcomponent.

Cross-component Problems

If feedback needs to move between two different subcomponents, e.g., from testing to development, then it is part of the cross-component problem domain. There are many combinations of subcomponents, but not all are relevant to the thesis. Focus was predominantly kept on those problems where a concrete stakeholder needed or produced feedback, not a tool.

Organization and Architecture

From the literature, organizational and architectural matters appeared to be a common source of feedback problems. Organizational matters could be questions of legal or access right issues, social feedback barriers, or a lack of a well-defined feedback system. Architectural matters are tied to things like development scale or project size, and the number of dependencies to other teams complicating interactions. These examples, as well as others, were commonly seen as a source of slow or inefficient feedback.

Testing

Another typical problem domain was testing activities. This domain contains all problems related to test quality and execution, as well as issues stemming from testing environments not mirroring production. Lack of automated testing or ambiguous test results, flaky tests, GUI or hardware tests were identified as common issues hampering feedback.

Feedback Execution

This domain relates all issues to the actual execution of feedback, encompassing aspects like timeliness, quality, method of delivery and others. Tools are also an interesting aspect, as well as manual feedback, which appears to be a common source of issues. Feedback distillation also belongs in this category, and over or under-abundance of feedback often causes communication issues.

Traceability

Traceability is broadly discussed in the literature, but is approached from many different angles. The specific feedback issue often noticed in literature was the tracing of a software problem to a specific change or commit. This information was identified as a key element of the feedback loop often missing in company settings.

3.2 Interviews 1

In this section, the data from the interviews is analyzed. This is done so that it can be compared to the academic perspectives on the value of feedback. Essentially, a diff between the

situation at Verisure and academia was performed. To analyze the interviews, patterns in the data from the interviews were identified.

3.2.1 Synchronized Releases

When asked about feedback-related problems, the majority of interview objects brought up synchronized releases as the most explicit source of troubles. The effects of monthly, company-wide deployments to production were clearly felt – especially the consequences of delayed feedback. One interviewee commented: "When release day comes, you have to remember what we even are releasing, which can be quite difficult. Especially when something happens in production, like any issues, you have to think about what this even is – what did we do four weeks ago?". Another developer said: "... When the release is out to customers and is being used, we have already switched focus to entirely new things. It is then costly to switch back when needed. Not only because a long time has passed, but also because you have switched focus and are peering forward at new things.". Another pattern that was commonly discussed was the intensity of feedback being extreme surrounding releases, since issues were most plentiful then. Despite this, asynchronous releases were not always seen as the most natural solution. Some developers were accustomed to their current environment and did not see the need for change.

3.2.2 Context and Architecture Variations

An overall observation from the interviews was the varied context of each interviewee. When probed about their working environment, the interview objects described somewhat similar processes, but very different code architectures. This is due to the broad scope of the company as a whole, as creating software and hardware products that interlink requires architectures of different types. Some developers, especially those working on internal tools and platforms, described a flexible microservice environment. Other developers worked closely to the hardware alarm system, and therefore had dependencies to monolithic legacy systems. The developers creating the website and mobile application had one foot in each world, with parts of the code already moved towards microservices, parts sitting firmly in monoliths. When discussing the effects of monoliths at the company, one interviewee commented: "And it is slightly problematic, things being this way. Partly because nobody is really aware of everything that goes in [the monolith], partly because no single person knows what exactly goes on in there. Releasing from it is quite slow, basically. Testing is sluggish, and so on. That is why we need these windows of code freeze, so we can test and do other things.". Essentially, the interviewee motivated the need for synchronized releases with the existence of monoliths.

3.2.3 Ownership and DevOps

Another pattern from the interviews related to code ownership and the cooperation between departments. Specifically, some managers that presided over that cooperation felt it was poor, and that the steps taken to improve it were unsatisfactory. One of these people said: " In the beginning, the release belonged to the operations department. The development teams developed their code until code freeze. Then, they threw the code over to ITops, 'here you go,

go ahead and deploy, we are starting the next sprint'. Afterwards, they did not care, and when issues occurred around system release, they said 'leave us alone, we are planning our next sprint'. They did not feel the problems were their responsibility.". Another interviewee discussed their view on how this cooperation could be improved, saying "ownership and responsibility is the key here – giving development teams control over [deploys], and also giving them responsibility over fixing any problems that appear. Instead of the "handover" mentality that exists today. … This puts some demands on the developers, to be responsive towards customers, instead of being sheltered like today.". These comments formed two opposing patterns. From the operations perspective, employees felt that developers needed to take responsibility for their changes all the way though release. Developers, on the other hand, felt that the deployment process took too long and hindered their further work by being scheduled to a set date. They also argued that the operations department did not want to relinquish control over the releases. One developer, working in an isolated context from the rest of the company, described how their team had adopted DevOps principles by including two operations personnel as part of their team. They said: "We have two ITops resources, two people that is, doing our releases. As long as our code does not really impact the rest of the company system, we can basically release whenever we want. And yes, that is much, much better. The context of even two weeks ago is much easier to recall than four weeks."

3.2.4 Testing

Some interesting comments were given surrounding the topic of manual testing, and the propagation of automated testing. In general, manual tests were seen as slow, but not always problematic. One developer commented: "If I am finished with something, somebody will start [testing] it immediately at best. Usually, however, it takes longer, a couple of days. Rarely, I will finish something at the start of a two-week sprint, and get it [tested] on the last day of the sprint." When asked if this was a problem, few developers really complained, as they were accustomed to the process. The overall attitude was that somebody else would quality assure code, sometime later – testing and deployment was not seen as part of the developers responsibility. This attitude was heavily dependent on context, however. One specific developer who worked in a microservice context skipped the testing process entirely, so their code could reach deployment as fast as possible. When developers were asked about automated tests, the most common answer was one of hesitant skepticism. Some believed their tests would be difficult or unnecessary to automate, specifically mentioning testing of the monolithic codebase. Another developer said "yes, we could automate as an initial step, but there is a lot of political resistance towards that. Personally, I believe all boring menial tasks should be automated.". Once again, the extent of and opinion on automatic testing was largely dependent on working context.s

3.2.5 Manual Feedback

Another problem domain which the interviews covered was the one of manual feedback. The general perception of manual feedback varied amongst the interviewees. Some had a more modern mindset in regards to software development methodologies, understood the value of continuous and fast feedback, and based their evaluations and answers on it. They acknowledged that manual feedback did not fit in a context where efficient and rational

software development happens, and that the fact that one would have to wait for days to see the results of one's actions was detrimental to the speed and the quality of the development progress. "Receiving feedback right as you have made a change would have been great. That would have been perfect, and would have made work so so much easier. Not just because it is all still fresh in my mind, but also because I would receive relevant feedback immediately. It would help a lot."

However, others claimed that they did not see any problems with manual feedback in their current way of working – "it fits well with our current processes", a developer said. "The feedback loop is not that long, after all. We close our sprint, and then it is ended. If [an issue] arrives after that, sure one would have wanted it earlier, but then it is considered a new issue for the next sprint." Also, the previously addressed issue of context switching was considered insignificant: "It is a part of the job, I would say. Of course it would have been more beneficial to receive [the feedback] right away through automation, but I have never worked that way before, and I am used to [manual feedback] taking its time."

3.2.6 Feedback Systems

Each interviewee was asked about the documentation surrounding feedback strategies, and whether feedback was performed in a systematic manner. Within teams, feedback was performed in an impromptu manner, utilizing direct communication to close team members. "When it comes to the team we have different roles, and very open channels of communications. That is our solution, but it is manual work.", one interview object said. The only systematic approach to feedback between different teams was the use of *Jira*, the issue tracking tool. Otherwise, most feedback was transferred using tools like *Microsoft Teams*, and never in a systematic, well-documented manner. To facilitate cooperation and communication between teams, many employees found themselves in positions of "feedback relays". Their job was to receive feedback from one team, parse it, and transfer it to the appropriate recipient. As the final destination was often unknown, interviewees described having to spend time looking for the correct recipient. One person said "We are trying to set up this kind of processes, that given one type of issue, we do the following, and another type, we do something else. Unfortunately, we are behind on the documentation."

3.2.7 Traceability

A problem identified by most interviewees was the lack of traceability in the development system, especially between issues and commits. The interview subjects discussed how difficult it was to connect events in the pipeline, and how tracing an problem in production back to the developer who caused it was often impossible. They described how this affected feedback, by making it hard to find the correct recipient of information. Some mentioned Jira as a potential source of traceability, but Jira was perceived as confusing and misused.

3.3 Problem Analysis Results

The aim of the Problem Analysis Results section is to explore potential answers to RQ1 and RQ2, by compiling relevant fragments of results identified throughout the conducted

literature study and interviews. By attempting to answer the first two RQs of this thesis, we not only reach interesting conclusions about them, but also provide the basis of which further exploration on RQ3 can be done. As the answer to RQ1 is heavily dependent on the answer to its subquestion, this section starts off by describing the results on which RQ1a is based upon, and how they are compiled to answer the RQ. Next RQ1 is answered by utilizing the results of RQ1a. Lastly, RQ2 is answered. The Problem Analysis section is ultimately concluded in a post-analysis, which briefly summarizes the problem results, and reveals the current thesis problem maturity.

3.3.1 Results RQ1a

This subsection presents the results of RQ1a – "How are Verisure's feedback systems and practices insufficient in a CD context?" The results are categorized after the patterns perceived in the interviews. To compile these categories, the results of the interviews discussing the situation at Verisure were compared to the results of the literature study. Each category includes a general discussion of why the observed practices contradict the principles of CD, and lists specific sub-problems associated with that category.

Synchronized Releases

The impact synchronized monthly releases have on feedback are broad – some developers wait a month for feedback from production. They then have to context switch back to the world of a month ago to try and figure out what caused issues and why, a costly and irritating process. As changes move to production from all over the company, there is often an information overload when everyone needs to receive feedback at once. These issues and others are partly why CD is being implemented in the first place. However, we found that some stakeholders are very accustomed to this process and have designed much of their workflow around it. Essentially, this fact is likely to cause problems down the line as employees struggle to adjust to asynchronous continuous releases. Most of Verisure's current feedback systems are entirely designed around the monthly release, as the rest of this document shows.

- **The philosophy of synchronized releases is the polar opposite of CD,** with slow feedback being a major consequence. Long stretches of little to no feedback are mitigated with intense bursts of confusing information during or near to release days.
- **Company culture is almost completely built around synchronized releases, and some stakeholders view fast product feedback as a superfluous luxury**. Once CD has been implemented, feedback stops being a luxury and starts being a requirement – a reality that will be difficult to adjust to.

Architecture Problems

Verisure currently develops and operates a varied suite of software products, each with its own architecture. The development environments have varying maturities, possibilities and needs. This fact affects both their attitude towards feedback and their transition to CD overall. There is likely no "one size fits all" approach to a continuous system, as the products different production environments deliver are variously critical. Alternatively, an opt in

system of CD could be implemented, with strict requirements on the teams wishing to deploy independently.

- **There is likely no "one size fits all" approach to a Verisure continuous system.** Varied maturity means varied possibilities and needs – both CD itself and CD-level feedback are hard to define and implement.
- **Monoliths do not coexist graciously with continuous practices.** When test suites of monoliths become too big, the possibilities of running them continuously become non-existent, as they require too much time and resources. An example of this is the Verisure API tests, which can currently only be run 1-2 times per day. This means that many changes are made in between each test, and the issue/commit traceability is lost. Without continuously run tests, continuous feedback is hindered, which is important in a CD context.
- **None of Verisure's software products currently qualify for true CD, as they lack the loosely coupled, isolated architecture needed for proper continuous development.** Even PF, which operates in and around microservices, has dependencies to monolithic codebases.

Ownership, Responsibility, and DevOps

Lacking ownership is a sentiment that permeates programming and other areas of development, which might be a contributing factor to teams not feeling responsible for the quality of their code. Without being responsible for quality, teams will not strive to produce done-done code changes, and thus will not mind not getting the feedback necessary to do so. This will not be sufficient in a future CD context.

- **Teams feel they lack ownership over their deployments.** They argue that they are encouraged to adopt automated testing, microservices, etc., but are not allowed to take further steps into autonomy.
- **IT Operations feel teams do not take responsibility for their contributions.** This issue is related to the one discussed above, but from another perspective. The same motivations apply.
- **The overall cohesion between Dev and Ops is low and clearly falting, resulting in poor communication over all.** Poor communication results in poor feedback; something that is insufficient in a CD context.
- **Contributions by Dev are being "thrown over the wall" with the attitude that they will be quality assured,** and that potential faults and bugs will be taken care of further down the line. This is further linked to the above Dev/Ops relationship, and the same motivation for feedback applies here.
- **Developers lack an understanding of what IT Operations need from them in order to go through the release process with confidence.** In other words, there is a lack of feedback on the quality of the deliverables provided by the developers. In a CD context, such feedback ought to be established.

- **Teams are responsible for services in which other teams can make changes to without notifying the responsible team.** This makes it difficult for the responsible team to discern what has changed and needs to be tested as a new release approaches. Such feedback is needed in a CD context.

Testing

Testing in continuous practices should be reliable, comprehensive, fast, of high quality, and continuous. Without these properties, information and feedback flows are negatively affected, and will not be sufficient enough to perform well in a CD context. The following identified problems in Verisure's current context all relate to deficiencies in information and feedback flows in a CD context.

- **A future-proof testing strategy exists, but it has not propagated through the company yet.** While this might just be a matter of time, some parts of the company will have a difficult time adjusting to or implementing parts of the new strategy. More effort needs to be put into ensuring that the company actually follows a coherent testing strategy, so that cooperating teams have similar understandings of what is considered "done-done".

- **There is an overreliance on manual testing.** Manual tests are slow, inconsistent, and prone to human error. In a CD context, information and feedback flows need to be continuous. Manual testing does not fit those requirements.

- **The existing automatic tests are slow, inefficient and do not play a significant role in testing.** If feedback is oxygen, automatic tests are the lungs of a continuous delivery system. They are currently being trialed to some extent at Verisure, but there is no well-defined strategy of how their adoption should be hastened.

- **The inconsistency between the system environment and production** has been perceived as a problem, as the system testing becomes an unreliable measurement of how well the system will perform in production. In other words, some things end up working in the system environment, but not in production. However, this problem can rather be seen as a symptom of poor architecture. If tests are rigorous enough, and components loosely coupled, there is no need for massive integration tests.

- **There is a lack of confidence that utilized external services are quality assured/tested.** If team 1 (t1) uses another team 2's (t2) service and something fails, t1 can not exclude the probability that the problem does not lie within t2's service.

Manual Feedback

Literature agrees that manual feedback is a major problem in continuous practices. Manual feedback has the detrimental characteristic of clogging otherwise well-established, continuous, process flows and systems, acting as a bottleneck from receiving timely and practical feedback. The following points address identified problems that would hinder the efficiency of a future CD context.

- **Manual feedback is slow, not sustainable nor scaleable.** This is consistently identified as a source of problems within CD pipelines, especially for product feedback.

- **There are individuals and roles whose responsibility is to act as information proxies.** The purpose behind the responsibility is to interpret information from one source, convert it for, and then explain it to another person. Another responsibility is to act as a router, as in determining who is the appropriate receiver for the information.

  - As there are no defined systems for these information flows, the work has to be done manually. Manual work is boring, and therefore error prone.
  - Additionally, the lack of defined systems results in a lack of communication protocols, which means that the proxies/routers often receive irrelevant and/insufficient information. This leads to latency, as a lot of back-and-forth communication is necessary to collect the necessary information.
  - A lot of back-and-forth communication also occurs when the proxy/router forwards the information, as what is perceived as the appropriate receiver not always is the case. Sometimes the perceived appropriate receiver helps guide the proxy/router right, other times they do not.

- **Difficulties in evaluating the importance of manual feedback through information channels, such as Teams.** There are channels where employees are to post whenever something critical happens, but there is nothing ensuring that the problem behind the manual message of an employee actually meets the criteria of what is considered accepted posts of that channel. As some, channel responsible employees, turn on notifications in order to quickly act on the posts, there can be a lot of "false alarms", which might motivate them to instead "silent" the channel.

Lack of Well-Defined Feedback System

The lack of well-defined feedback systems and information flows ties in to manual feedback, but is critical enough to be important on its own. In a small startup context, with smaller teams, and less dependencies, impromptu feedback in a spontaneous manner is sufficient. However, in larger companies, especially where many teams need to coordinate efforts, a rigorous framework for feedback and communication overall is required. Much effort is spent on identifying who needs to receive information, and what exactly to deliver to them. Once that person is found, they often hear back with clarifications, leading to information pinging back and forth repeatedly. The problems identified below affect the quality of information and feedback flows, and are decisive in their ability to function in a future CD context.

- **There is a lack of a proper feedback system.** Sometimes developers want to know when a build succeeds, and sometimes when a build fails, however receiving both when only interested in one is annoying, and is considered to be spam by the developer. Striking this balance is difficult, but important to achieve a sustainable continuous system.

- **There is too much trust and dependency on key individuals in the current information flows.** As no proper feedback systems are defined, but rather consist of soft information points, there is a great reliance on said individuals' expertise and communication

for the system to function. As the reliance is on individuals, and not on systems, there is a high risk involved as individuals might change roles, quit, or become unavailable due to other reasons.

- **Due to the lack of clear feedback channels, communication is comparatively slow and inefficient.** Time is wasted on repeated queries to incorrect individuals, especially, when it is unclear who to contact or what to do when something occurs.

Traceability

There is a lack of traceability. When a problem arises, it is difficult to trace it back to the responsible code, commit, or even issue. Without such information, taking a deliberate step towards resolving the issue becomes difficult, and results in poor feedback flows, without a clear message nor receiver of information. This in turn leads to confusion and diffuse decision making, which stalls advancements in development. This is inadequate in a CD context.

- **Difficult to establish what went wrong in production,** why that happened, and who is responsible for the error. This relates to the problem above, but emphasizes on the importance of proper feedback flows from production. However, the same motivations also apply here.
- **Metadata does not follow artifacts inside the pipeline.** Easily accessible and necessary information does not exist "close" to each artifact, but requires effort or communication to find.

3.3.2 Results RQ1

This subsection will use the results of RQ1a to discuss RQ1 as a whole. To reiterate and remind, RQ1 is "What is the purpose and value of feedback in CD?" First, the results of the Verisure-focused sub-question will be summarized, in the context of the broader question. Then, the final results concerning RQ1 will be discussed.

The results of RQ1a clearly show that Verisure's current feedback systems do not support a broad-scale CD environment. This is best summarized as feedback being somewhat undervalued and misunderstood, across the board. Some stakeholders at the company understand the role of feedback, but applying it in practice has proved difficult. The feedback systems that exist are built around the current velocity of the pipeline, and to some extent falter even today. If the company wishes to transition into sustainable CD, attention needs to be put on how feedback is approached within the pipeline, and how it is relayed between cooperating teams.

Analyzing the insufficiencies of a specific company's feedback approach gave us great insight into the overall importance of feedback in continuous software development. By exploring what problems can occur due to poor feedback, and how the entire development process can slow down, vital lessons were learned. These lessons allowed us to reach conclusions regarding RQ1.

CD requires a robust infrastructure of tools to even achieve software development. But a similar level of robustness is required when dealing with how information is transferred, too. This is especially true when the velocity of the pipeline is high, and artifacts flow from

coders to customers every single day. Essentially, the effect of feedback in CD is akin to the purpose of oxygen within the human body. Limbs that receive less oxygen perform poorly, and sicken quickly. If the brain loses oxygen, the entire body shuts down. The same is true in the context of software. The purpose of feedback in CD is to deliver necessary and useful information to the correct stakeholders, at the appropriate time. Using this information, they can independently make decisions and contribute changes that do not jeopardize the continuous nature of CD. The value of feedback is to support the entire continuous software life-cycle, and to enable the cooperation of teams at a high velocity. Feedback gives independent teams knowledge about their own domain. Equally importantly, desirable feedback also provides necessary insights into the domains of the nearby teams, especially the ones that are affected by the original team's changes.

3.3.3 Results RQ2

This subsection aims to compile an answer to RQ2 (What are some factors that are relevant when categorizing feedback in CD?) and RQ2a (What differentiates process and product feedback within CD?), by presenting the different CD relevant feedback types derived from the results of the literature study and interviews. First, factors that are identified as relevant when categorizing feedback in CD are listed. Next, there is a brief elaboration on each factor, followed by a motivation to why it is considered relevant in CD.

Some factors that are relevant when categorizing feedback in CD are:

- Stakeholder interests
- Urgency
- Product & Process

Categorizing feedback based on stakeholder interests is relevant to CD, as it ensures information needs vital to each stakeholder closely related to or dependent on feedback from the CD pipeline are supported. Stakeholder interests as a means of categorizing feedback is perhaps not only applicable in the context of CD, however its benefits become clear as the speed of processes, artifacts, and their subsequent feedback increases. As the development speed increases the need of a stakeholder to be able to conveniently follow along and stay updated with a specific change, or the overall software development progress, becomes clear. If this has not been properly considered at this point, the ability to support those needs become significantly more difficult.

Another factor relevant to CD is urgency. Feedback is categorized based on how urgent the reception of the information is. This becomes relevant for CD, as when the amount of information flowing increases enormously, all information will not be considered suitable or necessary to immediately notify the potential recipient about. By recognizing this, it is possible to only treat the information that a recipient actually wants to be aware of as soon as it occurs, in that manner. This type of information is usually information that plays a decisive part in the recipient's ability to make decisions on the next step forward.

The final factor for categorization of feedback that is relevant to CD, is the distinction between product and process feedback. A more in-depth characterization of product and process feedback is given in subsection 3.1.2. In summary, product feedback gives insight

into the status of one specific change through its life cycle, while process feedback provides insights in what many changes can reveal over time. This becomes relevant in CD, as the two areas of the categorization can be managed in different ways, since they support different interests.

3.3.4 Post-analysis

In conclusion, the problem analysis results adequately explore RQ1 and RQ2. The purpose of feedback in CD is to deliver necessary and useful information to the correct stakeholders, at the appropriate time. The value of feedback is to support the entire continuous software life-cycle, and to enable the cooperation of teams at a high velocity. Factors that can be used when categorizing feedback in CD are stakeholder interests, urgency, and product and process.

After exploring the value and purpose of feedback, and some factors that are relevant when categorizing feedback in CD, it is then reasonable to use that knowledge and start exploring how feedback design solutions should be approached for CD. This is how the research questions of this thesis are connected. RQ1 and RQ2 provide an understanding of the purpose of feedback in CD, what value feedback in CD should bring, and an understanding of different ways to categorize that feedback. The next step becomes evident. How should feedback design be approached for CD? That is what RQ3 is all about.

Chapter 4
Design

In this chapter, the approach to RQ3 is discussed, and the results of the research are shown and validated. To answer RQ3 appropriately, a re-scoping of the problem domain was performed together with Verisure, to identify a cohesive transition from the prior RQs into RQ3, as the focus of the remaining thesis resources. New data was collected from both literature and Verisure, this time focused on solutions and designs. That data was analyzed into a results, which were iteratively validated using the results of RQ1, RQ2, and Verisure. Specifics of the actual value of feedback in CD are used to explore how feedback loops are to be established/approached in CD. The contents of the chapter are a re-scoping, followed by data collection from literature and Verisure. Finally, the results of the design phase are discussed in relation to RQ3.

4.1 Re-scoping

To focus the remaining resources of the thesis on a coherent and realistic subset of practices at Verisure to explore a feedback approach for, a rescoping was necessary. In this section, the process of re-scoping our problem domain leading into RQ3 is discussed. At this point in the thesis, with RQ1 and RQ2 addressed, it was time to investigate new problem domains. A natural continuation on the previously covered problems would be to focus on solutions going forward. Specifically, given the results of RQ1a, a useful area of discussion could be what feedback in CD should look like, to actually fulfill the value described by RQ1. However, the topic of feedback loops in practice has not yet been thoroughly explored by academia. Therefore, a strict set of best practices or a complete solution is difficult to derive.

To approach this problem domain, we decided to explore how CD feedback design should be approached, using the results from the previous phase to bolster our conclusions going forward. Essentially, what does a system that provides valuable and useful feedback look like? How should it be approached to cover the different types of feedback discussed in RQ2? These questions formed the basis of the resulting RQ3:

***RQ3**: How could feedback design be approached for CD?*

After this question was formed, the connection to the previous problem analysis phase was further developed for the context of Verisure. Given the set of workplace practices identified by RQ1a, exploring approaches to solutions that resolved or improved some of the issues of RQ1a seemed an appropriate way of tackling RQ3. Addressing all of the subproblems in RQ1a, however, was deemed both impractical and challenging. To reduce the range of the design phase, a re-scoping was performed together with Verisure. The goal of the re-scoping was to identify a coherent and realistic subset of practices at Verisure to design for.

The first way that this was done was by exploring the connections between the problem domains of RQ1a. If dependencies were identified, multiple problems could be improved by addressing one root cause. Examining what characteristics that connect problem domains proved to be a valuable exercise. The results of the exercise can be seen as a dependency graph in figure 4.1.

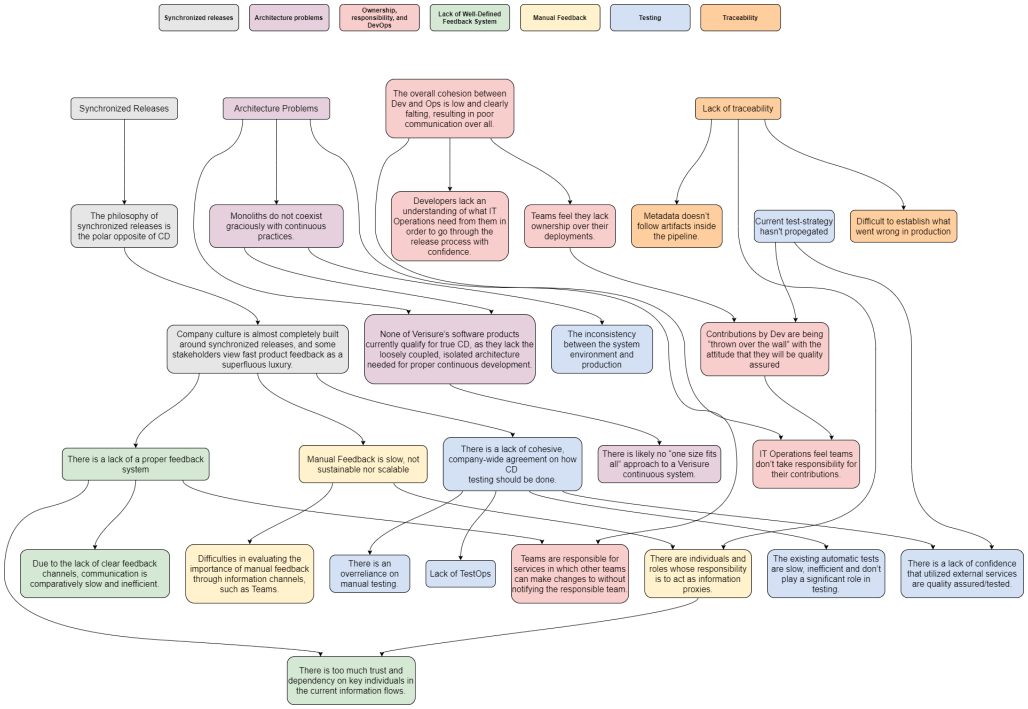

Figure 4.1: A problem domain dependency graph, connecting the different problem domains of RQ1a.

From the dependency graph, two potential directions seemed promising. The first direction focused on testing as a problem domain, specifically the propagation of automatic tests and a company-wide testing strategy. The second direction connected the lack of a CD feedback system with the overreliance on manual feedback. These problems were seen as closely related, and could potentially be resolved with one design – a system of automatic feedback.

The two directions were both appreciated by Verisure. The most valuable input they provided at this stage was the matter of return on investment. The most value we could provide to them was to explore those problems that no one else at the company was looking at. With this input in mind, we evaluated to what extent the Testing domain was under control. This was done through a brief interview with the chief of Testing at the company. From that

interview, we concluded that a future-proof testing strategy does exist at the company, but it has not propagated thoroughly yet. This strategy contained the elements we were looking for, and directly addressed most of the subproblems of the Testing domain. Therefore, the scope of the design phase was limited to the following problem domains: the lack of a feedback system, and the overreliance on manual feedback. By exploring this concrete domain, RQ3 could be approached in a concise manner.

4.2 Literature Study 2

In this section, pieces of literature relating to RQ3 are presented and discussed. Specifically, possible approaches toward feedback design for CD, as well as example feedback systems that exist in literature are discussed. Together, they comprise some initial results to RQ3, but lack grounding in the context of Verisure. First Software Analytics is presented, followed by a continuation on the topic of stakeholder interests. Lastly, insights regarding design composition are presented.

4.2.1 Software Analytics

Insights about Software Analytics are considered extremely relevant to answer RQ3, as the lessons learned from previous implementations of such projects, appear to be applicable for approaching feedback design in CD. While performing our second literature study, we came across Software Analytics [5] [19], a term which we found shared many characteristics with what we had labelled as “Feedback System”. By utilizing our newly discovered keyword, we were able to explore topics related to feedback in CD, that were previously inaccessible due to the limitations of our search terms. The new set of topics also included implementations of Software Analytics projects, and even more importantly the key take-aways from those experiences.

What is Software Analytics?

Software Analytics aims to obtain insightful and actionable information from software artifacts that help practitioners accomplish tasks related to software development, systems, and users. By supporting practitioners who perform tasks related to software development, productivity is improved. Software quality is improved by providing reliability, security, and performance related information to those who perform systems related tasks. Lastly, the user experience is improved through information about user perspectives. There is a broad range of stakeholders who benefit from Software Analysis. Developers, testers, program and software managers, designers, as well as usability, service, and support engineers.

Implementing Software Analytics?

At a first glance, software analytics can seem like a simple, prosperous practice, without any apparent negative implications. However, actually adopting and implementing software analytics in reality involves some significant challenges. Existentially critical parts, such as ensuring that the information is perceived as insightful and actionable, or determining what

problems the practitioners actually care about, are often overlooked or difficult to get right. This often results in a lot of resources being wasted on a project without the target audience's needs properly considered and ultimately satisfied. However, there are some things that can be taken into consideration when implementing software analytics projects, that help make it become successful.

The reason behind the software analytics project, and what is aimed to achieve through it, should be clearly propagated and communicated to everyone involved in or affected by the project. This naturally includes the ones part of the project, but not to mention the ones using the resulting system. This is important in order to quickly become aware of potential inconsistencies or misalignments between what is being developed, and what is actually needed. The procedure of picking data, making observations on it, and returning evaluation results, loses its value when there is no defined recipient of the end results. It is therefore extremely important to first explore what key problems there are to solve, then find the appropriate data to observe, and lastly solve the problems by turning that data into valuable and useful information. This is the blueprint for making sure that the resulting solutions really have a beneficial impact on the practitioner's ability to complete their tasks.

It is not always easy for researchers to extract the underlying key problems practitioners experience in their work. To overcome this issue, and ensure that the right problems are worked on, it is recommended to establish feedback loops between the two parties early on, and aim for many iterations. By iterating on the perception of a communicated problem, and receiving feedback on a proposed solution, it is more likely to eventually unveil the key problem, and doing so with less resources wasted on the wrong issues. Additionally, a trust between researchers and practitioners can be built as a consequence of the feedback loop, as empathy and genuine interest for the practitioners' needs are shown from the start.

As the data for continuous delivery software analytics projects come from many different systems and tools with varying terminology and quality, it can be vital to establish a data standardization early on in the project. This is to ensure that there is a common understanding of how the information extracted from one source, is referred to in the context of another. This is not always a simple task, and also here it is recommended to initiate collaborative work between the researcher and practitioner to figure this out.

A final take-away from the lessons learned about the practical implementations of software analytics projects is to utilize visualizations whenever applicable. Visualizations help guide the audience to more easily understand the value of data, and how that data should be interpreted and related to. This helps practitioners more easily gain a perception of the overall status of their work, and likewise grasp when something needs to be dealt with. Another benefit of visualizations is that it makes the interpretation of data competence agnostic. No matter what role or experience the practitioner has, they should be able to understand the message behind the data, as the data is transformed into information.

4.2.2 Test Activities Based on Stakeholder Interests

In section 3.1, information about different stakeholders and their respective interests and needs from the Continuous Integration and Delivery pipeline is described. With this understanding in place, it naturally becomes interesting to understand how a CD pipeline should be designed in order to support the identified interests and needs.

The first step in doing so is to understand where to find, or how to generate the data

necessary to support the stakeholder needs. As the CD pipeline consists of a series of different test activities at different stages of the pipeline, they can be utilized in order to achieve this. It is obviously not the test activities themselves that are of particular interest for supporting stakeholder needs. What is interesting however, is what the test activities can reveal about the progress of the CD pipeline, and the artifacts within. By extracting relevant data from the test activities, this data can, when properly utilized, be converted into valuable information that can be used as feedback to support stakeholder needs.

According to Mårtensson et al., the CD pipeline can be described in three main phases [9]. Each stakeholder need is then met in one or more of those phases, depending on how relevant or urgent they are in that phase. The three phases are (1) early in the pipeline, (2) later in the pipeline, and (3) release pipeline. (1) represents the phase where development on the team branch is made. When the development is delivered from the team branch to the main branch, a transition to (2) is made. This is where test activities are executed to see if functions and subsystems work together. In (3) the main branch is delivered to the release branch. This phase introduces new stakeholders, who are primarily interested in verifying compliance with requirements or user scenarios.

As mentioned in section 3.1, the four main stakeholder interests in the CD pipeline are to

1. Check quality and correctness of software changes
2. stability and integrity in the system during development
3. Measure project progress
4. Verify compliance with requirements or user scenarios

Checking quality and correctness of software changes is an interest primarily expressed by developers. This is something they want to know as early on in the pipeline as possible, and thus occurs in phase 1 and 2 of the CD pipeline. The feedback from the main branch mainly regards if the changes also work together with the rest of the system. The interest to secure stability and integrity in the system during development comes mainly from test manager related roles. This interest is supported by information collected from system tests, and is relevant both in the team and main branch. Project managers and similar roles primarily need to be able to measure project progress. This information is gathered from unit/component tests and system tests in the main branch, but also towards the end of the pipeline. Finally, the need to verify compliance with requirements or user scenarios is supported by tests and checks conducted on the release branch. This information is primarily requested by technical managers.

4.2.3 Design Composition

As to the composition of an actual design solution, few detailed descriptions exist in academic literature. In the examples that do exist, authors often comment on the general applicability of their results; even if the design performs well in one context, it is unclear if it will in others [16]. From the descriptions, interesting or useful patterns can be extracted. This subsection aims to discuss two specific design examples that we believed are relevant to

RQ3. Overall, the focus will be on general design patterns and approaches, rather than on the technical specifics of each implementation.

The first example composition came from research done at ING Nederland (ING NL) by Vassallo, et al. [18]. Throughout a number of research papers, the authors evaluated the CD system of a large financial organization with pre-conditions similar to Verisure. As part of the CD infrastructure, ING NL developed a continuous monitoring layer sitting on top of the pipeline. The purpose of this layer was, in name, to enable what they call *Continuous Monitoring*. However, we identified that the actual contribution of this layer was to enable many of the functionalities and features we identified as important within this thesis. Specifically, the authors describe the technical composition of the design. The layer sits on top of the development pipeline, and consists of three central functionalities: instantiating or controlling the CD pipeline, collecting measurements from the pipeline, and analyzing those measurements.

The above mentioned functionalities are achieved by utilizing an event bus. Essentially, occurrences in the pipeline that are of interest are transmitted as events across the layer and into a database. These occurrences are events that necessitate product feedback, for example, build failures, crashes or test results. Within the database, analysis is performed to produce the equivalent of process feedback. Using a layer that collects data into a central point of storage and analysis is an interesting concept, and proved a key point of discussion for RQ3. Each tool in the pipeline produces unprocessed information of very different formats and types, a complicated reality that is verified by Huijgens, et al.[5]. The use of events instead of raw data simplifies data storage and analysis considerably. In practice, the monitoring layer and the CD system as a whole were applauded at ING NL.

The second example comes from initial trials of the Eiffel framework performed at Axis Communications. In their thesis, authors Hramyka and Winqvist study the effects of adopting Eiffel in a CI/CD system with the goal of improving traceability [4]. The purpose of the Eiffel framework does not align with the domain of this thesis, as our focus is feedback, not traceability. However, the conceptual structuring that Eiffel provides could be applied to the context of RQ3, especially in regards to the design composition.

The Eiffel framework and accompanying protocol work in a similar fashion to the monitoring layer of ING NL. However, Eiffel takes the event approach even further, by applying the event label to a much broader number of occurrences. By logging a record of essentially everything note-worthy that happens within pipelines, Eiffel provides traceability by connecting events into dependency trees. When the events have clear connections to other events, the journey of an artifact can be easily followed through the entire development process. This idea could be applied to feedback, if each feedback-worthy event was connected to other events. Overall, Eiffel provided us with more confirmation that a centralized solution has worked. Also, Eiffel cemented the concept of events as a better way of information management than storing raw data.

4.3 Interviews 2

In this section, the data collected from literature is reconciled and “trialed” with Verisure, to identify to what extent it was applicable in their context. The Verisure-specific information was collected through 4 separate free-form interviews and discussion meetings, with

team members from the Continuous Delivery project team at Verisure. Topics regarding architectural approaches were discussed, as well as feedback system features, metrics, and requirements. First, the outcomes from discussions revolving centralized versus distributed design approaches are presented. Next, the insights gained about how to approach design robustness is covered. Last, the interests, potential features, and useful metrics gathered from the interviews are described.

4.3.1 Centralized versus Distributed Design

A central point of discussion throughout the interviews was how a feedback system should be structured, on the broadest level. Two approaches were discussed, a distributed and a centralized one, each with distinct advantages and drawbacks. Also, a choice between the two directions had direct consequences on the feasibility of certain functionality.

The first approach, a distributed one, seemed very reasonable given the current shape of the pipeline. One stakeholder described how each tool in the pipeline already had support for some feedback delivery, and most had some data storage functionality. A distributed solution would entail each tool storing what data it can, and providing feedback from its specific domain. To tie the system together, some data from each tool could be piped into a visualization front-end of some sort. The focus of this front-end would be to enable observability, not necessarily feedback or analysis, in the opinion of the interviewee. The subject felt that this concept would not be too different in scope from what exists today at the company, and that it would allow each team to display the information that was relevant to them. The ability to tailor the design was expressed as important.

Landing on the other end of the spectrum, a more centralized design solution was also talked about. One interviewee discussed the fact that while most tools do have some data storage capabilities, this was not true for all tools. Also, some tools only had short-term data storage in a proprietary format. A possibility that was explored was the storage of data within tools, but also inside a central database of some sort. This would allow for analysis and long-term storage to be performed on the data, and also expand on the capabilities of some tools. However, this compromise would entail storing the same data in two places, which was seen as wasteful. A single, central storage and analysis design would cost more to set up, but would enable more intelligent functionality. Stakeholders at Verisure agreed that this direction would open more doors, but once again, only on the condition that the solution was flexible. Some key metrics should be standardized across teams, but the customization after team and stakeholder interests was still necessary. One interviewee commented, “The general aspects having to do with CI/CD are one thing, generic metrics from running services for example, but then the DevOps teams have to create their domain-specific dashboards themselves”.

4.3.2 Design Robustness

A parallel discussion to the question of centralization concerned the robustness of the design. This question was connected to the previous topic, and was brought up in the context of a timeframe. Should a robust, large-scale design be created from the start, or should the goal be a less functional but easier solution? The advantage of a lean and iterative design is that Verisure can get it working relatively fast, and improve it along the way. This idea was valued

by the company, as was expressed as “not having to reinvent the wheel”. Aside from a lower cost of implementation, a stripped-down version of the system could be used to convince sceptics of its value, and the value of feedback overall.

In contrast, implementing more robustness from the start would lead to a more future-proof solution. Although costlier, a proper infrastructure was discussed as more in line with the results of RQ1 and RQ2. That is, more of the core functionality could be trialed and validated by the company, and feedback would not be a risk factor in their journey into CD. However, even if this approach was appreciated, Verisure felt that it was difficult to know what exact tools should be utilized from the start – even if given a list of desired functionality. The specifics of the design would be difficult to implement before more was known about the final CD infrastructure.

4.3.3 Interests, Features, Metrics and Metadata

During the discussions and interviews, a couple of interests, features and metrics were brought up and suggested. The interests and features included elements that the interviewees considered valuable to use and benefit from in a feedback design. The metrics included things which the interviewees had interest in tracking and executing on. Altogether, the listed points are aspects the interviewees perceived as central interests and needs in a feedback system.

Interests and Features

- The ability to monitor the overall software development progress of one or multiple team(s).

  An interviewee meant that the best way to monitor is by representing the information in the shape of graphs and other visualizations. “Without visualizations...”, the interviewee said. “It would be impossible to comprehend anything”. He further expressed that a good idea could be to have several levels of information abstraction. For some, just a switch of a color from green (functioning) to red (not functioning) is sufficient to get what they want out of the monitoring. Others might want to take it one step further, and get the detailed information necessary to track down what caused the color switch.

- The ability to establish thresholds based on different conditions and complexity. These thresholds are triggers when the set conditions are met, and indicate this for example through notifications or dashboards.

  One interviewee found it to be interesting to further look into how much the setup of conditions to track could be predefined and turned into templates. The idea behind this thought was to eliminate as much as possible of the overhead that comes with such a feature, for the teams.

- The ability to receive notifications about events that are of interest.

  It was communicated that it was crucial to have the ability to customize this on an individual level.

The following interests and needs were also expressed as significant by the interviewees. However, there is an absence of an explicit example to support the claim.

- The ability to visualize and monitor the progress of an artifact as it proceeds through the CD pipeline.
- The ability to track what needs and issues that are fulfilled and solved in a deploy.

Metrics and Metadata

It was expressed that some general guidelines to metrics and metadata were that in order for them to be of interest they should fall into one of the following two categories.

- It should be actionable, meaning that its primary purpose is to prompt the practitioner to take action.
- Its presence should primarily bring a sense of control to the practitioner, and secondarily be of use when investigating seemingly related issues or similar.

The majority, if not all, of the expressed examples are considered to belong to the second category. This outcome is not entirely unexpected, as the discussion about metrics and metadata arose in connection with topics about monitoring, dashboards, and thresholds, where data of category two is more common. Some items of the following list could perhaps in some cases be seen as primarily actionable (and thus belong to category one), but that is generally not the case in the context of feedback systems.

- The runtimes of builds, tests, and deploys
- The environment in which an application has been deployed to
- The person responsible for a specific deploy
- The amount of changes done in a commit
- Infrastructure metadata, such as OS versions, etc.

An example of a metric or metadata that belongs to category one could be the status of a test activity. When a test activity fails, there likely exists a responsible stakeholder who would like to take action as a result of it.

4.4 Design Results

The purpose behind this section is to answer RQ3: How could feedback design be approached for CD? In this section, the process of transforming data from literature and interviews into an actual design solution that deals with RQ3 is briefly presented. Also, the initial iterative validation of the solution is discussed, using the results of RQ1 and RQ2. Finally, the resulting solution is discussed, as well as its reception and final validation at the company.

4.4.1 RQ3 Results

This subsection aims to compile an answer to RQ3, by utilizing information from the literature study and interviews of the design phase, that is relevant in order to understand how feedback design could be approached for CD.

Software Analytics, and more importantly the experience acquired from implementing it, is considered highly important for answering RQ3, as it presents a set of preparatory and central steps to consider when implementing software analytics solutions in CD. Due to the similarities between Software Analytics and Feedback System, the lessons learned from previous Software Analytics projects can successfully be applied in a Feedback System setting, to help better understand how to approach feedback design in CD. Ensure that all parties involved in, or affected by, the project clearly understand the aims of the feedback design project. This is important in order to quickly become aware of potential inconsistencies or misalignments, to make sure that the problems being worked on actually solve real, expressed, and observed problems. Feedback is key, and no exceptions are made especially when implementing a feedback system. Aim to as early as possible establish a feedback loop between the researchers of the project, and the practitioners whose interests and needs the researchers attempt to support. The iterative process of understanding the problems, and receiving feedback on solutions, leads to a higher quality end result, less resources spent on the wrong things, and a healthy relationship between the two parties.

The reason why subsection 4.2.2 is important to answer RQ3, is because it helps unveil what stakeholder interests exist around the CD pipeline, and how the desired information to support those interests can be generated through different test activities within the pipeline. By taking the stakeholder interests into consideration in the beginning and end of the Feedback System cycle, it can be ensured that the interests stay consistently in focus throughout the cycle. This stakeholder interests centered model can be considered when approaching feedback design for CD.

Another clear result concerning RQ3 was that a centralized design is better at supporting the required functionality of an automatic feedback system. This topic was thoroughly discussed in the interviews of the design phase, as it was an appropriate starting point. From the interviews, we concluded that a distributed solution would be very similar to the current feedback approach at Verisure. However, that solution is clearly not sufficient in a CD context, as shown by RQ1a. Therefore, a better approach is to design around a central data management system of some sort. That system is not only more future-proof, but also easily scalable. The specific functionality discussed in this subsection is much easier to implement and realize with a centralized approach. Also, this approach is somewhat tool agnostic. This characteristic is vital, as tools can change, especially as the actual CD solution is improved and optimized. A centralized approach does not limit a feedback systems functionality after the chosen tools, either. It also achieves a great mix of standardized, but tailorable – which was valued as crucially important by Verisure.

Some conclusions can also be reached in regards to design robustness. As this discussion is connected to the question of centralized versus distributed, it is natural to explore how a centralized design should be approached. It is clear that robustness is more appropriate, given the importance of feedback in CD. As the conclusion of RQ1 directly shows the value of feedback, going with the lean route risks missing the target, and thereby jeopardizing the CD project as a whole. Some lessons can be learned from the lean design approach,

however. For example, building an initial skeleton system and then expanding it with input from stakeholders is an appropriate procedure that also goes hand in hand with the insights from Software Analytics discussed above. Essentially, our results show that some compromise between robustness and leanness is the best approach when designing a feedback system in CD.

Finally, some results concerning the technical implementation were also reached. Overall, the concept of a monitoring layer above the pipeline, inspired by ING NL, is a great visualization tool, and helps explain how a feedback system fits into CD as a whole. Also from ING NL and from Eiffel, the concept of events is also a great means of managing both data collection, storage and analysis. As these functionalities from the basis of a feedback system design, utilizing events may be a relevant way to achieve them in practice.

Overall, our results show that feedback design should be approached with stakeholder interests in mind first. A system that does not deliver useful information is unlikely to succeed. The field of Software Analytics can be used for inspiration, especially the lesson of establishing an early feedback loop between a system's users and designers. The concept of events, together with a monitoring layer serves as a great combination to achieve any feedback system in practice. Finally, a centralized, robust design is more aligned with the previous results of this thesis.

4.4.2 Design Example

In this subsection an initial applicability of the results of RQ3 is explored. This is done as a first step into investigating how well the design results function in practice. To achieve this, a feedback design example sketch has been created, with the intent of preserving relevance to the context of Verisure. The relevancy is preserved by incorporating the components and guidelines of the RQ3 results, that are deemed to fit well in a Verisure context. This also includes the specific interests, features and metrics which the interviewees communicated as important. In the following points we want to address our thoughts and motivations behind some of the components in the design example of figure 4.2. This is done to briefly refer the pieces of the feedback design back to the results of RQ3.

- The design example aims to support stakeholder interest. This is done by having the interests in mind, both in the beginning to extract the right data from the pipeline, but also to filter and categorize the information that is provided to the suite of feedback features and services.
  - This design follows the guidelines about setting and communicating clear goals for the feedback design to achieve. The stakeholder interests act as the fundamental needs to support in this example.
- The RQ3 results state that the agile relationship between researcher and practitioner is an essential practice to ensure that proper solutions to the right problems are developed. The proper solution includes both the appropriate data source, but also the practitioner's preferred way of receiving the feedback. This researcher-practitioner relationship is stripped in this example due to limitations in time and resources. The RQ3 results from subsection 4.3.3 serve as the foundation to the features and services in the example.

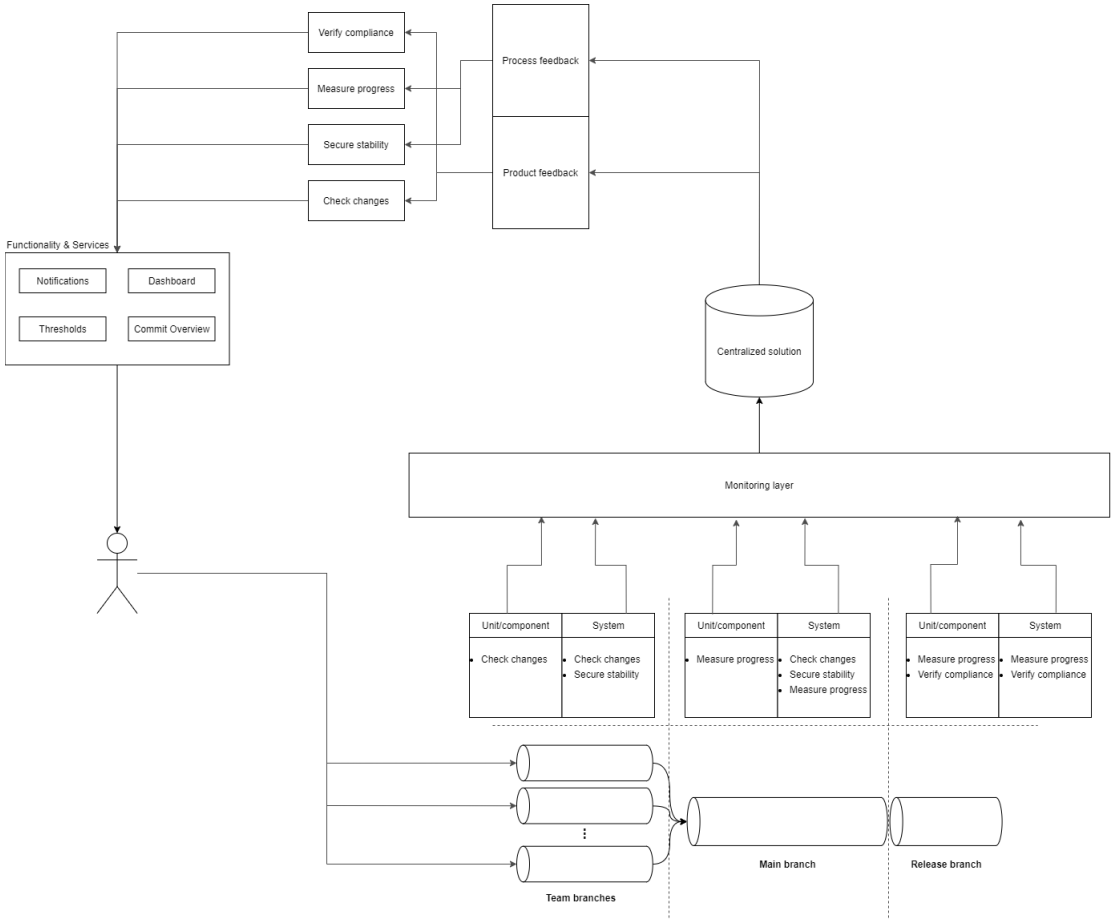

Figure 4.2: An example feedback design solution for Verisure, primarily based on the results of RQ3.

- Furthermore, the recommended architectural guidelines are taken into account, by incorporating a monitoring layer for data collecting, and a centralized solution of data storing.

Chapter 5
Discussion & Related Work

This chapter's aim is to discuss and evaluate the results of the thesis work. This is necessary in order to understand not only how the results relate to the objectives of the thesis itself, but also to discuss external related work, and various threats to validity. This is done by discussion about the chosen methodology, and how it could be done differently in upcoming projects. Furthermore threats to validity are presented and discussed, followed by discussion about if the RQs were considered justly answered, and the generalizability of said answers. Finally, related and future work is presented.

5.1 Methodology Discussion

In this section we want to present and discuss what parts of the methodology we think turned out to have been chosen wisely, and what parts that turned out to not work perfectly. Also, we aim to motivate what we would like to do differently on our next project, and why we believe that is.

To fit the designated time for interviews during the problem analysis phase, we decided to leave the number of interviews at 7. In the context of the thesis, and the research questions we aimed to answer with the collected interview data, 7 was deemed sufficient to provide us with an adequate understanding about Verisure's problems and insufficiencies regarding feedback systems and practices. Hence, 7 interviews was considered a good choice, as we achieved the goal of the interviews within the appointed time frame.

However during the interviews, new interesting stakeholder roles and interview candidate names were brought up and suggested as potential interview objects. As the interview trajectory slightly adjusted with each conducted interview, the relevancy of the newly discovered roles and candidates were potentially higher than the ones already booked. Consequently, there were some potentially interesting interviews that never took place as a result of the time constraint of the thesis work, that could have provided us with significant and novel insights. Considering this limitation in the current method, there are things we would have

done differently. A possible alteration to the method would be to book only half the available interview sessions in advance, and then assess potential additions of interview candidates before booking the other half. This would have allowed us to adjust the set of interviewees according to what roles we deemed relevant and valuable, after having some prior interview experience, and a better idea of what we wanted to gain from the interviews. However, it is yet unsure if that additional booking time would have fit the designated time frame, as the bookings had to be done almost a week in advance.

Changes could have also been made to the original selection of interviewees, as the managerial role felt slightly overrepresented, and some candidates seemed to be a little bit detached from the topics of discussion, and could in some cases be considered secondary sources. If given the opportunity to redo the interviews, perhaps a more even distribution between CD related and dependent roles could have been requested. More importantly, it would have been interesting to include the views and opinions of test-related roles for the problem analysis as well – a set of roles which was completely excluded and deprioritized in the case of the thesis. The reason for the exclusion of test-related roles was primarily due to the lack of them in the supposedly most CD mature teams at the company. In retrospect, we realize that this was not a valid enough motivation, as the level of CD maturity between the teams at Verisure had little to no impact on the interview results. The interview results instead appeared to be affected by the sole individual's attitude towards CD. Consequently, we believe that there definitely could have been some interesting opinions and perspectives from test-related roles, which could have either expanded or altered our current results associated with the testing domain.

For the design phase, the choice of performing free-form interviews was made to achieve a more exploratory approach. This objective turned out to be successful, and resulted in four very different interview sessions, all with their unique value to the design process. On the other hand, four very different interview sessions also meant four very disconnected interview sessions. Without any strong link between the interview topics, it is difficult to measure the progression of the work done. As the interviews were done in an iterative manner, with changes done to the design process in between, it would have been interesting to see how those changes were reflected in the interview results; something that was not possible due to the lack of recurring questions or thoughts. By choosing semi-structured interviews, the desired structure could be achieved, while still preserving the exploratory nature of the free-form interviews. The free-form interviews did work as intended though, and doubled-down on the variety and diversity of the topics discussed. However, the value gained by utilizing this interview form was saturated relatively quickly during the course of an interview. Topics of discussion could easily become irrelevant as there was no explicit scope to stay within, and after roughly 20-30 minutes of an intensive free-form discussion, the meeting could be considered over, as the session lacked a predefined goal to reference to. With that said, yes – the free-form interviews did work as intended, and yes – they did provide value. However, we believe the sessions could have had more potential to leverage.

The interviews during the design phase were conducted with three members of the Continuous Delivery project team, with roles as configuration managers, build chain managers, and IT operations architects. Even though the roles hold some relevance to the objectives of the design phase, it could be interesting to involve more practitioner-related roles, to include their perspectives when approaching feedback design.

5.2 Validation

This section discusses any threats to validity, and is based on the previous discussions. One large threat is the fact that no actual CD pipelines were studied directly, and no experiments were performed in an actual CD context. However, conclusions and results concerning our research questions can still be reached. For example, the value of feedback can still be understood – but some factors do impact validity.

The greatest threat to the validity of the thesis results, is the fact that no actual complete CD environments were studied in person. The consequences of this fact are difficult to gauge, but not all research questions are equally bound to the existence of a fully operational CD system. Also, the problems initially perceived, and therefore actual formulation of the research questions, take the situation of Verisure into account. The process of investigating RQ1 is not directly threatened by the lack of a true CD environment, as the value of feedback was explored by studying the negative effects of current poor feedback practices. It is reasonable to assume that these problems are only likely to compound on each other if the velocity of the studied pipelines were to be increased, as they will when CD is reached. Also, RQ1 has a foundation of academic sources, referencing complete CD systems. Therefore, our results are still valuable. RQ2 is more academic in nature, and therefore also not directly affected. The results of RQ3 are also thoroughly grounded in actual CD environments studied in literature. The data collection of the design phase served as an opportunity to gather more data specific to Verisure's needs, and do not impact the validity of the overall results other than strengthening them. However, It would be very interesting to compare all the conclusions of this thesis to ones originating from an actual CD pipeline.

Another threat is the fact that none of the conclusions or sub-conclusions were tested using experiments. Most interactions with Verisure were used for data collection and assistance, and for discussions of our results. Given the exploratory nature of the problems and research questions, the need for practical experiments is not vital. Performing them could be used to strengthen the validity of our results, however. The exploratory essence of the research questions is in itself a threat to the validity of the thesis. By formulating more strict and concrete questions, and investigating best practices for example, would lead to different results.

Finally, the overall scope of the covered problems is a factor to consider. Together, the research questions cover a plethora of topics, each containing a thesis full of data in themselves. The breadth of our thesis could have been limited slightly, to properly explore a subset of problems in a more complete manner.

In defence of the results, a brief validation session was performed with industry experts in the field of CI/CD and Configuration management. During this discussion, preliminary results for RQ1 and RQ3 were presented and evaluated. Overall, the conclusions of RQ1 and RQ3 were supported. Special attention was given to the importance of feedback to high-velocity environments, and the use of a centralized solution to support feedback delivery. The utilization of a monitoring layer and events was also validated as both realistic and helpful.

5.3 General Discussion

This section includes a general discussion of the results, in connection with the original problems and research questions. Specifically, the results of the broader RQ1 will be compared to the more specific RQ2 and RQ3. To what extent do we feel that we have reached our goals? RQ1 definitely, as it is more about exploration. RQ2 is slightly inconclusive, as we did not find evidence that one specific factor was more useful than the others, but still interesting. RQ2 is heavily influenced by the context of Verisure. Are our findings applicable in other cases, too?

The first point of discussion is the extent to which we believe we have reached the goals of the thesis. In general, we believe that the majority of the goals were reached, and that most of the initially perceived problems were addressed, or at least, touched on. When it comes to RQ1 and the originating problem, the goal of the thesis was to expand our knowledge and to satisfy our curiosity concerning the value of feedback. This was definitely achieved, and both the results and the method of reaching them in the context of Verisure are difficult to invalidate. As RQ1 was the broadest of the questions, the entirety of the thesis contributed to our understanding of feedback in CD. In regards to RQ2, results were useful, but slightly inconclusive. A number of factors were presented, but the chosen question did not allow for direct comparisons between them. Perhaps more insights could have been gained if a valuation of feedback factors was performed. As to RQ3, the results of the research question definitely reached the goals. However, RQ3 was intentionally limited to an initial exploration of the subject of feedback in practice. If taken one step forward would be to continue on the path of RQ3 and actually design and trial a feedback system in a CD environment. This topic is further elaborated on in the Future Work section.

The second and final topic of discussion is the general applicability of the results. Overall, the results are clearly generalizable to contexts similar to Verisure. On a broader level, the results still apply, perhaps more than might be expected from a thesis performed on only one company. Due to the fact that the company did not have a complete CD pipeline to study, the research and subsequent results were primarily based on literature of companies of similar size and characteristics as Verisure. This makes the results very applicable to companies in similar situations, especially the results of RQ1a. RQ1 as a whole is likely true in other contexts, because the value of feedback is probably the same in other companies of the same size. RQ2 is also generalizable, but other contexts might uncover other, more relevant factors. Specifically, the distinction between product and process feedback is likely to be found in other contexts.

Finally, the results of RQ3 are of varied applicability. The results concerning stakeholder interests and software analytics are very generalizable, as they are sourced in academic literature and likely apply to other contexts. When it comes to the results that originated from discussions with Verisure, general applicability is very difficult to estimate. Other companies might have other experiences and desires that contradict our conclusions. For example, another company might find a distributed solution to work better.

5.4 Related Work

The overall goal of this section is to describe and compare research related to the one of the thesis, and to present where the contributions of the research fits within the domain. Furthermore, it is described how the thesis work differentiates itself from past work, and points out the limitations and essential topics that the related work does not cover. This is done by presenting four related works within the context, summarizing and evaluating.

5.4.1 Test Activities in the Continuous Integration and Delivery Pipeline

Mårtensson et al. identified what stakeholder interests exist in the continuous integration and delivery pipeline, and how each interest can be supported by test activities [9]. We found this paper to be relevant to our work, as we also wanted to map the different interests and needs revolving the CD pipeline, and how a feedback system should be designed to best support the identified interests and needs.

Summary

As the need for more frequent and more reliable code changes increase in software development, so does the number of companies that adopt concepts and practices such as continuous integration and continuous delivery. However, scaling such practices to fit the size of the software system, and cover the needs of all possible roles within that system is, according to literature, not an obvious task. According to previous findings of the authors of the paper, the majority of the issues that come with scaling continuous practices are related to the domain of test activities. In order to get a better understanding of issues of such nature, the authors conducted a workshop regarding test activities, where the participants conveyed an interest in knowing what test activities that best serve the interests of the stakeholders within/around the continuous integration and delivery pipeline. Consequently, the research question of the paper is the following: How can the continuous integration and delivery pipeline be designed in order to support all existing stakeholder interests?

In the paper, the authors have contributed with the Test Activity Stakeholders (TAS) model – a model which shows how it is possible to design the continuous integration and delivery pipeline to include test activities that satisfy all of the different stakeholder interests in the organization. The authors have helped narrow down the different needs expressed by stakeholders throughout 25 interviews, into four comprehensive categories: “Check changes,” “Secure stability,” “Measure progress,” and “Verify compliance.” For each of these interests, different combinations of three test activity properties have been formed, which then have been positioned at different stages of the pipeline, in order to best satisfy each interest. Furthermore, researchers and practitioners might find great value in the interview results alone, as they provide information about how the case study companies meet different needs through different types of test activities.

The authors’ contributions are based on a total of 44 interviews, distributed between three interview rounds, all with their unique objective. The interviewees were participants from four different case study companies, in different industries, but who all develop large-

scale software-intensive embedded systems. The first interview round helped identify the different stakeholders and stakeholder interests. The second round was then used to identify the test activities that best best support each stakeholder interest. The third and final set of interviews was used to validate the TAS model. Furthermore, their contributions are supported by their previous work within the domain, where some of which have served as the motivation for pursuing this specific paper.

The authors conclude, backed by the contribution validation, that the contributed TAS model is valuable to practitioners of continuous practices. The TAS model demonstrates how continuous integration and delivery pipelines can be designed in order to best satisfy all of the different stakeholder interests in the organization.

Comparison and Discussion

First of all, we consider the research work of Mårtensson et al. to be extremely well done, making us deem the claims of the study as valid. The care and delicacy taken are clearly reflected throughout each step of the research method, where an emphasis on solidifying and substantiating claims have been made. For example, a literature review was conducted following the guidelines from Kitchenham, with the purpose of understanding what previously published literature reveal about the topic. This was done on 140 papers, all reviewed by the same researcher. Next the results of the literature review was reviewed by two other researchers, to secure quality and correctness. The same care permeated the interviews as well. Background material and the interview questions were sent out to all interviewees at least one day before the interview, to allow time for reflection. Each response during the interview was read back to the interviewee to ensure accuracy, and a validation step on the analysis of the interview results was conducted afterwards.

Both the research problem and contributions are considered to be of significance. The research problem originates from a need that Mårtensson et al. identified from their previous research, and further investigated it by conducting a workshop on that matter, with participants from 10 companies. The outcome of the workshop, which was turned into the research questions of the paper, resonates well with a problem we encountered during our thesis work. Our work also required us to identify existing stakeholders in CD at our case company, and attempt to ensure that all their main needs and interests were supported by our proposed solutions. We thus found the contributions from Mårtensson et al. to not only be interesting when answering what type of test activity that best serves each existing stakeholder interest, but not to mention how the test activity data can be transformed into valuable and useful information, and thus act as effective stakeholder feedback. Knowing what information to provide is great, but without effectively presenting that information to the right recipient, and thus not helping the recipient to respond with the appropriate next action, the value of collecting that information does not reach its fullest potential. By studying a day in the life of each stakeholder, identify what is relevant to them, and what information they prioritize, one can learn how to filter through the generated test activity, and how, when, and where to show that information, to further support the stakeholder in his or her work.

5.4.2 Software Analytics in Continuous Delivery: A Case Study on Success Factors

Huijgens et al. determined a total of 36 factors that either help or hinder the implementation of software analytics projects [5]. Since Verisure themselves currently are in the early stages of a transition towards Continuous Delivery, we deemed this work to be extremely relevant, as it gained us access to the experiences acquired by companies implementing what Verisure are currently attempting to implement. Having a list of recommendations and safety measures to utilize, could help Verisure realize what potential pitfalls to avoid, and what well-functioning aspects to double down on. This in turn could result in Verisure saving a lot of invaluable time and resources.

Summary

Research activities of financial organizations have traditionally leaned towards financial-, risk- and economic-oriented aspects. However, as technology advances, and its use cases and applications are discovered in more and more fields, new opportunities begin to arise where they previously did not exist. The same applies to the field of finance, where more and more research is done regarding technological issues, one of which is analytics. The authors of this paper believe that the focus on analytics will grow, and especially put emphasis on *software analytics* – the practice where analysis, software data, and systematic reasoning is used to help software engineers and managers make better decisions. Furthermore, the authors mean that despite there being plenty of research regarding the subject, how to implement it in practice, and especially within a continuous delivery setting, is yet to be explored – something that the authors aim to accomplish in this paper. The case company of this paper is *ING* – a Netherlands-based bank, which has in recent years implemented software analytics to a large number of teams operating within the context of continuous delivery.

With this paper, the authors wish to help future implementations of software analytics in continuous delivery settings, by understanding what have either helped or hindered the implementation of software analytics at the case company. There were a total of 36 factors identified, of which 16 helped, and 20 hindered, such a project.

The contributions of this paper are primarily based on 15 semi-structured interviews that the authors conducted with stakeholders of various roles within the case company. The interviews focused on the stakeholders' experience with the software analytics project, and covered the following five topics: goals of the analytics project; getting data; analyzing data; visualization; and collaboration with researchers. The interviewees communicated their opinions on statements within each topic by providing a score on the Likert-scale, and by answering open-ended follow-up questions to elaborate on the score of choice. The interviews were then coded, and translated into a total of 36 factors that were perceived to either help or hinder such a project. Furthermore, to become familiar with the architecture and features of the case company's software analytics project itself, the authors also studied the system and its artifacts within.

The authors conclude that even though the findings of the study were strictly related to stakeholders' experience within the case company's software analytics project, and that the outcome might simply not be generalizable to other environments, there are still some results that apply to the implementation of software analytics within a continuous delivery setting:

define and communicate the aims of the software analytics project upfront; standardize data at an early stage of the project; build efficient visualization, that are actionable and accessible; and utilize an empirical approach when starting the project.

Comparison and Discussion

The research problems of the paper are considered to be significant, and explore areas within software analytics which are yet to be comprehensively studied. The choice to delve into the less researched area of software analytics implementation in practice has been long awaited, and we believe that the findings – no matter how limited to the context of the case company – will benefit many when starting their own software analytics project. The authors even took it one step further, and studied software analytics in the context of the most state-of-the-art software delivery practices. This helps close an evidently wide gap in literature, and allow companies attempting to implement such a project to learn from the experiences of others in peer reviewed and acknowledged resources. Considering this, the research problems are believed to be significant. One could argue that the case study is not sufficiently extensive, as the contributions are based on the experiences of a single company's project. This is evidently a threat to validity, and the authors have treated the case with care in the paper. Regardless, there are still 3-4 take-aways which the authors believe could be applicable in any context, and – what we believe is even more important – the authors have paved the way for new research to be based on their work. This might accelerate the pace in which research on software analytics implementation in practice grows.

The paper aids practitioners initiating a software analytics project to realize what potential pitfalls to avoid, and what well-functioning aspects to double down on. Just as in the case of our master thesis work, the case study has been done in a continuous delivery setting. However, we have only been able to implement and study such a software analytics system in theory, as the case company of our thesis work currently does not have a continuous delivery pipeline in place. We therefore rely on the experiences of others in order to learn what works and what does not. Therefore, getting to digest work such as the one of the authors is critical to our success, and take-aways such as setting common, transparent goals upfront, and how to effectively visualize and deliver information to stakeholders, become invaluable to our own design solutions.

However, we find our study to have a relatively bigger focus on developers, and a single stakeholder's ability to become aware of how well he or she performs within the continuous delivery pipeline. This type of feedback – the feedback that is based on data points of a single event – is what we call *product feedback*, and is an area which does not appear to be the priority of the software analytics project of the case company of the paper. Instead, focus is on what we call *process feedback* – feedback regarding many events over time. Consequently, the software analytics project of this paper seems to primarily support stakeholder needs at a team or manager level, and has left us room to build upon the authors' work to explore how the software analytics project experience might differ with more consideration towards product feedback. In order to reach the state where this comparison can be done, our case company first has to finish the implementation of their continuous delivery pipeline, adopt the design solution of our thesis report, and then invite us for a contemplative revisit.

5.4.3 Continuous Testing and Solutions for Testing Problems in Continuous Delivery: A Systematic Literature Review

This subsection aims to summarize and evaluate an academic research paper related to the field of Continuous Delivery. In the paper, the authors Mascheroni and Irrazábal present a systematic literature review of the term *Continuous Testing* and explore how the term has evolved in literature [10]. They also discuss academias opinion on what testing stages exist within continuous software development, as well as the extent to which any related problems and solutions are documented. The other goal of this subsection is to explore the relevance of the paper to this thesis.

Summary

Mascheroni and Irrazábal have performed a systematic literature review exploring the concepts *Continuous Testing* and CD testing practices in general have evolved in academic literature. The authors claim that the field of Continuous Delivery and other related practices are on the forefront of modern software development in an industry context, but are not thoroughly explored by academics yet. Specifically, they identify the lack of clearly established best practices, together with lacking specifics regarding the importance of CD subcomponents and how to implement them. To mitigate this, the authors explore four research questions. Two are related to the term Continuous Testing – what definitions and open issues exist – and two relate to testing stages in CD and what problems and solutions are discussed.

To answer these questions, the authors applied a thorough and systematic approach to analyze existing literature. Articles from six sources such as *ACM Digital Library* using search terms related to continuous practices. After narrowing down 655 articles to 56 using inclusion and exclusion criteria, data related to each research question was extracted from each article. Finally, the extracted data was analyzed using a number of different techniques, and summarized.

To understand how the answers they were searching for may have changed over time, the authors applied a temporal view of the papers. Their results supported their initial sentiment – Continuous Delivery and its related fields are more prevalent in the recent five years. A majority of the articles were of the category *solution proposal*.

In regards to Continuous Testing, the authors describe a definition of the term that converges with time: “CT is the process of running any type of automated test case as quickly as possible in order to provide rapid feedback to the developer and detecting critical issues before going to production.” They discuss how this definition has changed with time to expand more and more types of automatic testing, and ascribe that to the evolution of technological possibilities. They also document a number of open issues with CT, notably *flaky tests* and *continuous monitoring*.

The results pertaining to the other two research questions thoroughly document and discuss six different types of testing stages prevalent in CD contexts, from *peer review* to *exploratory manual testing*. Mascheroni and Irrazábal briefly discuss the role of each testing stage within a CD pipeline, and show how some types of testing have emerged over time as a consequence of CD establishing itself. The authors also present the most common and

challenging testing related issues in CD, and group them by similarity. These issues were things like *time-consuming testing* and *automated flaky and ambiguous tests*. For each test issue, the authors discuss to what extent solutions are prevalent in the studied literature.

Comparison and Discussion

The validity of the results is clear – the authors have approached their problem from multiple directions and done their work thoroughly. It is also apparent that their research is of some significance, as *Continuous Testing* is most likely going to be an integral part of the domain of continuous software practices.

There exist connections between Mascheroni and Irrazábals paper and our thesis. This subsection aims to explore to what extent their research is relevant to the problems and conclusions of our thesis. In general, both documents investigate related problem domains. However, Mascheroni and Irrazábals paper focuses on testing within CD instead of feedback. There are dependencies between testing and feedback, as this document has discussed, but the subjects do not entirely overlap.

One interesting comparison that can be made between the results of Mascheroni and Irrazábal and our paper concerns the definition of Continuous Testing. According to the authors, that definition has become more and more well-formulated with time. Discussing the applicability of the term Continuous Feedback is not within the scope of this thesis, as it has not yet been accepted as a general term. However, given the similarities between the two domains, it is likely that Continuous Feedback will become more and more used and well-defined, just like Continuous Testing. Another reasonable inference is the expansion of the term Continuous Feedback to include more and more automation. This was observed by Mascheroni and Irrazábal in regards to testing, and has already occured with feedback in CD.

5.4.4 Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development

The purpose of this subsection is to present and discuss a research paper related to the questions asked by this thesis. The paper in question, *Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development*, describes the requirements of a feedback system [13]. The authors also show how those requirements are used in practice. As the scope of RQ3 of our thesis covers very similar questions, the paper is highly relevant to the thesis.

Summary

In their paper titled Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development, authors Selby, et al., present a feedback framework that utilizes metrics and other data from development environments to support the creation of high-quality software. The purpose of the paper is to motivate what advantages such a framework provides, and also to describe how their solution uses empirically guided inputs from users to optimize a number of processes. Finally, the authors describe how their framework has been

put into practice through an actual feedback and analysis system prototype called *Amadeus*, giving some validation to their hypotheses.

The purpose of Amadeus is to fulfill a number of high-level requirements centered around what they call empirically based software development and evaluation. These requirements originate from a number of observations taken from software development processes, insights that shine light on the value of information transfer. Other than identifying feedback as valuable, the authors also claim that complete code comprehension is often low in teams, leading to an uneven distribution of efforts during problem solving. They reason that if done empirically, problem-prone parts of the code base could be easily identified and addressed. To achieve this, they recommend integrating measurements and processes in an active manner. As such, the requirements of their framework, and thereby their system Amadeus, focus on the following aspects: enabling measurement that leads to empirical feedback, supporting empirically based maintenance processes, and allowing for an expandable number of analysis techniques.

Other than fulfilling these requirements, Amadeus also has a number of key characteristics that make it applicable to large-scale software environments. Overall, any feedback system should be scalable to support an expanding code architecture and scope. It also needs to be flexible, to support varying needs of users. Finally, it also needs to incorporate time as a factor into its design – when did something occur within the system? This can be achieved by utilizing event-based programming. The authors also describe two vital analysis techniques of the system. The first, classification analysis, uses the history of a software project to predict and classify components that are likely to cause errors in the future – so called high-payoff components. The second technique, interconnectivity analysis, enables architecture comprehension of systems, by exploring the traceability of dependencies between subcomponents. The authors argue that these two techniques are highly useful, but also note that the system should support the addition of other techniques.

In conclusion, the authors argue that feedback systems play a vital role in supporting software creation, and posit that their framework and accompanying implementation fulfill some of those needs. They also claim that their results imply that even a basic feedback and analysis system is better than no system.

Comparison and Discussion

The research performed by the authors is significant, but it is difficult to verify the validity of the results. In part this is due to the time gap – software development has moved on and the actual technical implementation is impossible to experiment with. The authors do not describe their system working in the context of any specific software product, making it difficult to know whether their design proved useful or not.

To analyze the relevancy of this paper to our thesis, the context of their research must be compared to our context. The first aspect to consider is that the year of publication – 1991 – makes this paper considerably dated in the context of software research. Nonetheless, many of the conclusions presented by Selby, et al. are mirrored by our own results. This is especially true when considering that the authors repeatedly press on the value of feedback. Likewise, there are similarities within the described characteristics of a feedback system. Both the results of the paper and our thesis conclude that flexibility and scalability are important. Selby, et al. also implemented an event-based approach similar to the one discussed in the

design phase of this thesis.

Another important distinction between our thesis and Selby, et al. is the fact that their research was not done in the context of Continuous Delivery. This directly impacts the relevancy of the paper, as this thesis sits firmly within the ramification of CD – high software velocity. How this context impacts the comparability of their results to ours is difficult to estimate. One potential consequence is that the primary purpose of their system seems to be to localize and predict hidden faults and error sources. To Selby, et al., basic feedback information is only secondary, probably due to the relative slowness of the software of their times. The fact that our thesis focuses on CD environments could explain the fact that our results value feedback from basic test results just as highly as fault identification.

Overall, the fact that Selby, et al. reached similar conclusions to us while studying different software systems in a by-gone era suggests that feedback has always played an important role within software development. However, drawing stronger comparisons is difficult, due to the difference in context between our domains.

5.5 Future Work

During our work we discovered many interesting things that we did not have time and resources to follow up on – or that were not completely within the scope. In this subsection we would like to present some of the derived things as future work. The following suggestions help provide an understanding of what we would deem interesting to continue on and explore further. Furthermore, they also function as a source of inspiration for anyone interested in extending our research, or make our results stronger and more general.

It would be interesting to take the results of RQ2 and RQ3, and explore how well they apply in the case of a company that has come further on their transition towards Continuous Delivery, and perhaps already have established both a CD pipeline, and an equivalent to the Feedback System of this thesis. In the case of RQ2, it could be interesting to see how well the primarily theoretical results of this thesis compare what factors actual CD practitioners consider relevant when categorizing feedback in CD. If no distinct factors have been mindfully established at the case company, perhaps it is interesting to see if the contributions of RQ2 bring any new value, or if they become redundant. The same applies to RQ2a, where there is still a considerable amount of research left to be made. In this thesis, product and process feedback were defined and introduced in an attempt to bring order to the otherwise perceivably disorganized and chaotic topic of feedback types. What has been taken here is no more than an initial superficial step into the research, which leaves many options for what direction to continue on. For RQ3 it could be interesting to trial the resulting feedback design approaches when implementing a Feedback System at a team working in CD. How well do they work in practice? What approaches are essential for a successful project? What approaches are less important, or optional? In other words, use the results of RQ3 and trial their applicability in practice.

The results of RQ3 currently lack any information about how the different approaches relate to or compare against each other. A future work could be to attempt to examine the different approaches, and prioritize them based on their contribution value. How could the currently unordered results of RQ3 be used when constructing a roadmap, a set of guidelines, or a step-by-step instructions guide? Another thing that could be interesting to take further

is primarily based on the findings of the work by Mårtensson et al. [9]. It may be of interest to investigate how the information utilized to support different stakeholder needs, is best provided and presented to each respective stakeholder. By exploring this, we not only know what the interests and needs of the stakeholders are, where the necessary data can be collected from, but also how to best convert that data into proper, timely, and effective feedback, which can best support the stakeholder's decision making.

Chapter 6
Conclusion

In summary, this thesis had the overarching goal of exploring the value of feedback in Continuous Delivery. To achieve this goal, the questions of feedback categories and approaches in practice were also investigated. To collect both academic and industry-based data, interviews and literature studies were utilized. To collect academic data relevant to the questions, two literature studies were performed. Two sets of interviews at a company transitioning to CD were also carried out, to base the research in an industry context.

Our results show that feedback plays a key role in supporting high-velocity software development systems, by giving individual teams vital knowledge about their domain. This knowledge helps team members make correct decisions independently and in a timely fashion, maintaining the high artifact throughput central to CD without sacrificing quality. Feedback also gives insight into the domains of cooperating teams – the importance of which cannot be understated in a mid-to-large size company. If code from two teams is eventually integrated, sharing vital information across team borders ensures a continuous deployment process, from change commit to customer delivery.

The results also illuminated four factors that categorize feedback in CD – stakeholder interests, urgency, product and process. Each factor has its distinct purpose, and all can be used when understanding and implementing feedback. Stakeholder interests categorize feedback after the roles present within a team. This ensures no person's needs are overlooked or undervalued. Urgency is useful as a factor when studying how fast feedback needs to be. Some information needs to arrive as fast as possible to maintain continuous development. Other information is not as time sensitive, but no less valuable. Finally, process and product allows feedback to be categorized after the size of the change, in relation to the contribution's destination. Our results show that some overlap exists between these factors, as urgency and stakeholder interests connect to product and process feedback.

When approaching the design of feedback in practice, a centralized and automatic feedback system is the best approach for a mid-to-large sized company. The complexity of this system scales with development size and speed, but even a rudimentary system is better than each team developing their own ad hoc feedback solutions. Complete ad hoc solutions often

make information comprehension across teams very difficult, and are difficult to document and scale. In practice, feedback can be implemented using a monitoring layer above the development pipeline, and can utilize events instead of raw information for storage and analysis. Finally, feedback design should be approached with stakeholder interests in mind, and should focus on usability and practicality to remain relevant in the long term.

References

[1] Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54, 2015.

[2] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016.

[3] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[4] Alena Hramyka and Martin Winqvist. Traceability in continuous integration pipelines using the eiffel protocol. 2019.

[5] Hennie Huijgens, Davide Spadini, Dick Stevens, Niels Visser, and Arie Van Deursen. Software analytics in continuous delivery: a case study on success factors. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2018.

[6] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[7] Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, and Barbara Paech. How do practitioners capture and utilize user feedback during continuous software engineering? In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 153–164. IEEE, 2019.

[8] Oliver Krancher, Pascal Luther, and Marc Jost. Key affordances of platform-as-a-service: self-organization and continuous feedback. *Journal of Management Information Systems*, 35(3):776–812, 2018.

[9] Torvald Mårtensson, Daniel Ståhl, and Jan Bosch. Test activities in the continuous integration and delivery pipeline. *Journal of Software: Evolution and Process*, 31(4):e2153, 2019.

[10] Maximiliano A Mascheroni and Emanuel Irrazábal. Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review. *Computación y Sistemas*, 22(3):1009–1038, 2018.

[11] Victoria Neufeldt et al. *Webster's new world dictionary*. Simon and Schuster, 2002.

[12] R Owen Rogers. Scaling continuous integration. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 68–76. Springer, 2004.

[13] Richard W Selby, Adam A Porter, Doug C Schmidt, and Jim Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of the 13th international conference on Software engineering*, pages 288–298, 1991.

[14] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.

[15] Mojtaba Shahin, Mansooreh Zahedi, Muhammad Ali Babar, and Liming Zhu. An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering*, 24(3):1061–1108, 2019.

[16] Emad Shihab, Ahmed E Hassan, Bram Adams, and Zhen Ming Jiang. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.

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

[18] Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta, and Andy Zaidman. Continuous delivery practices in a large financial organization. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 519–528. IEEE, 2016.

[19] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. Software analytics in practice. *IEEE software*, 30(5):30–37, 2013.

Appendices

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-01-29

EXAMENSARBETE Establishing Feedback in Continuous Delivery – Benefits and Approaches
STUDENTER Emanuel Eriksson, Keiwan Mosaddegh
HANDLEDARE Lars Bendix (LTH)
EXAMINATOR Emelie Engström (LTH)

Closing the Feedback Loop in Continuous Delivery

POPULAR SCIENCE PAPER **Emanuel Eriksson, Keiwan Mosaddegh**

High-velocity software systems like Continuous Delivery put strict requirements on the surrounding development infrastructure. This thesis work explores the purpose and value of feedback within CD, focusing on the benefits of robust feedback delivery systems and approaches to practical implementations.

The goal of many modern software development paradigms is to reduce the length of development cycles until terms like ”cycles” or ”iterations” lose their meaning. Instead of planning and micro-managing the release of a product at a set monthly or weekly time, software systems like *Continuous Delivery* enable programmers to push their contributions directly to end users, essentially at any time. To support this, a robust technical infrastructure of tools and inter-connected systems allows for code to be built, tested and deployed automatically without sacrificing quality.

However, the relatively extreme speed of these systems leads to both many competitive advantages and greater risks. To mitigate these risks, practitioners often point to feedback from tools and people alike as an important part of the CD puzzle. When software velocity is high, continuous feedback can ensure that quality is maintained without sacrificing throughput. However, the precise impact and value of feedback in modern development paradigms like CI/CD is not thoroughly explored. Distinctions between different types of feedback, as well as descriptions of how a feedback loop is to be established, are incomplete in literature. Why exactly is feedback so valuable, and how should it be approached in practice?

Based on literature analysis and data gathered from a company transitioning to CD, we have attempted to categorize and evaluate the urgency and utility of different types of feedback within CD – specifically distinguishing between process and product feedback. We also explored the value of feedback by analyzing feedback-related problems at the company. To address a subset of these problems, a number of approaches to feedback design in practice were evaluated.

Overall, our results show that any generalizable feedback system, even a rudimentary one, is an extreme necessity to achieve sustainable Continuous Delivery. This is especially true when multiple teams cooperate, as one improvised ad hoc solution per team is likely to hinder comprehension across teams. In practice, this system should be centralized but tailorable after specific team needs.