Synthesizing Perceived Challenges in Continuous Delivery - A Systematic Literature Review

Ville Pulkkinen

MSc Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, October 17, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author

Ville Pulkkinen

Työn nimi — Arbetets titel — Title

Synthesizing Perceived Challenges in Continuous Delivery - A Systematic Literature Review

Oppiaine — Läroämne — Subject

Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc Thesis	October 17, 2016	91 pages + 36 appendix pages

Tiivistelmä — Referat — Abstract

Continuous delivery is an approach to software development which incorporates the practices, technologies and processes in order to achieve frequent delivery of valuable software to customers. Even though the continuous delivery approach has not existed very long yet, there has been quite a lot of a buzz around it and terms related to it (continuous deployment, deployment pipeline, and DevOps). Practices and benefits of the approach are presented in the literature, and organizations have been adopting it to a varying extent.

However, as easy as the advocates of continuous delivery make the adoption look like, there have been reported challenges along the way. In order to focus research on finding the causes and creating solutions to these challenges, we must first identify them. To address this, we conducted a systematic literature review in order to collect perceived challenges related to the adoption of continuous delivery practices in software development projects, and analyzed the findings in order to provide synthesized information about these challenges.

From among 13 publications 59 different challenges were identified which we categorized either as a social (procedural or organizational) or as a technical type of a challenge based on the evaluation of the findings. Among these challenges we found 14 more frequently occurring ones which also spanned across multiple software domains. We described these as common challenges. We also analyzed the reasons behind these challenges and identified five different themes (main reasons) that were immaturity, unsuitability, complexity, dependency, and security. We also analyzed how the software domain affected these reasons. Based on the observed mitigation strategies and research proposals, and our analysis, we proposed suggestions for future research directions.

This study can be used as a support for finding future research directions regarding the challenges in the area of adopting continuous delivery practices in software development projects.

ACM Computing Classification System (CCS):
Software and its engineering ~ Software creation and management
Software and its engineering ~ Software development techniques
Software and its engineering ~ Rapid application development
Social and professional topics ~ Management of computing and information systems

Avainsanat — Nyckelord — Keywords

Systematic Literature Review, Continuous Delivery, Continuous Deployment, DevOps, Challenges

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

1.1 Motivation

Continuous delivery is an approach to software development which incorporates the practices, technologies and processes in order to achieve frequent delivery of valuable software to customers. Even though the continuous delivery approach has not existed very long yet, there has been quite a lot of a buzz around it and terms related to it (continuous deployment, deployment pipeline, and DevOps). Practices and benefits of the approach are presented in the literature [HF10] [Fow13] [All], and organizations have been adopting it to a varying extent. However, as easy as the advocates of continuous delivery make the adoption look like, there have been reported challenges along the way [MAK$^+$14] [RHL$^+$16].

Challenges are usually encountered when an organization is trying to adopt continuous deployment and delivery approach i.e. when implementing the deployment pipeline (technical challenge) or is adopting continuous delivery practices in the organization (organizational challenge). To focus research on finding the causes and creating solutions to these challenges we must first identify them. We conducted a systematic literature review in order to collect perceived challenges related in adopting continuous delivery practices to software development projects, and analyzed the findings in order to provide synthesized information about these challenges.

Description of continuous delivery (or continuous deployment) practices and incorporated concepts are described in literature [HF10]. Many benefits of continuous delivery are presented [HF10] [Fow13] [RHL$^+$16] [All] but the challenges related to implementation are less studied. One reason might be that the authors and practitioners are more willing to report positive than negative results [RHL$^+$16]. Research of continuous delivery is still in its early stages [RHL$^+$16] and the data is quite dispersed. Even though the continuous delivery process and the benefits are presented to some extent, and probably also comprehended by adopting organizations, we feel that the community is missing a clear view about the challenges encountered when implementing continuous delivery systems and how to overcome these challenges. In this study we are going to address and synthesize these challenges encountered in the implementation of continuous delivery process and also explore the mitigation strategies used to overcome the challenges.

We found only two comprehensive systematic literature review studies related to continuous delivery of software where also challenges are studied [MAK$^+$14] [RHL$^+$16]. Rodriguez et al. [RHL$^+$16] made a systematic mapping study about continuous deployment and identified four different challenges regarding the transformation when moving towards continuous deployment. Mäntylä et al. made a composite paper of case study and semi-systematic literature review of releasing software more rapidly [MAK$^+$14]. They also found four different more frequently occurring issues relating to rapid releases and reflected those findings to the results from the case study. In this study, we are going to focus only on the challenges faced in continuous delivery and continuous deployment and deepen the knowledge about the challenges.

1.2 Research Questions

The goal of this study is to synthesize the challenges encountered in the adoption of continuous delivery approach in software development projects, and to find out if there are some common factors between these challenges. We also aim to provide some insights for the existing mitigation strategies and future research directions related to challenges in adopting continuous delivery. Hence, the main research questions (RQs) are the following:

RQ1 What are the perceived challenges of adopting continuous delivery practices in a software development project and why?

RQ2 How should the research on continuous delivery be directed according to these challenges?

To be more precise, in RQ1 we want to know what are the perceived challenges, which challenges are more common, which are more specific to certain software domain (considering the frequency of challenges, company sizes and software/business domain), and also what is causing these challenges. In RQ2 we try to seek new directions for future research related to the challenges by figuring out the quality of current studies and presenting

Table 1: Research questions

Id	**Research question**
RQ1	What are the perceived challenges of adopting continuous delivery practices in a software development project and why?
	RQ1.1 What are the perceived challenges?
	RQ1.2 Which challenges are more common?
	RQ1.3 Which challenges are more specific?
	RQ1.4 What are the reasons for these challenges?
RQ2	How should the research on continuous delivery be directed according to these challenges?
	RQ2.1 What is the quality status of current research related to these challenges?
	RQ2.2 What are the presented mitigation strategies used to overcome these challenges?
	RQ2.3 What are the proposed research directions related to these challenges?

proposed mitigation strategies with proposed research directions. Main research questions and sub questions are presented in Table 1.

To answer these questions we did a systematic literature review. With the systematic literature review we found a wide range of challenges which we categorized and analyzed in order to find the common ones. We created higher-order themes to find out the reasons behind these challenges. We also studied the quality of the papers related to the challenges met in adopting continuous delivery practices with the help of quality checklist. This study will aid researchers to better direct future research regarding the challenges met in the area of continuous delivery adoption.

1.3 Thesis Structure

In the first chapter we familiarize the reader with a brief introduction to the concept of continuous delivery, and also present the motivation, goals, and the research questions so the reader will get a good overview about why this study has been made. The second chapter is where we deepen the knowledge about the concept of continuous delivery. In the third chapter we will present research methods used in this study. It is the chapter for those who are interested about the methodological aspects of how the research has been made. The fourth chapter is where the results are presented before the analysis. In the fifth chapter we will analyze the challenges and the reasons behind the challenge to provide more understanding of them. The discussion chapter is the sixth one and consists of discussion of the results and potential threats to the validity of the study. This study will be summed up in the last chapter where contributions and future research proposals are presented.

2 Continuous Delivery

We will briefly go through the basic foundations of continuous delivery in order to help the reader understand the context from where we are seeking the challenges. Firstly, we present the origins of continuous delivery practice and after that, we will describe several terms that are related to continuous delivery (continuous deployment, continuous experimentation, deployment pipeline, and DevOps). Secondly, expected benefits and used practices will be presented. Finally, we will go through the challenges that are already observed in the previous studies. After this chapter the reader should be familiar with the main concept this study is considering (i.e. continuous delivery).

2.1 Origins of Continuous Delivery

Continuous delivery approach has evolved from *continuous integration (CI)* practice. Continuous integration is a practice that ensures the use of some *Agile* and *Extreme Programming (XP)* practices appreciated by practitioners. The goal of CI is to continuously integrate (preferably automatically to save human labor) all of the pieces of an application in order to form a working software. Working software is a software that is somehow verified, usually by testing, to be working as it should be and is deployable [DMG07]. The main function of continuous integration is to run tests in a centralized server, as soon as possible, so that developers will get feedback that the code they commit to the code base is functioning with the rest. With continuous integration large integration problems can be avoided. The software can be developed with *test-driven development (TDD)* approach where tests are made just before the actual features in small increments.

Beck wrote about continuous integration practice in his book *Extreme Programming Explained* already in the year 1999 [BA99]. At that time the idea of continuous integration was quite new and there was a lack of tools for such a development strategy. The author also mentioned that testing will be the most time consuming part of continuous integration and that a complete test suite should be run in few minutes. He also recommended the practice of continuous integration because it will help you to split your development tasks in smaller pieces and thus will help to reduce the integration problems and thereby reduce the risk of delivering the software.

The idea of continuously delivering and deploying software came from practitioners who embraced agile software development methodologies [HRN06] [DMG07] [Fit09] [HF10] [Fow13] but the idea of continuously delivering software (as fast as possible) is also promoted by the *lean software development* approach [CSA15]. Lean software development is an approach to software development that has certain principles (optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone, and keep getting better) to which everything else (practices, processes and philosophy) is based on [PC12]. However, the continuous delivery (or deployment) approach can be used regardless of the software development process or methodology. Continuous deployment is said to be *"a culmination of practices and steps which enable us to release working software any time, any place, with as little effort as possible."* [DMG07]. It can also be seen as an extension to continuous integration practice [All].

Because there are slightly different interpretations about the terms continuous delivery and continuous deployment in the research studies and literature [FS14] [RHWP15], we will also define these terms to clarify how we understand them. One trend is to define continuous deployment as a practice of deploying code immediately to the production environment after some change [Fit09] [HF10] [Fow13] while another is that it means automation of the process of deployment to any environment [DMG07] [FS14]. Also some kind of hybrid interpretation exists as *"We define continuous deployment as a software engineering process where incremental software changes are automatically tested, and frequently deployed to production environments"* [RHWP15]. In any case, there is no big difference in the overall process of making changes to the source code and releasing the change to the actual users. Continuous delivery is mainly an enabler for continuous deployment [FS14]. To achieve continuous delivery you need *DevOps-culture* and *deployment pipeline* to be in place [Fow13]. That applies also for continuous deployment. Next we will go briefly through all these terms.

2.2 Terminology

In this section we define the key terms related to continuous delivery. These terms are somewhat overlapping in the literature. At the same time we also set the foundation to our systematic literature review search scope (especially search keywords).

2.2.1 Continuous Deployment

By *continuous deployment*, we mean an approach where all changes to the software that a developer commits to the *version control system (VCS)* are deployed automatically, without human interaction, to the production environment (to end-users) immediately after all the required tests are passed successfully [Fit09] [HF10] [Fow13]. This approach relies fully on the TDD approach. The testing phase includes unit, integration, system, and acceptance (functional and non-functional) testing.

2.2.2 Continuous Delivery

It is said that *continuous delivery* is *"a set of practices and principles to release software faster and more frequently"* [KA14]. Continuous delivery is a less radical approach, compared to continuous deployment, where new builds are deployed to the production environment if one chooses so (for example by a click of a button) after all of the required tests are passed successfully [Fow13]. With continuous delivery, it is possible to perform for example user assurance testing, exploratory testing [BF14] or other actions operated by humans before the build is deployed to the production environment [HF10] (which is impossible with our definition of continuous deployment). By definition, you are conducting continuous delivery when your software is deployable all the time, your priority number one is to keep it that way, you get feedback fast and automatically about the system, and you can perform "push of a button" deployments of any version, to any environment, at any time [Fow13].

2.2.3 Deployment Pipeline

To achieve continuous delivery or continuous deployment, the process must be relying on a fully automated *deployment pipeline* [Fow13]. A deployment pipeline automates the process where the latest source code is fetched, compiled, built, tested, and deployed to different environments which are also automatically provided and configured [HF10]. Earlier, when the concept of deployment pipeline was introduced, it was referred to as *deployment production line* [HRN06]. Also some other terms are used such as continuous integration pipeline, build pipeline, or living build [HF10]. The deployment pipeline consists of different stages which act as gatekeepers for the process of software delivery automation. These stages are for example commit stage, automated acceptance testing stage, manual testing stage, and release stage [HF10]. If a build is failing in some stage it will not proceed to the next stage. Deployment pipeline is also said, as being part of continuous delivery, to take continuous integration practice to its logical conclusion [HF10].

2.2.4 DevOps

DevOps is a way of doing things related to delivery which emphasizes a close and collaborative relationship between everyone involved in the delivery [Fow13]. In a study which was focusing on definition and perceived adoption impediments of DevOps, the authors defined DevOps to be as *"...a set of engineering process capabilities supported by certain cultural and technological enablers"* [SNP15]. As such, the definition is overlapping with the continuous delivery approach. Humble et al. note in their book "Continuous Delivery" that *"the DevOps movement is focused on the same goal we set out in this book: encouraging greater collaboration between everyone involved in software delivery in order to release valuable software faster and more reliably"* [HF10]. In the paper where DevOps definitions were studied, technological and cultural enablers and capabilities were presented [SNP15]. Technological enablers were automation of build, test, deployment, monitoring, recovery, infrastructure and configuration management. Cultural enablers were sharing, communication, constant experimentation, and learning. Capabilities were relating to collaborative and continuous approach to software development.

One good observation we found was that the term DevOps is a bit misleading because it is a combination of development and operations, but actually there are also other parties involved like testers and database administrators etc. [Fow13].

2.2.5 Continuous Experimentation

By *continuous experimentation* approach, we are referring to an approach of the software development that is heavily influenced by rapid experimentation of different variations or new features of the software [OAB12]. With the help of collected customer usage data from the experiments, developers can steer the direction of the software development project. Continuous deployment is one of the enablers for efficient utilization of customer usage data throughout development [OAB12]. However, before you can continuously deploy a software you need to have a continuous delivery system in place [Fow13]. Because of these pre-requisites, we decided that continuous experimentation is included in our scope considering the challenges in adoption of continuous delivery practices.

2.3 Benefits and Practices

In the Internet-article by Fowler [Fow13] which discusses about continuous delivery, the author states that continuous delivery will reduce the risk of deployment since the deployment is done more frequently and in smaller batches so there are fewer things that could go wrong and it is easier to fix if some problem appears. He also said that the process of software development is more credible when "done" means that it is actually deployed to production (or production like environment). The author also noted that the biggest risk to any effort of software development is that you have built something that is not useful. The earlier and more frequently you get the feedback from the actual users, the quicker you will find out if the software is valuable to your users and you are able to utilize this information to the benefit of development.

The Agile Alliance community claims that the main benefits arise as a result of reducing lead time by *"achieving earlier return on investment for* *each feature after it is developed, which reduces the need for large capital investments"* and by *"earlier feedback from users on each new feature as it is released to production, which affords techniques such as parallel (or A/B) testing to determine which of two possible implementations is preferred by users"* [All].

Also in the book "Continuous Delivery" the following benefits were presented for continuous delivery: it is empowering teams, it will reduce errors and lower stress, it enables deployment flexibility, and by practicing it multiple times before actual production deployment the process of delivering will become better [HF10]. Next we will briefly describe these benefits in more detail.

Continuous delivery is said to empower teams as the pull system enables efficient deployment of any version to any environment thus it is easier for example for testers to get the version they need to test, support personnel to get some specific version for reproducing a defect, and operations to deploy version that they want to the production. Also by automating all things in the delivery process (e.g. provisioning, configuration) continuous delivery is said to reduce errors because the possibility of introducing errors by doing things manually is eliminated. Instead of having release days on rare occasions when the deployment is done to production, continuous delivery encourages to deploy frequently which will lower the stress in releasing a software. The automation of the deployment process should also make it easier to create new instances of an application to completely new environments so the deployments will become more flexible. Also, the automation begins from the dedicated development environments that are provided to every developer so the same deployment process is used for providing these first steps in the development process. This ensures that the automation is practiced many times before the actual software is delivered to customers.

Perceived benefits were studied in a systematic mapping study about continuous deployment of software intensive product and services [RHL+16]. These benefits were shorter time-to-market, continuous feedback, improved release reliability, increased customer satisfaction, improved developer productivity, rapid innovation, and narrower test focus.

In a multiple case study of synthesizing continuous deployment practices, in which cases were mainly based on Internet sources, there were 11 different practices identified [RHWP15]. Identified practices were automated deployment, automated testing, code review, dark launching, end-user communication (efficient use of feedback), feature flags (feature toggles), intercommunication (sharing of development status), monitoring (especially production environment), repository use (use of VCS), shepherding changes (developer being responsible for a change throughout the deployment process), and staging (dogfooding or gradual rollout). Only a few of these were identified as common ones. Common ones were automated deployment, automated testing, and repository use. Even though these practices were identified from the case companies, the authors failed to understand why these practices help the case companies to achieve continuous deployment. These perceived practices also reflect the practices presented in the literature [HF10].

In the systematic mapping study of continuous delivery of software intensive products and services [RHL+16] the authors observed following 10 different factors that were recurring among the cases:

1. The approach was to release software fast and frequently so that organizations could release software on demand. Preference of releasing frequency was given to shorter-cycles or even continuously.
2. The product usually had a flexible architecture so that balancing between stability and speed was possible.
3. Typically products were continuously under a test so that quality was assured alongside fast releases.
4. Delivery pipeline was built so that the whole process (building, testing, deploying, and monitoring) of delivering software was automated.
5. Version control branching strategies and system configuration management were used in a way that enabled continuous delivery.
6. Customers were more involved in the development process by the means of providing usage data and feedback as early as possible to steer the development process direction.
7. Also for the same reason small experiments were done rapidly or even continuously.

8. Post-deployment activities were typical for the support of business and technical decision making.
9. Agile and lean approaches to software development were used and extended to better fit for continuous delivery.
10. Also organization was adjusted to better support continuous delivery and towards more experimental and transparent mindset.

2.4 Challenges

However, even though benefits and practices are presented in multiple sources, there are also reports of challenges along the way in transformation towards the continuous delivery approach.

Rodriguez et al. made a systematic mapping study about continuous deployment and identified some challenges regarding the transformation towards continuous deployment [RHL$^+$16]. Frequent challenges were the process of transformation towards continuous deployment, customers' unwillingness to receive continuous product updates, increased quality assurance (QA) efforts, and difficulties applying continuous deployment in the embedded domain [RHL$^+$16]. They also analyzed briefly in detail the challenges faced. They noted that challenges were less reported than benefits which may be because authors and practitioners might be more willing to report positive than negative results [RHL$^+$16].

Other challenges which appeared less frequently in the literature were lack of trust in software quality, difficulty in managing various configurations and run-time environments, a tension between the desire to deliver functionalities quickly and the need for reliable products, release planning and managing roadmaps in a fast-paced environment, and risks associated with gathering user feedback from limited population that might even mislead the development of the product [RHL$^+$16].

Mäntylä et al. made a composite paper of a case study and semi-systematic literature review of releasing software more rapidly [MAK$^+$14]. Even though the study itself does not state to be concentrating on continuous delivery, it is closely related to continuous delivery. In the systematic literature review search phase they used search words of "continuous release", “continuous delivery”, “continuous deployment” and some related terms and variations. In the semi-systematic literature review they found four different problem areas relating to rapid releases. They observed from the literature that:

1. there are problems with high reliability and lack of high test coverage,
2. it seems unlikely that automated testing can completely solve the challenges related to reliability or test coverage,
3. customers might get weary of yet another update, and
4. rapid releases may increase technical debt (or lead to staff burnout) as there is less time available for thinking things through [MAK+14].

Mäntylä et al. also reflected the findings from the literature to their case study and stated that their findings from the case study partially supported the findings from the literature. They noticed that there was a reduction in environments which to test (because of reducing legacy support in testing) but at the same time increased some in-depth testing of few important features which was considered as a great benefit. Also, there was not much of a decline in quality or reliability. Problems of customer adoption were mitigated by extended support release which did not have so much updates. Time-pressure, at least to some extent, came into reality as the amount of testing and testers was decreased.

3 Research Method

In this chapter we will present our research method for this study. The research approach for this study is a systematic literature review which is presented at the beginning of this chapter. After that we will present the research scope, search strategy and selection criteria to define the search process in more detail. We will also present a quality criteria which we will apply to our primary studies to aid the synthesis process. We will also provide forms for our data extraction and gathering of the challenges. Finally we will describe briefly how we conducted the synthesis.

3.1 Research Approach

A *systematic literature review (or systematic review)* is one of the main methods for conducting research. It is originating from medical sciences [KC07] but have also been applied to other fields of science (e.g. social sciences [BCT$^+$06]). The purpose of a systematic literature review study is to identify, evaluate and interpret existing relevant research related to a particular research question, topic area or phenomenon of interest [KC07]. The Cochrane Handbook for Systematic Reviews of Interventions sums the key points of systematic literature review as:

> *Systematic reviews seek to collate all evidence that fits pre-specified eligibility criteria in order to address a specific research question [HG08].*

However, the evidence-based paradigm in software engineering research is quite different from traditional medical research as noticed in a study that investigates the applicability of evidence-based paradigm to software engineering [BCT$^+$06]. Nevertheless, in the same study, the authors noted that there is no obvious reason why systematic literature review could not be successfully adaptable also to software engineering research. They also encouraged the use of evidence-based paradigm to software engineering. Guidelines have been developed to help researchers in employing systematic literature reviews especially in the software engineering domain [KC07].

A *traditional literature review (or literature review)* is a summary of evidence from the studies often conducted by experts or other well-known figures in the field. However, that does not ensure that the information produced is unbiased and reliable [PR08]. In comparison to traditional literature review, which is a selected collection of information gathered by authors that might be biased, a systematic literature review aims to minimize bias by using explicit, systematic methods [HG08] that are documented accurately. In other words, a systematic literature review tries to synthesize existing work in a manner that is fair, is seen to be fair and is (at least to some degree) repeatable [KC07].

Because a systematic literature review does not provide empirical data itself, it is a type of *secondary study* [PFMM08] compared to the publications that are contributing to it (which provide empirical data). Contributing studies are called *primary studies.* Secondary studies are gathering the data from the primary studies to synthesize and integrate evidence to answer some specific research question.

When there are enough secondary studies available related to some research question, there is a possibility to do a review of the secondary studies (so called systematic literature review of systematic literature reviews). The review of the secondary studies is called *tertiary study (or tertiary review).* Tertiary studies are usually conducted to answer research questions with a wider scope [KC07].

According to the guidelines for performing systematic literature reviews in software engineering there exists also a third variation of systematic literature reviews called a *systematic mapping study* [KC07]. In the same guidelines the authors stated that a systematic mapping study can be useful in situations when there is very little evidence existing or the topic is very broad. A systematic mapping study can be made before a systematic literature review to get an overview of the topic area [PFMM08]. After a systematic mapping study is conducted, there is a better possibility to investigate certain topics in more detail. In an example related to our area of interest, Rodriguez et al. have made systematic mapping study about continuous deployment [RHL$^+$16] where some challenges were collected related to continuous deployment.

The purpose of this study is to conduct a systematic literature review of the challenges perceived in the adoption of continuous delivery practice

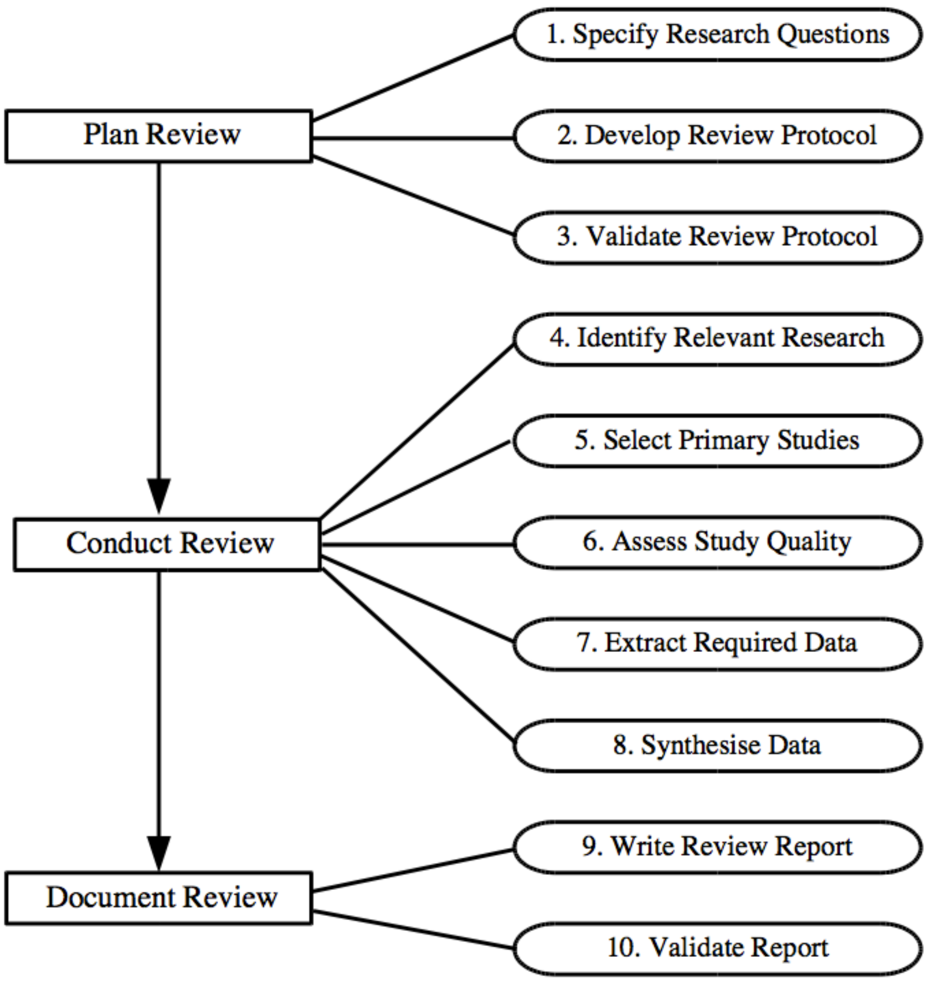


Figure 1: Systematic literature review process [BCT+06]

to software projects. The systematic literature review will be made using the guidelines for systematic literature reviews of Kitchenham et al. [KC07] that are specially made for the software engineering domain. Our process is following the systematic literature review process that is described in a diagram by Budgen et al. [BCT+06] (see Figure 1) and in more detail in the guidelines by Kitchenham et al. [KC07]. The process consists of three different phases: planning the review, conducting the review, and finally documenting the review.

The search strategy used is automatic database search as the primary way of finding our primary studies. This is complemented with manual scan of references found in the selected primary studies. Also, the snowballing strategy [Woh14] was considered as an option for the complementary search for automatic database search but it was rejected because of the assumed laborous influence of the method related to the possible outcomes taking into account the limited amount of time scheduled for the phase of conducting the review. The snowballing strategy could rather have been an option for automatic database search.

3.2 Research Scope

The scope of the systematic literature review may be broad or narrow when answering some research questions. The choice of the scope is usually based on multiple factors like perspectives regarding a question's relevance and potential impact, supporting theoretical information, the potential generalizability and validity of answers to the questions, and available resources [HG08].

The main goal of this study is to gather the information about challenges when adopting continuous delivery practices in software development projects. Because the concept of continuous delivery has been around only for a few years at the time of writing this study, and thus the available research papers are scarce, we chose to have a broad scope for this study. We will include every evidence that is somehow related to adopting the continuous delivery practice. Because of the broad scope, our study faces challenges in categorizing the findings and trying to resolve the common ones. Also discovering the challenges that are specific for the setting and are nongeneralizable is one goal of this review.

Based on preliminary investigation of relevant literature, the continuous delivery term is used interchangeably with the term continuous deployment. Also continuous experimentation is a fairly new approach to software development that would be enabled by continuous deployment [OAB12]. Hence, it might lead us to the challenges met in adopting continuous delivery practice. Because the enabling core technology for both continuous delivery and continuous deployment is deployment pipeline [HF10], we will include also that in our study scope. In addition, practitioners have incorporated the term DevOps with continuous delivery and deployment pipeline so we will include that to our scope too. To learn more about these concepts we ask readers to refer to the section 2.1 of this study.

3.3 Search Strategy

Search strategy is the definition of how the gathering of the relevant studies for the systematic literature review is conducted. It describes different stages of finding and selecting primary data for the study. In an editorial which discusses about systematic literature reviews in software engineering the authors made an observation that the search strategy is the key to ensuring a good starting point for the identification of studies and ultimately for the actual outcome of the study [WP13]. Hence, we are also defining our search strategy. The search of primary studies can be made with a manual process or automatically with the help of database search engines.

Manual search is the form of searching papers by means that do not necessarily need search databases for conducting the search. However, you may utilize search databases to actually get the papers you try to find. One good and relevant example of (systematic) manual search is the *snowballing* approach described in the paper "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering" [Woh14]. In the snowballing approach, you utilize the references of some selected starting set of papers (backwards snowballing) and papers that are citing those papers (forwards snowballing). Gradually you increase the amount of primary studies as you go through the references and citing papers.

Automatic database search, on the other hand, is a way of finding studies from scientific publication databases using search strings. It is noted that even though some studies may have proven that automated database search is better than manual search, it is up to good search strings to ensure the quality of automatic database search [Woh14]. For example, if the search string is too broad there might be substantial manual work in the selection stage of the search process. Also the terminology might not be standardized which may affect the formulation of good search strings.

In the proposed guidelines for systematic literature reviews in software engineering authors state that automatic database search with a good search string from the area of study is a good starting point but complementary search methods are needed [KC07]. They also state that most of the systematic literature reviews in the field are lacking the complementary search methods in addition to automatic database searches. Hence, to complement the automatic database search we are conducting our own additional and small search, which we are calling *reference scanning*, to the primary studies we identified in the automatic database search. The reference scanning is similar to the backward snowballing phase of the snowballing search strategy [Woh14].

In the reference scanning we are going through all the references of our primary studies found in our automatic database search process and try to find more primary studies. For example in a typical scientific publication, there might be references to similar studies in the section that discusses related papers. For every reference we used the same filtering strategy as we used for our automatic database search results. If a new primary study was found then it was added to our primary study pool and reference scanning was done also for that paper. However, we did not find any new primary studies with the reference scanning.

At first we were planning to conduct an additional snowballing search for the most relevant (most cited) papers of the area found in the automatic database search but the idea was rejected at a later stage of the search because of the expected outcome and the workload of the snowballing search taking into account the limited time scope.

As there were suggestions in the "Lessons from applying the systematic literature review process within the software engineering domain" [BKB$^+$07] we searched many different electronic sources in automatic database search. Search databases (indexing services) utilized were IEEE Explore, ACM Digital Library, ScienceDirect and Scopus. We also tried to use CiteSeerX Digital Library (http://citeseerx.ist.psu.edu) and Google Scholar but their exporting capabilities were limited and thus we rejected them. The list of search databases with abbreviation for referencing in this study is provided in Table 2.

The database searches were conducted with the following keywords:

Table 2: List of search databases

Name	**URL**	**Abbreviation**
IEEE Explore	http://ieeexplore.ieee.org	IEE
ACM Digital Library	http://dl.acm.org	ACM
ScienceDirect	http://www.sciencedirect.com	SDT
Scopus	http://www.scopus.com	SCP

continuous delivery, continuous deployment, continuous experimentation, devops and deployment pipeline. We limit the results to only include studies published from 2010 onwards. That was the decision we made based on the assumption that the publication of the book Continuous Delivery [HF10] had influenced the research in the area. The assumption is at least to some extent supported by our brief review of the primary studies in the mapping study of continuous deployment by Rodriquez et al. [RHL+16]; We noticed that in the primary studies that were published before 2010 the concept of continuous delivery did not exist or was too vague. Hence, we decided not to include papers published before that. The complete search expression in generalized form is provided in Figure 2.

('continuous delivery' <or> 'continuous deployment' <or> 'continuous experimentation' <or> 'deployment pipeline' <or> 'devops' <in> (metadata, pdfdata)) <and> publicationyear >= 2010

Figure 2: Generalized search expression

We planned that we would expand the study to include also *grey material* in case the number of the primary papers remains low (eg. less than six studies). Usually grey material is understood as material that is not formally published [HG08]. These kinds of studies include for example books, workshop and conference presentations and abstracts, technical reports, white papers, master's thesis', world wide web articles and other "non-peer reviewed material". Because these kinds of grey literature studies may be of lower quality from a methodological point of view, this should be borne in mind when analyzing these types of studies. At the end of our search process we were left with thirteen primary studies which was a sufficient amount for our study to continue without grey material.

3.4 Selection Criteria

In order to narrow down the amount of primary studies included in this review we made study selection criteria to define which kind of studies we include in our pool of primary studies. In their guidelines for performing systematic literature reviews in software engineering, Kitchenham et al. suggest that these criteria are also used to determine which studies are excluded from a systematic review [KC07].

Firstly, we developed the inclusion criteria to know what kind of studies we should include as our primary studies. Secondly, we created an exclusion criteria to restrict the amount of papers included in the systematic literature review. As we were doing the search, the screening of papers and the final selection of including some study in our primary study pool we incrementally developed the inclusion and exclusion criteria.

Because we are trying to gather synthesis about the perceived challenges, the type of studies to be included in the pool of primary studies should be based on empirical research methods. In a paper that discusses the future of empirical methods in software engineering research, the authors describe empirical research method as *"Empirical research seeks to explore, describe, predict, and explain natural, social, or cognitive phenomena by using evidence based on observation or experience. It involves obtaining and interpreting evidence by, e.g., experimentation, systematic observation, interviews or surveys, or by the careful examination of documents or artifacts."* [SDJ07]. More precisely we are aiming to find experimentation, case study, surveys or action research type of papers. These are the most common empirical research methods in software engineering as noted by authors in the same publication [SDJ07].

When inspecting the challenges presented in surveys we must clearly detect that the challenge is based on experience and not on assumption. In experimentation type of papers we must take into account that they are usually done in a controlled environment and are lacking the uncertainty of uncontrolled environment. In action research, which may be iterative, reflective or linear, we must take into account that the intentions of practitioners is to secure the successful outcome for the client organization and as such may be lacking objectivity [SDJ07].

The main goal is to find evidence-based and objective data to gather the occurred challenges. If a research is not based on evidence or the presented challenges are based on previous works or literature, we exclude those kind of papers and challenges. The main domain is continuous delivery and the keywords are curated in our search strategy based on research scope and preliminary investigation of pre-selected papers.

Type of papers we included in the search results:

1. papers that discussed continuous deployment, continuous delivery, continuous experimentation, deployment pipeline or devops, and
2. were published in 2010 or later.

Type of papers we excluded from the search results (adapted from [RHL$^+$16]):

1. were not related to software domain,
2. were not peer-reviewed science publications,
3. were books or book chapters (non-peer reviewed),
4. were not written in English,
5. were duplicates,
6. did not mention continuous deployment, continuous delivery, continuous experimentation, deployment pipeline or devops in topic or keywords,
7. were not available (i.e. were not available in the network of University of Helsinki without payment),
8. were not based on empirical research method (i.e. were not experiment, case study, survey or action research),
9. were not related enough to continuous delivery of software as an approach to release software more rapidly, and
10. were not discussing the challenges related to adopting continuous delivery in the empirical study results section.

In addition to these selection criteria for our pool of primary studies we also included quality criteria to assess the rigor and relevance of the selected papers. The quality criteria is described in more detail in section 3.6.

3.5 Search Process

An overview of our search process is:

1. conduct automatic database search,
2. conduct reference scanning, and
3. apply quality criteria.

The stages of our automatic database search strategy are:

1. conducting the search on each of the databases based on generalized search string (fig. 2),
2. screening of papers, phase I (topic and metadata applying selection exclusion criteria 1-6),
3. combining the results from different search databases, and
4. screening of papers, phase II (by full text applying selection exclusion criteria 7-10).

Depending on the search database we tried to filter out (with the help of search query) invalid results based on our exclusion criteria. However, we did also execute the first phase of filtering to each of our result set from different search databases. After the first phase we executed the second phase to the rest of the results. Between these stages we combined the result sets from each of the search databases and some duplicates were removed. Stages of automatic database search strategy were adjusted after pilot search in ACM and IEEE databases. More detailed description of database searches and the first phase filtering is available in the section 3.5.1. The description of the second phase filtering is available in the section 3.5.2.

3.5.1 Search and Filtering Phase I

We combined search and filtering phase I to reduce manual work in the first phase filtering. We applied generalized search string with parameters from phase I filtering to each database search engine. Because of differences in search engines the level of filtering varied a bit. Each search query for different databases is described in Table 3 and each search process later in this section. Screen captures of search pages and search results pages are listed in Appendix C.

Table 3: Search queries based on generalized search expression

Service	Search query and prelimenary filtering	Results*
IEE	Search query: *(((((("continuous deployment") or "continuous delivery") or "continuous experimentation") or "deployment pipeline") or "devops") and "software")* Refined by: *Content Type: Conference Publications or Journals & Magazines, Year: 2010-2016*	450 (413)
ACM	Search query: *"query": { ("continuous deployment", "continuous delivery", "continuous experimentation", "deployment pipeline", "devops") } "filter": { "publicationYear": { "gte":2010, "lte":2016 } }, { owners.owner=guide }* Refined by: *Publications: Proceeding, Periodical*	156 (129)
SDT	Search query: *pub-date > 2009 and "continuous deployment" or "continuous delivery" or "continuous experimentation" or "deployment pipeline" or "devops"[All Sources(Computer Science)]* Refined by: *Include only Journals*	92 (90)
SCP	Search query: *all("continuous deployment" or "continuous delivery" or "continuous experimentation" or "deployment pipeline" or "devops") and all("software") and pubyear > 2009 and (limit-to(doctype, "cp") or limit-to(doctype, "ar")) and (limit-to(subjarea, "comp")) and (limit-to(language, "English"))*	317 (302)

** The figure in brackets reflects the amount of results after phase I screening*

Before we applied phase II filtering we combined the results from each database with the help of EndNote basic (https://www.myendnoteweb.com). The EndNote basic ("EndNote") is a free tool for managing a reference library [Reu]. That result set contained 934 publications. We noticed that there were many duplicates that EndNote did not find because there were minor differences in metadata (e.g. author name was presented in a different way) so we manually removed all duplicate entries and finally before entering phase II filtering process we had 734 publications.

After we inspected the results from phase I filtering we noticed that we needed to adjust our phase I filtering because we got too many results before entering to phase II filtering. In phase II filtering we were supposed to address abstracts and conclusions of publications. We decided to add criteria for keywords and topic to get a more precise result set from phase I filtering. Hence, we added exclusion criteria 6 to the phase I filtering. We noticed that our results from ACM were missing keywords so we had to redo the search to ACM database to conclude our phase I filtering with the new exclusion criteria based on keywords. After the redo of phase I filtering with added exclusion criteria 6 we got 152 publications.

IEEE: We did an advanced search in the IEEE Xplore Digital Library on 12 April 2016 (full-text and metadata) applying generalized search string with selection criteria to only include conference publications, journals and magazines. Thereby we got 541 search results. To narrow down the search results we decided to add an obligatory word "software" to the search string to filter out non-software domain related papers. After that we got 450 search results. We imported these references to EndNote Basic in our IEEE search results group for further processing. When the data was listed in EndNote we noticed that four (4) references had no authors provided because they were conference keynotes, table of contents or speaker presentations so we removed them as invalid results. The unique count after automatic database search was 446. No duplicates were found at this stage. After that we did the first phase of screening (based on topic and metadata) and were left with 413 papers from IEEE Xplore Digital Library.

ACM: We did an advanced search in the ACM Guide to Computing Literature on 21 April 2016 applying the generalized search string and got 224 results. There were three types of publications in the result set: periodicals (28), proceedings (128), and books (68). Because we were not supposed to include books in our primary study pool we selected results only to include publications that were periodicals or proceedings so we got 156 publications. After that we imported those results to EndNote Basic in our ACM search results group for further processing. When the data was listed in EndNote we noticed that four (4) references had no authors provided because they were conference welcome letters or book summaries so we removed them as invalid results. Also thirteen (13) duplicate papers were found so the final count for unique papers after the automatic database search in ACM was 139. After that we did first phase of screening (based on topic and metadata) and we found still one extra duplicate paper. Finally after first phase of screening we were left with 129 papers as a result from ACM.

ScienceDirect: We did an expert search in the Elsevier's ScienceDirect search database on 16 April 2016. We applied generalized search string and applied content filter to include only journal articles (books were excluded). We also limited search results to only contain publications from computer science field. This is how we got 92 search results. We imported the results to EndNote in our ScienceDirect search results group for further processing. In EndNote we noticed that two (2) of the publications did not have any author information because they were index papers so we deleted those. No duplicate articles were found within ScienceDirect search results group. After that we were left with 90 search results. We did phase I filtering for the results but all publications passed the filtering.

Scopus: We did an advanced search also in the Elsevier's Scopus bibliographic database on 17 April 2016. We applied the generalized search string and applied content filter to include only conference papers and articles written in English language and only in computer science subject area. With the database search we got 317 search results. We imported the search results to EndNote in our Scopus search results group and found five (5) duplicate papers which we deleted. After that we did phase I filtering and got 302 publications passing the phase I filter.

3.5.2 Filtering Phase II

After the filtering phase I, we had 152 papers for filtering phase II. We read the abstract from each paper to get a good overview and then we scanned the papers to find words such as challenge (challeng*), problem (problem*), barrier (barrier*), issue (issue*), trouble (trouble*), or obstacle (obstacle*). If such a word was found, we analyzed the context of the sentence to find out if it was strictly related to continuous delivery domain. If a correct type of sentence was found and if it was in the section of empirical results of study (case study, survey or action report section) the paper was included.

After the phase II filter we had 13 papers selected in our primary study pool. The results from each stage is shown in Table 4. The number of primary studies found in each search databases is shown in Table 5. Almost all of the primary studies would have been found from SCP search database. Only two of the primary studies were not found from SCP search database. Five primary studies existed only in SCP search database and one primary study existed only in IEE search database. All studies would have been found only with the help of IEE and SCP search databases. List of resulting primary studies is shown in Table 6. Bibliographic information of primary studies is available at Appendix A.

Table 4: Search results after each stage

After search	**After phase I**	**After phase II**
934	152	13

Table 5: Primary studies found from each search database

Search database	**Count**	**Studies**
IEE	7	S1, S3, S4, S6, S7, S8, S9
ACM	3	S6, S8, S9
SDT	1	S2
SCP	11	S1, S2, S3, S4, S5, S8, S9, S10, S11, S12, S13

3.6 Quality Criteria

To assess the quality of individual studies included in the systematic literature review researchers should develop quality criteria checklists [KC07]. As the author noted in a study of procedures performing systematic literature reviews [Kit04], it is important to define the quality criteria because it can provide:

- extra inclusion and exclusion criteria based on the quality of the paper,
- explanation for differences in study results originating from quality,

- weighting of importance of primary studies in the synthesis phase,
- guidance for interpretation of findings and strength of inferences, and
- guidance for recommendations for future research.

Table 6: List of primary studies

Study	**Title**
S1	Continuous Delivery: Huge Benefits, but Challenges Too
S2	On the Journey to Continuous Deployment: Technical and Social Challenges Along the Way
S3	ResearchOps: The Case for DevOps in Scientific Applications
S4	Automated Testing in the Continuous Delivery Pipeline: A Case Study of an Online Company
S5	Hitting the Target: Practices for Moving Toward Innovation Experiment Systems
S6	Towards DevOps in the Embedded Systems Domain: Why is It So Hard?
S7	The Highways and Country Roads to Continuous Deployment
S8	Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)
S9	Climbing the "Stairway to Heaven" - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software
S10	Towards R&D as Innovation Experiment Systems: A Framework For Moving Beyond Agile Software Development
S11	Towards Agile and Beyond: An Empirical Account on the Challenges Involved When Advancing Software Development Practices
S12	Transitioning Towards Continuous Delivery in the B2B Domain: A Case Study
S13	DevOps: A Definition and Perceived Adoption Impediments

Even though the exact definition for "quality" is not clearly stated, it is said that "quality" assesses the minimization of bias (systematic error from the "true" results) and internal (preventing error when conducting the study) and external validity (generalisability of results outside the study) [Kit04].

In a quality checklist study [KBBL09] authors evaluate quality checklist proposals applied to systematic literature reviews in software engineering. Even though they state that general guidelines for quality checklists do not provide sufficient help to anyone to allow them to construct appropriate quality checklists for specific systematic literature review, the guidelines do have some value as a starting point for constructing such a quality checklist.

Table 7: Quality checklist questions (adapted from [DD08], [ABCS10], and [GWT+14])

Id	**Question**
Q1	Is the paper based on research (or is it merely a “lessons learned” report based on expert opinion)?
Q2	Is there a clear statement of the aims of the research?
Q3	Is there an adequate description of the context (e.g. industry, laboratory setting, products used, etc.) in which the research was carried out?
Q4	Is there a justification and a description for the research design?
Q5	Is the research design appropriate to address the aims of the research?
Q6	Is the data collected in a way that addressed the research issue?
Q7	Is there a clear statement of the findings?
Q8	Is the data analysis sufficiently rigorous?
Q9	Is the relationship between the researcher and the participants considered to an adequate degree?
Q10	Do the authors discuss the credibility of their findings?
Q11	Are limitations of the study discussed explicitly?

In the same paper authors provide a generalized quality checklist that works as a good starting point for human-based and quasi-experiments primary studies. As such the checklist should be tailored to fit a specific systematic literature review, keeping in mind the following aspects:

- answers should preferably have some ordinal scale, not just yes or no,
- number of items should be limited between 6-12,
- questions should be answered as objectively as possible, and
- team should discuss, refine and agree on appropriate quality checklist items.

Because the purpose of the quality assessment will guide the development of checklists [KC07] we created an appropriate checklist considering the research questions presented in this study. We also took into account the above guidelines except the last point. The quality checklist can be seen in Table 7.

Other studies have previously used a three point scale [DD08] [ABCS10] [GWT+14] and we also applied the same scale: yes (1 point), to some extent (0.5 points) and no (0 points), to answer the questions. This decision is also in line with the quality checklist recommendations [KBBL09]. The quality score for each individual study was the sum of the points of each question. Quality criteria were assessed after the primary study selection was made so it was not used as a filter to our primary study selection process.

3.7 Data Extraction

As instructed in the guidelines [KC07] we will next provide information about how we obtained the information required from each primary study. Selected primary studies were carefully read through in order to extract the data needed for our synthesis of continuous delivery challenges. To obtain data for all research questions we created two extraction forms before conducting the extraction process. After piloting the extraction process we realized that we needed a third separate form for the challenges.

Table 8: General information form (Form I)

Field	**Description**
General Information	
Identifier	Identifier for primary study
Authors	Names of the authors
Year	Year of publication
Title	Title of publication
Reference	Bibliographic reference
Paper type	Is the paper from a conference or a journal
Source	From which source the article was aquired
Search database	Which search databases included the study in search results
Citation count	How many times the paper has been cited*
Quality score	Quality score provided by quality checklist
Research Specific	
Research type	Which research method was used (case study, experiment, survey, action research)
Data collection	Was the data collected by questionnaires, interviews, forms or some other way
Pertinence	Is the paper fully, partially, or marginally related to our scope

** Citation count from ResearchGate (https://www.researchgate.net) on the 18th of June 2016*

The first form is for general information about the inspected study. There is general information i.e. authors, year of publication, research type, etc. about each individual study. The information of the first form can be seen in

Table 8. The second form is for the cases that discussed the challenges found in primary studies. The information of the second form can be seen in Table 9. Because there might be multiple challenges perceived in each individual study and each individual case, we decided to collect the challenges to a separate third form. The information of the third form can be seen in Table 10. The relations between these data extraction forms (and data) is seen in Figure 3. As we can see from that table there can be multiple cases inspected from one primary study and in one case there can be multiple challenges inspected.

Table 9: Case information form (Form II)

Field	**Description**
Identifier	Identifier for the case
Primary Study	Reference to primary study where this case is presented
Software Domain	Software domain for the case (embedded systems, web software, mobile software, or something else)
Industry Domain	Industry domain for the case (telecom network, web service, mobile gaming, etc.)
Product or service	What is the product or service the case is considering
Scope	Scope considering the practice of delivering software more rapidly (continuous delivery, continuous deployment, or both)
Organization size	Size of the organization
Team size	Size of the team the case is related to

Table 10: Challenge data extraction form (Form III)

Field	**Description**
Identifier	Identifier for the challenge
Case	Case this challenge belongs to
Category	Challenge category as described in the original study
Title	Brief interpretation of the challenge by inspector (or as in the original study if available)
Description	Challenge as descibed in the original study
Mitigation strategy	How the organization reacted or was planning to react to overcome the challenge
Research proposal	Research proposal from the authors of the primary study

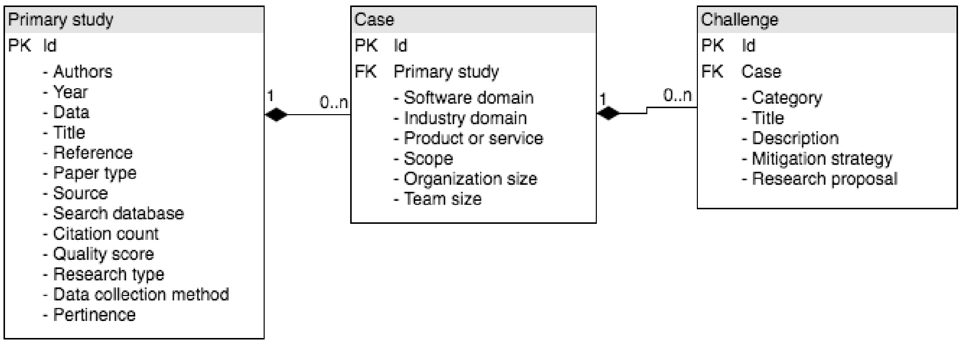


Figure 3: Relations of data extraction forms (and data)

3.8 Synthesis Strategy

After the data was extracted we synthesized it to answer the presented research questions. Synthesis process was done according to our synthesis strategy which we describe in this section. Synthesis strategy should state what kind of techniques are used to provide the analysis i.e. formal meta-analysis or some other technique [KC07]. As noted in papers which have studied the experiences of applying systematic literature reviews to software engineering domains it may not be possible or suitable to perform meta-analysis for software engineering studies because of the qualitative type of data [BKB+07] [DDH07]. Formal meta-analysis techniques are based on statistical techniques to obtain quantitative synthesis of data [Kit04]. Because our data is more of a qualitative type, we did not use any formal meta-analysis technique to synthesize the data. Instead we used a narrative approach to the synthesis i.e. *narrative synthesis*. A paper which studied narrative approaches to synthesis of evidence describes narrative synthesis as a *"family of methods for synthesizing data narratively, focusing particularly on the application of narrative approaches to the synthesis of qualitative evidence"* [SOV12]. As opposed to a statistical based meta-analysis approach, narrative synthesis is used *"to seek and generate new insights and recommendations by going beyond the summary of findings from different studies as in traditional narrative reviews"* [SOV12]. Many different types of approaches are found to exist to provide qualitative analysis such as content analysis, thematic summaries, framework synthesis, thematic synthesis, realist synthesis, meta-ethnography, grounded theory, textual narrative synthesis, meta-study, meta-narrative, critical interpretive synthesis, and ecological triangulation [BPT09] [SOV12].

We applied the thematic synthesis [TH08] approach to our synthesis strategy taking into consideration the recommended steps for thematic synthesis in software engineering [CD11]. After the extraction of data was done we started creating the synthesis. First, in the coding stage, we added labels to the findings (raw challenges) and categorized the labels by similarities (i.e. created descriptive themes). This was done by utilizing a mind mapping tool. In the coding stage we utilized the so called integrated approach [CD11] to create the codes where we had pre-specified (collected) codes but also created new ones along the way. With the help of the labels (i.e. challenges), we answered the RQ1.1 by the means usually familiar from the content analysis: frequency of findings i.e. how many findings (raw challenges) were related to a certain label. We also created descriptive themes to categorize these labels at this stage of synthesis. To better answer the RQ1.2 and RQ1.3 we also added information of the domain (web software, customer specific web software, embedded systems, scientific software, or mobile software) in which the label usually appeared in. From now on we are referring to the label as a challenge.

After we had found some answers to our RQ1.1-3, or actually realized that we could not provide answer to RQ1.3, we tried to find the causes behind these challenges (RQ1.4). To get an answer to our last sub research question (RQ1.4) of RQ1 we put our categorization aside and created new descriptive themes to figure out the reasons behind these challenges that emerged from these labels. Lastly we created higher-order themes [CD11] (or higher-order interpretations or analytic themes) to better answer the RQ1.4 and thus to complete our analysis for RQ1. These reasons behind the challenges provided also some insights to our sub research question RQ1.3.

4 Results

In this chapter we will present our results based on the collected data and the first part of synthesis process. First we will give a brief overview of the collected data to have some insight of what kind of data we collected and what is the quality of the included publications (answer to the RQ2.1) in our primary study pool.

After the data overview we will enter into the synthesis of perceived challenges where we provide answers to our first main research question: "What are the perceived challenges of adopting continuous delivery practices in a software development project and why?". We will list all perceived challenges (RQ1.1) with the following information: frequency, domain, organization size, and evidence level. Full description of all of the observed challenges is available at Appendix B. Raw challenges (i.e. findings) are available at Appendix D.

4.1 Data Overview

Table 11: General information of primary studies

Study	Year	Paper type	Citations*	Quality score	Evidence
S1	2015	Journal	6	2	low
S2	2015	Journal	14	10	high
S3	2015	Conference	0	4.5	medium
S4	2015	Conference	0	2.5	low
S5	2015	Conference	0	8	high
S6	2016	Conference	0	10	high
S7	2015	Journal	4	8	high
S8	2013	Conference	13	2.5	low
S9	2012	Conference	39	8.5	high
S10	2013	Conference	16	8.5	high
S11	2014	Conference	1	6	medium
S12	2015	Conference	0	8	high
S13	2015	Conference	3	8.5	high

** Citation count from ResearchGate (https://www.researchgate.net) on the 18th of June 2016*

We found 13 different research papers to be included in our primary studies pool. Primary studies are presented in Table 6. Bibliographic information of

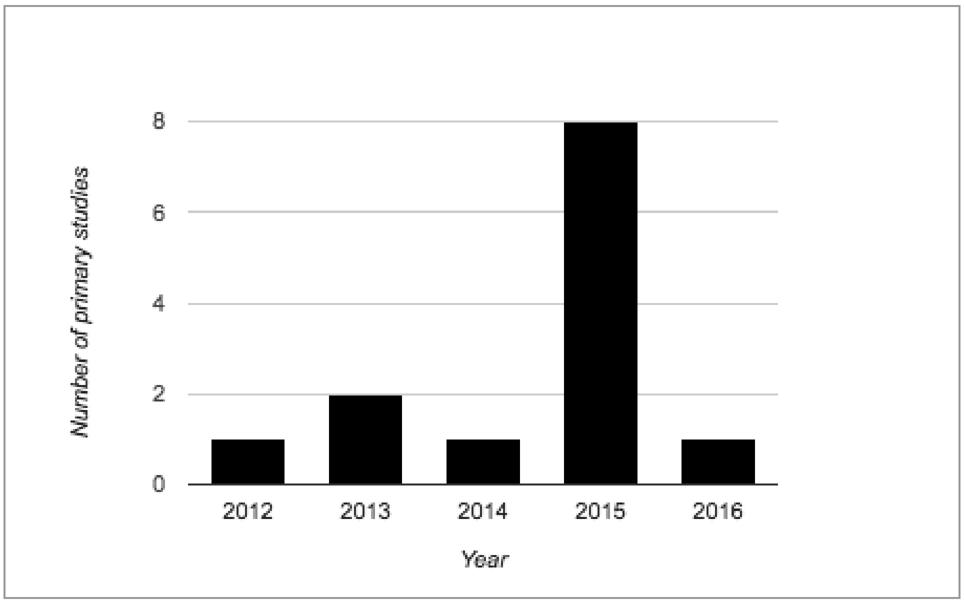


Figure 4: Publication year distribution

primary studies is available at Appendix A. General information about the publications is presented in Table 11. Five of the papers did not have any citing publications and four of the papers got only few citations. Three of the publications had moderate citation count and one of the papers had over two times more citations (39) than the second most cited (16). Most of the papers were published in 2015. The number of primary studies per year is presented in Figure 4.

The evidence level of our findings was based on the quality score obtained by the quality score checklist shown in Table 7. We had a quality score scale from 0 to 13 points and the distribution of our primary studies' quality scores was between 2 and 10 points (see Table 11). We divided our quality scores into thirds after normalization of our quality scores so that we had three different categories of evidence (based on quality score): low (2-4 pts.), medium (5-7 pts.), and high (8-10 pts.). The distribution of quality scores is shown in Figure 5. Even though relatively few had a moderate citation count, eight out of thirteen studies were considered as a high quality based on the obtained quality score. Only three of the primary studies were considered as having a low evidence level and they were all action research type of studies.

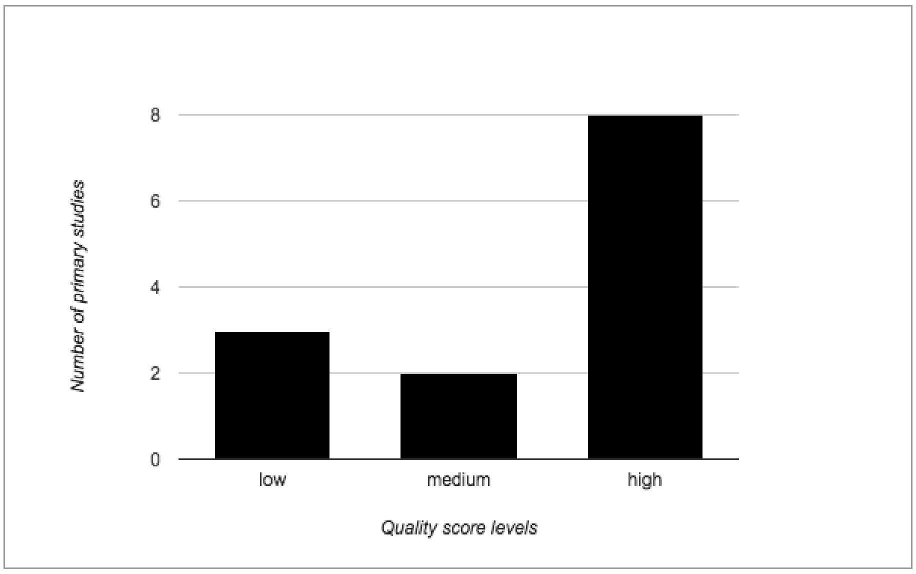


Figure 5: Quality score distribution

The remaining action research type of paper got medium evidence level as scoring fourth-worst by 4.5 scores. A more detailed view to our quality scores is presented in Table 12. Questions for quality score checklist are available at Table 7 in section 3.6. Quality scores were taken into consideration when analyzing the perceived challenges in the data synthesis phase.

Most weakest points were given for the first quality score checklist question *"Is the paper based on research (or is it merely a "lessons learned" report based on expert opinion)?"*. This was because all of the studies were either action research or case studies with no controlled environment. Action research type of papers were given zero points while case studies were given half a point. Also the three last questions Q9: *Is the relationship between the researcher and the participants considered to an adequate degree?*, Q10: *Do the authors discuss the credibility of their findings?*, and Q11: *Are limitations of the study discussed explicitly?* got rather low points overall (6 points or lower). The rest of the questions got 8 or more points.

Research specific information is presented in Table 13. In the case of action research type of papers the data collection method was type of participant observation or it was not described and it was not derivable from the content (i.e. it was unknown). Those kind of papers were usually retrospective reports

Table 12: Quality score matrix

Study	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Score
S1	0	0	1	0	0	0	0.5	0.5	0	0	0	2
S2	0.5	1	1	1	1	1	0.5	1	1	1	1	10
S3	0	1	0.5	0	0	0.5	1	0.5	1	0	0	4.5
S4	0	0	1	0	0	0.5	1	0	0	0	0	2.5
S5	0.5	1	0.5	1	1	1	1	0.5	0.5	0.5	0.5	8
S6	0.5	1	1	1	1	1	1	1	0.5	1	1	10
S7	0.5	1	0.5	1	1	1	1	1	0.5	0.5	0	8
S8	0	0.5	1	0	0	0	0.5	0.5	0	0	0	2.5
S9	0.5	1	0.5	1	1	1	1	0.5	0.5	1	0.5	8.5
S10	0.5	1	0.5	1	1	1	1	0.5	0.5	1	0.5	8.5
S11	0.5	1	0.5	0.5	1	1	0.5	0.5	0.5	0	0	6
S12	0.5	1	0.5	0.5	1	1	1	1	0	0.5	1	8
S13	0.5	1	0.5	1	1	1	1	1	0	0.5	1	8.5
Overall	4.5	10.5	9	8	9	10	11	8.5	5	6	5.5	-

Table 13: Research specific information of primary studies

Study	Research type	Main data collection method	Pertinence
S1	Action research	Participant observation	Full
S2	Case study	Semi-structured interviews	Full
S3	Action research	Participant observation	Partial
S4	Action research	Unknown	Full
S5	Multiple case study	Semi-structured interviews	Partial
S6	Multiple case study	Semi-structured interviews	Full
S7	Multiple case study	Semi-structured interviews	Full
S8	Action research	Participant observation	Full
S9	Multiple case study	Semi-structured interviews	Partial
S10	Multiple case study	Semi-structured interviews	Partial
S11	Case study	Semi-structured interviews	Partial
S12	Multiple case study	Semi-structured interviews	Full
S13	Case study	Semi-structured interviews	Partial

of adopting continuous delivery practices by some expert from the adopting organization. In all case study papers the used data collection method was semi-structured interviews, sometimes accompanied with open-ended questions, field notes, and other project related documentation.

From thirteen research papers we identified 28 different cases from where the challenges were found. All cases are presented in Table 14. Most of the cases were considering web software or embedded systems domain. In web software domain, there were two distinct type of applications that we felt should be considered differently: web software as a product that is installed in a certain environment that is used by many customers (typical web application), and web software that is installed on specific user environment (customer hosted or some other customer specific environment). We noticed that there were slightly different challenges between those types of web software. We took this into account when we analyzed the domain specific (and common) challenges. Only in the cases that the raw challenge was related to a product that had customer specific environments we labeled it as being in such a domain.

If the software domain was not explicitly mentioned or it was not easily deducible from the text or context, we stated that the domain was unknown. Two of the cases belong to such unknown category. In one multiple case study some of the challenges spanned on multiple cases and thus was labeled as various. In addition to web software, customer specific web software, and embedded systems two other software domains were inspected, both with one case study: scientific software and mobile software. Scientific software considered customized high-performance computing software products that were used by the case organization or its customers (probably in supercomputers). Mobile software considered typical mobile application with application distribution system (like Apple's App Store or Google's Google Play) as the way of installation.

We categorized organization size as micro (less than 10 people), small (10 - 49 people), medium (50 - 249 people), and large (over 250 people). If the organization size was not stated the case organization size was unknown and the challenges found from such data were not related to any size of organization. The size of organization was known in only twelve of 28 cases. Two of those were micro, four medium, and six large organizations. The team size was sometimes ambiguous from the data so we decided not to use that in our analysis.

Table 14: Case descriptions

Case	Study	Software Domain	Industry Domain	Product or service	Scope	Org. Size	Team Size
CS1	S1	Web software	Internet service	Web software	Continuous Delivery	Large	4-8
CS2	S2	Web software / Customer specific	Internet service	SaaS and customer-hosted platforms	Continuous Deployment	Large	Unknown
CS3	S3	Scienctific software	Science	Scientific software, for customer use also	Continuous Delivery	Unknown	Unknown
CS4	S4	Web software	Internet service	Web software	Continuous Delivery	Large	Unknown
CS5	S5	Embedded systems	Wireless embedded system	Customable software solution for device	Continuous Deployment	Unknown	Unknown
CS6	S5	Embedded systems	Telecom network	Compact mobile broadband solution	Continuous Deployment	Unknown	Unknown
CS7	S5	Embedded systems	Telecom network	Network traffic-monitoring tool	Continuous Deployment	Unknown	Unknown
CS8	S5	Embedded systems	Industrial automation	Factory automation platform solution	Continuous Deployment	Unknown	Unknown
CS9	S6	Embedded systems	Wireless embedded system	Customable software solution for device	Continuous Deployment	Unknown	Unknown
CS10	S6	Embedded systems	Telecom network	Compact mobile broadband solution	Continuous Deployment	Unknown	Unknown
CS11	S6	Embedded systems	Industrial automation	Factory automation platform solution	Continuous Deployment	Unknown	Unknown
CS12	S6	Embedded systems	Telecom network	Network traffic-monitoring tool	Continuous Deployment	Unknown	Unknown
CS13	S7	Various	Various	Various	Continuous Deployment	Unknown	Unknown
CS14	S7	Embedded systems	Medical embedded system	Medical software	Continuous Deployment	Medium	10
CS15	S7	Embedded systems	Industrial automation	Industrial automation	Continuous Deployment	Large	50
CS16	S7	Mobile software	Mobile gaming	Game	Continuous Deployment	Micro	3
CS17	S7	Web software	Web software development	Web software	Continuous Deployment	Micro	7
CS18	S7	Web software	Internet service	Web service	Continuous Deployment	Medium	3
CS19	S7	Web software	Web software development	Web framework	Continuous Deployment	Medium	7
CS20	S7	Web software	Web software development	Web software development	Continuous Deployment	Medium	8
CS21	S7	Embedded systems	Telecom network	Unknown	Continuous Deployment	Large	Approx. 100
CS22	S8	Web software	Internet service	SaaS product	Continuous Delivery	Unknown	Unknown
CS23	S9	Embedded systems	Telecom and multimedia solutions for mobile and network	Mobile communication solutions and telecommunication infrastructure components	Continuous Deployment	Unknown	Unknown
CS24	S10	Embedded systems	Telecom and multimedia solutions for mobile and network	Telecommunication systems and equipment, communications networks and multimedia solutions for mobile and fixed networks	Continuous Deployment	Unknown	Unknown
CS25	S10	Unknown	Finance tools and services	Financial and accountings software products and services for consumers, small businesses, acountants, financial institutions and healtcare providers	Continuous Deployment	Unknown	Unknown
CS26	S11	Embedded systems	Network video	Network cameras, video encodes, video management software and camera applications for video surveillance	Continuous Deployment	Unknown	Unknown
CS27	S12	Web software / Customer specific	Marketing automation	Marketing automation tool and master data management solution as background service	Continuous Delivery	Unknown	Approx. 6
CS28	S13	Unknown	International IT company	Unknown	Both	Large	Unknown

4.2 Perceived Challenges

All 113 findings (raw challenges) that we extracted (see Appendix D) from the primary studies were analyzed and coded into labels of challenges. After analyzing the findings we found that two challenges were basically the same (58 and 59) and one was not an issue (98) so the resulting amount of findings was 111. Then labels were arranged into higher-level categories. Categorized challenges are listed in Figure 6. After the analysis was done for the findings, 59 different challenges were found. More detailed descriptions of challenges are available in Appendix B where mitigation strategies, research proposals and related findings are also listed.

With the help of collected data and based on our analysis of the findings we noticed that the challenges were considering either technical or social aspects of software development. Challenges on the social side were divided into two main categories: procedural and organizational challenges. Organizational challenges were divided into subcategories of resources, marketing and sales, customer adoption, hierarchy, and culture. Procedural challenges were not divided into any subcategories. Technical challenges were divided into nine different subcategories (tools, architectural, documentation, testing, resources, dependencies, deployment, security and safety, and integrations) each containing two to nine challenges. Most of the challenges that we found were on the technical side (34 of 59) while the social side had 25 observed challenges.

In Tables 15, 16 and 17 we list the challenges with the information about the frequency of the findings, software domain of the case organization where the challenge occurred, size of the organization and the evidence level of the primary study based on our quality criteria checklist. The software domain of the case organization is presented as a letter that represents the following software domains: W as web software, C as customer specific (web) software, E as embedded systems, S as scientific software, and M as mobile software. Occurrences of the challenge in question is marked on the domain column. In some challenges the frequency of challenge is more than the sum of the occurrences in the domain columns which indicates that some observations were found from a case with an unknown domain. Sizes of companies are also represented as letters with the following meanings: T as micro, S as small, M as medium, and L as large size organization.

Level of evidence is presented in the same way as previously mentioned with the general information of primary studies in the data overview (section 4.1). Only the most highest evidence level of the primary studies from where the findings were found was preserved. Most of the challenges had evidence level of high (49 challenges), while medium and low evidence level both had five challenges. Mainly all of the challenges that got the evidence level of low or medium had a frequency of one and they did not span to more than one domain.

We also created high-order themes from these challenges to answer one of our research questions considering the reasons behind these challenges. Analyzed reasons will be presented in section 5.2.

Table 15: Procedural challenges (social)

Id	Challenge label	Freq.	W	C	E	S	M	Sizes	Evidence
C1	Incompatibility of processes	2	1		1			L	High
C2	Working in small batches	3	3					L	High
C3	Utilizing shorter feedback loops	1	1					L	High
C4	Lack of standard practices	2	1		1			L	High
C5	Keeping the build green	1	1					L	High
C6	Propagation of changes	1			1			-	High
C7	Ambiguity in coordination	1	1					L	High

Table 16: Organizational challenges (social)

Id	Challenge label	Freq.	W	C	E	S	M	Sizes	Evidence
Resources									
C8	Lack of motivation	4	3					T, L	High
C9	Increased pressure	4	2	1				T, L	High
C10	New roles	1	1					L	High
C11	Lack of knowledge	1	1					L	High
C12	Deep specialization	1			1			-	High
C13	Fear of unemployment	1	1					-	Low
Marketing and sales									
C14	Marketing a versionless product	1	1					L	High
C15	Uncertainty of ready features	1	1					-	Low
C16	Selling experimental functionality	1						-	High
C17	Customer policies	2	1		1			M, L	High
Customer adoption									
C18	Feature discovery	1	1					L	High
C19	Cycle of update (and feedback)	8	3	1	3			L	High

Hierarchy									
C20	Coordinating units	1	1					L	High
C21	Barriers between units	1	1					L	Low
C22	Different pace between units	2	1		1			-	High
Culture									
C23	Releasing experimental functionality	1						-	High
C24	Sharing of status	2			2			-	High
C25	Lack of business model	3		1	2			-	High

Table 17: Technical challenges

Id	**Challenge label**	**Freq.**	**W**	**C**	**E**	**S**	**M**	**Sizes**	**Evidence**
Tools									
C26	No comprehensive platform suite	1	1					L	Low
C27	Branching in VCS	1	1					L	High
C28	Cannot use optimal hardware resources	1				1		-	Medium
Architectural									
C29	Incompatible architectures	2	1					L	High
C30	Legacy system integrations	1	1					M	High
C31	Codebase size	1	1					M	High
Documentation									
C32	Maintaining documentation	1		1				L	High
C33	Maintaining change logs	1		1				-	High
Testing									
C34	Insufficient testing	8	4		3			L	High
C35	Testability of outputs	1				1		-	Medium
C36	Long acceptance tests	2	1		1			M	High
C37	Configuration differences	3	1		2			M	High
C38	Automated UI testing	2	1				1	T	High
C39	Performance testing	1			1			L	High
C40	Manual testing	2	1					M	High
C41	Differences in environments	1						L	High
C42	Ignoring test suite	2	2					-	Low
Resources									
C43	Infrastructural requirements	1	1					L	High
C44	Hardware based infrastructure	1			1			-	High
Dependencies									
C45	Cross-product dependencies	4	3	1				M, L	High
C46	No packages available in repositories	1				1		-	Medium
C47	Outdated environments	1				1		-	Medium
C48	Long product lifecycle	1			1			-	High
Deployment									
C49	Seamless upgrades	5	2	1	2			L	High
C50	Diversity of client configurations	5		3	1			L	High
C51	Third-party distribution channels	1					1	T	High
C52	Efficient rollback mechanism	1			1			-	Medium
Security and safety									
C53	Critical systems	1			1			-	High

C54	Not enough access	1						L	High
C55	Limited view on run environment	2			2			-	High
C56	Product quality may decrease	1						L	High
Integrations									
C57	Partner plugin integration	1	1					L	High
C58	Changes in database schema	1	1					L	High
C59	Multiple versions out	2		1				-	High

Table 18: Mitigation strategies and research proposals

	Challenges	**Sum**
Mitigation Strategy	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C18, C19, C20, C21, C22, C23, C24, C25, C26, C27, C29, C32, C34, C42, C43, C44, C45, C46, C47, C49, C56, C57, C58	38 (64%)
Research Proposal	C1, C2, C9, C16, C21, C26, C29	7 (12%)

As we can see in Table 18, 64% of the challenges got some kind of observed mitigation strategy. Mitigation strategies varied from as simple as *"Apply continuous deployment for non-business critical software development"* [S2] to more advanced as identifying and engaging pilot customers to successfully adopt the continuous deployment culture and capability [S10]. Research proposals were less identified as only 12% got some observed proposal. However, 33% of the challenges did not have any observed mitigation strategy or research proposal.

Almost all challenges (except C16: Selling experimental functionality and C17: Customer policies) in social category had some kind of observed mitigation strategy. Also 5 out of 7 research proposals were considering social aspects. Only two research proposals were considering technical issues addressing the issue of missing comprehensive continuous delivery platform suite and non-modular architectures. While there were 38 mitigation strategies found overall, 13 of them were considering technical challenges. Hence, 21 of technical challenges did not have any observed mitigation strategy.

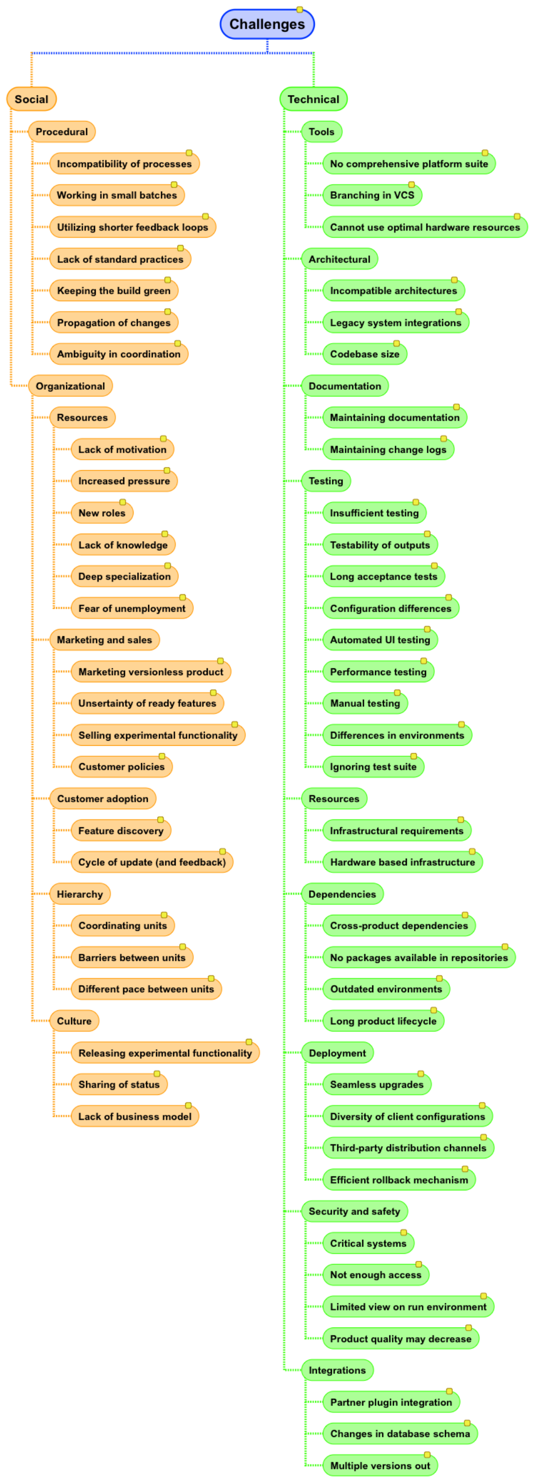

Challenges
Social
Procedural
Incompatibility of processes
Working in small batches
Utilizing shorter feedback loops
Lack of standard practices
Keeping the build green
Propagation of changes
Ambiguity in coordination
Organizational
Resources
Lack of motivation
Increased pressure
New roles
Lack of knowledge
Deep specialization
Fear of unemployment
Marketing and sales
Marketing versionless product
Unsertainty of ready features
Selling experimental functionality
Customer policies
Customer adoption
Feature discovery
Cycle of update (and feedback)
Hierarchy
Coordinating units
Barriers between units
Different pace between units
Culture
Releasing experimental functionality
Sharing of status
Lack of business model
Technical
Tools
No comprehensive platform suite
Branching in VCS
Cannot use optimal hardware resources
Architectural
Incompatible architectures
Legacy system integrations
Codebase size
Documentation
Maintaining documentation
Maintaining change logs
Testing
Insufficient testing
Testability of outputs
Long acceptance tests
Configuration differences
Automated UI testing
Performance testing
Manual testing
Differences in environments
Ignoring test suite
Resources
Infrastructural requirements
Hardware based infrastructure
Dependencies
Cross-product dependencies
No packages available in repositories
Outdated environments
Long product lifecycle
Deployment
Seamless upgrades
Diversity of client configurations
Third-party distribution channels
Efficient rollback mechanism
Security and safety
Critical systems
Not enough access
Limited view on run environment
Product quality may decrease
Integrations
Partner plugin integration
Changes in database schema
Multiple versions out

Figure 6: Categorized challenges

5 Challenge Analysis

In this chapter we will try to find and analyze the common and specific challenges (RQ1.2, RQ1.3) based on the domain and frequency analysis. We will also try to go beyond the data (perceived challenges) to analyze the reasons behind these challenges (RQ1.4) in sections 5.2 and 5.3. Finally we will go through observed research proposals in section 5.4.

Also for every common challenge we will provide data on how organizations have reacted to the challenge (mitigation strategies) and what are the research proposals regarding the challenges from the authors of the primary studies (see section 5.4). This information is the grounding information for our second main research question: "How should the research on continuous delivery be directed according to these challenges?". Answer to the second main research question will be processed in the concluding chapter (see section 7.2). Mitigation strategies and research proposals are also available for reading in the Appendix B.

Along with the common challenges we tried to find specific challenges to some domain or size of an organization. Our definition for a challenge to be specific for a certain domain was that there is high frequency and high level of evidence but no span across the domains. With such a definition and with our data we could not find any challenges that were specific to certain software domains. We did not find any other reliable way to determine which of the challenges were more specific to a certain domain. We also tried to consider reliable ways to determine if the size of organization would have any effect on the challenges but failed to find any. However, we did find some characteristics in domains regarding the reasons for the challenges (see section 5.3).

5.1 Common Challenges

We defined a challenge as a common one when it spanned to more than one software domain and had a high evidence level. The common challenges with most frequent occurrences (5 or more occurrences) were:

- customers' adoption of fast cycle of update (and feedback),

Table 19: Common challenges

Id	Challenge	Category
Social		
C1	Incompatibility of processes	Procedural
C4	Lack of standard practices	Procedural
C9	Increased pressure	Organizational (recources)
C17	Customer policies	Organizational (marketing and sales)
C19	Cycle of update (and feedback)	Organizational (customer adoption)
C22	Different pace between units	Organizational (hierarchy)
C25	Lack of business model	Organizational (culture)
Technical		
C34	Insufficient testing	Testing
C36	Long acceptance tests	Testing
C37	Configuration differences	Testing
C38	Automated UI testing	Testing
C45	Cross-product dependencies	Dependencies
C49	Seamless upgrades	Deployment
C50	Diversity of client configurations	Deployment

- insufficient level of automated testing,
- configuration diversity in different environments (especially in deployment phase but also in testing), and
- implementing seamless upgrades that are automated and reliable.

The category with the most perceived common challenges was testing (3 common challenges) which was on the technical side of the observed challenges. It was also the category with the most number of identified different challenges (9 different challenges). Next we will go through all 14 common challenges (see Table 19) we observed in the order they appear in Figure 6 (not in the order of importance). Mitigation strategies and research proposals are also presented in addition to the challenge description if observed from the primary studies.

5.1.1 Procedural: Incompatibility of processes (C1)

Many traditional processes are too slow for continuous delivery. There is usually stages involved in the process where unnecessary delays are introduced (e.g. change advisory board, internal verification loop, or manual acceptance testing) which might delay the process even with many days [S1] [S9]. Total cycle time of a feature might become significantly longer for example from a few days to weeks. The smaller the change is, the more significant this kind of multiple day delay will become [S1].

There were no mitigation strategies related to the challenge of incompatibility of processes but one research proposal stated that there is a need for a research that would identify these incompatible processes (business, development, operations, etc.) within an organization that hinders the adoption of continuous delivery [S1].

5.1.2 Procedural: Lack of standard practices (C4)

Lack of understanding of the continuous delivery process and standard practices [S5], especially for the novice developers [S2], is seen as a challenge. For some of the teams in a specific case organization there was only some sort of documentation available associated with the continuous delivery process or it did not even exist [S2]. However, some developers felt that there were not so many differences to their original workflow when they were adopting the continuous delivery practices [S2]. Still there is a demand for industry standards for the software development processes regarding the continuous delivery practices.

One mitigation strategy was found for this challenge which advised others who encountered the same challenge to adopt social rules that complement the continuous deployment of software which must be adhered to [S2].

5.1.3 Organizational / Resources: Increased pressure (C9)

When adopting the continuous delivery practices software developers will become more responsible overall, but also on deployments [S2] [S8] [S12]. In addition, the idea that some code could be released to the production environment immediately will increase the pressure for the software developers [S7]. In case there is no manual QA phase the pressure will be increased even more [S2]. Also maintaining the code quality is an increasingly important part of the software developer's daily work if there is no manual quality assurance personnel [S2]. For that the practices and static code analysis tools should be firmly in place.

A software developer's reputation might suffer if she deploys a broken build or some other way impairs the customer's user experience [S7]. This might even affect the business negatively. Also the possibility to decide what to deploy and when might create confusion in co-operation with internal (management, sales, etc.) and external (e.g. customer) stakeholders [S12]. All things considered, the responsibilities and pressures will be increased for the software developer.

Improving communication between developers and managers was one way to mitigate the problem of increased pressure. Another advise was that software developers must be *"diligent in writing tests and monitoring the system"* [S8]. There was also expressed need for proper management program for adopting the continuous deployment process in an organization to ensure that the process is not destructive [S2]. There were no explicit research proposals for this challenge.

5.1.4 Organizational / Marketing and sales: Customer policies (C17)

Releasing software in small increments, in fast pace, and possibly with some experimental mindset may not necessarily meet the expectations of all customers. Large companies may hesitate to buy a software which might not seem to be thoroughly tested because of the risk or fear that there might be increased amount of bugs in production [S2]. Some internal policies may even prevent a company from buying such a "not thoroughly tested" software [S2]. Also compliance with standards (national or international) will pose a challenge for continuous delivery [S7]. Usually such standards require that patient or user safety is guaranteed by not introducing adverse side effects with the changes [S7]. No mitigation strategies or research proposals were inspected regarding this challenge.

5.1.5 Organizational / Customer adoption: Cycle of update (and feedback) (C19)

Not all of the customers welcome continuous delivery type of development method simply because of too frequent updates [S2] [S5] [S7] [S12]. It may be difficult to demonstrate the added value of such a fast pace to all customers, especially if it might affect somehow negatively the customer organization [S2]. Negative effects may arise when the customer organization has to react somehow to the change for example to learn new features or user interface (UI) (especially if the existing UI is already hard to learn [S12]), adjust somehow to changes (organization or processes), do acceptance testing [S12], or to be involved in the installation process [S6]. Especially in the embedded systems domain, deploying a new software may involve numerous activities in customer sites (e.g. updating software or dependencies in multiple places to the same version) and thus be very tedious to customer organizations [S6]. In some cases some actions (e.g. customer acceptance testing) need to be scheduled with the customer which hinders continuous delivery [S12].

Also just the possibility of introducing new bugs may be a big enough reason to be reluctant to update, especially for organizations who experience that the software is working correctly and there is no need for new features at the moment [S6]. Two different advises (mitigation strategies) were interpreted from the data: apply continuous deployment only to non-business critical software [S2] and invest in good UI design [S12]. No research proposals were found.

5.1.6 Organizational / Hierarchy: Different pace between units (C22)

There might be challenges in the collaboration between units (or company functions) if they operate in a different pace. For example development might run on continuous releases while other business units (for example marketing) runs with a six month release cycle [S5]. Business runs usually on *"dates and dollars"* [S8] with annual and quarterly goals and marketing events.

To mitigate the problem of different paces between units, the following strategy was presented [S8]: Use annual roadmap but do not make detailed plans for the future. Plan for current and next quarter but not any further (some candidate initiates can exist). Participate in monthly product meetings and discuss the released and to be released features. Take input on the near-term roadmap. Share the status of the project to other stakeholders. Use historical data to estimate future throughput of the team. No research proposals were found.

5.1.7 Organizational / Culture: Lack of business model (C25)

There is no known business model that suits the continuous delivery type of development [S5]. It is hard to implement continuous delivery practices, as it flows continuously, if business acts in a more conservative way of promoting fixed releases and fixed pricing according to the promised features and usually frozen upfront [S10].

Inspected mitigation strategy for this challenge was to identify lead (pilot) customers with whom you can start to build culture and capability to continuously deliver a software [S10] [S12]. Use lead customers as role models to other customers [S10]. Review the business model to support continuous delivery practices and align all business functions with the R&D unit in order to facilitate continuous delivery [S10]. Word of caution was that getting a pilot customer might be challenging [S10]. In a way, this mitigation strategy can be interpreted also as a research proposal of how to identify a lead customer and how to proceed with such a piloting approach to successfully develop a suitable business model for continuous delivery.

5.1.8 Technical / Testing: Insufficient testing (C34)

There may be challenges in having enough production quality tests, test automation, and maintaining existing automated acceptance testing [S2] [S5] [S6] [S7] [S12]. It will be very laborious to build automated testing suite from the scratch that is sufficient. This might be especially laborious for mature software [S12]. Also there might be challenges in prioritizing the tests that should be created to validate a release [S12]. Testing suite should also be maintained which needs resources [S12]. Automated acceptance testing must be in place in deployment pipeline before deployment to production is possible with continuous delivery [S12].

One suggestion for a mitigation strategy was to invest in testing practices, testing, testing automation, and ensure its thoroughness [S2] [S5]. No research proposals were given.

5.1.9 Technical / Testing: Long acceptance tests (C36)

In embedded systems some factory acceptance testing might take several months [S6] and it is done after the development is completed. This can be considered as a barrier for continuous delivery. However, a test suite that takes hours, for example a test suite run in a web software development [S7], delays instant releases and hence, it is a hindering issue for the continuous delivery practice. For this challenge no mitigation strategies or research proposals were found.

5.1.10 Technical / Testing: Configuration differences (C37)

In software which is installed particularly for each customer, usually there are different environments with different configurations and external dependencies, probably connected with other vendors [S6]. That poses major challenges for automated acceptance testing. These kinds of configuration differences lead to multiple and complex variations which are very difficult to test reliably [S6]. Hence, the testing is usually done with similar configurations. This leads to the problem that there might be many faults to be found in the acceptance testing phase on the customer end [S6]. The same problem occurs also in internal development process if there exists differences between environments (for example development, testing and production environments) [S7]. We did not find any mitigation strategies or research proposals related to this challenge from the data.

5.1.11 Technical / Testing: Automated UI testing (C38)

Web browsers introducing different implementations of standards and mobile game user interfaces with a fragmented device base may pose a challenge for automated acceptance testing [S7]. No mitigation strategies or research proposals were found for this challenge.

5.1.12 Technical / Dependencies: Cross-product dependencies (C45)

If a software product has tightly coupled cross-product or module dependencies it may pose a challenge for deployment process as there is a great need for rigorous integration testing across these products [S2] [S7]. Also there may be challenges if the products or some parts of the products are in a different stage of development [S7]. The same applies for application programming interfaces (API) integrations [S12]. And in addition to the rigorous testing, the API changes must be planned and discussed in advance with stakeholders [S12]. The support for automatically updating the integrations may *"require unduly amount of work considering the results"* [S12].

One mitigation strategy to overcome this challenge was that an organization should provide enough hardware resources to testing infrastructure [S2]. No research proposals were found.

5.1.13 Technical / Deployment: Seamless upgrades (C49)

There may be complications related to seamless upgrades that continuous delivery demands. Extra resources (for multiple instances of software) might be needed to perform zero-downtime upgrades and customer data has to be preserved correctly [S2]. This is also a great problem for embedded systems domain where it is critical to have no downtime and two parallel systems would be inconvenient to use. For example some factories must stop the whole production line for a day or so to run the update process [S6]. This is usually also a big cost for the customer. The same applies for a web software, if seamless upgrades are not achieved, the user has to stop working or might lose data in the progress [S12]. There should be no interferences with the ongoing usage in case of a frequent update pace.

The use of two parallel systems to perform upgrades to shift a user from an old to a new version of the software (known as blue-green deployment technique) was a presented mitigation strategy related to this challenge [S2].

No research proposals were found.

5.1.14 Technical / Deployment: Diversity of client configurations (C50)

Diversity of configurations is also a great challenge for application deployment phase. For example diversity of client network configurations increases the deployment time complexity when deploying new releases of software to network devices [S7]. Diversity of configuration is also a challenge when there is a legacy product to be updated and the configuration is very complex [S9]. Configurations should be updated automatically in the deployment process [S12]. Managing the diversity of configurations is also problematic [S12]. No clear mitigation strategies or research proposals were observed regarding this challenge.

5.2 Reasons Behind Challenges

In order to clarify the reasons behind these challenges, we processed them further to create descriptive themes. We identified 9 different reasons behind the technical challenges and 8 different reasons behind the social challenges. To continue our analysis in order to find out the main reasons behind the perceived challenges, we created higher-order themes (main reasons) and identified five different ones. The identified main reasons were: immaturity, unsuitability, complexity, dependency, and security. The complete list of reasons with higher-order themes is presented in Table 20.

From the extracted data we resolved the distribution of findings to the main reasons (see Table 21). Most of the findings were related to immaturity (39%). Dependency was the second most common main reason (27%). Least number of findings were related to security (7%). The frequency of challenges we observed from these findings was following the frequency of findings pretty well. From the differences in frequency of challenges and frequency of findings we can say that there was slightly more findings behind the main reasons of immaturity and dependency while unsuitability had bit less in contrast to our observed challenges.

Table 20: Reasons behind the challenges

Reason	**Type**	**Challenges**	**Freq.**
Immaturity			
Inability to work in small and fast batches	Social	C1, C2	5
No standardized process plan	Social	C3, C4, C5, C6, C7	6
Lack of motivation and skills	Social	C8, C11	5
Change resistance	Social	C9, C10, C12, C13	7
Unavailability of mature tools	Technical	C26, C28	2
Immature automated testing	Technical	C34, C35, C36, C38, C39, C42, C43	17
Unsuitability			
Organization silos	Social	C20, C21, C22, C24	6
Unsuitable culture	Social	C23, C25	4
Non-modular architecture	Technical	C29, C31	3
Incompatible practices	Technical	C27, C33, C40, C52	5
Complexity			
Complex configuration	Technical	C37, C41, C50	9
Multiple version product	Technical	C32, C59	3
Dependency			
Traditional marketing	Social	C14, C15, C18	3
Non-adaptable customers	Social	C16, C17, C19	11
Legacy systems	Technical	C30, C44, C46, C47, C48	5
Third-party dependencies	Technical	C45, C51, C55, C57, C58	10
Security			
High-level safety, security or availability	Technical	C49, C53, C54, C56	8

5.2.1 Immaturity

Immaturity in software development organization means that the knowledge, practices, or tools are in an insufficient state which will result in unsuccessful adoption of continuous delivery. Immaturity consists of six different reasons, four of them being on the social side (inability to work in small and fast batches, no standardized process plan, lack of motivation and skills, and change resistance) and two of them being on the technical side (unavailability of mature tools and immature automated testing). Immaturity could be addressed by training, education, improvement of current methods, tools, and technologies. Creating industry standards would be one major solution for reducing immaturity. There indeed was an expressed need for standardization of technological aspects in continuous delivery in one of our primary studies [S1].

Overall, most of the challenges we observed were related to immaturity (37%) (see Table 21). As we can see in Table 22 most of the challenges that were immaturity based were in the web software domain (60%) followed by the embedded systems domain (25%). In scientific software (7%), customer specific web software (4%), and mobile software (4%) the immaturity was not so often the reason behind perceived challenges.

Inability to work in small and fast batches. Traditional processes are described as too slow for continuous delivery practice. In traditional processes there is usually stages involved that may take from hours to even weeks to complete by a person or a group of people. These kinds of delays should be eliminated to achieve a faster release cycle that the continuous delivery embraces. However, it is not enough to only adapt processes to the faster release cycle but also the people who plan and implement the features or changes should be adapted. Developers are supposed to be able to split tasks so that the tasks could be done in small and fast batches. To overcome these kinds of challenges, education and suitable technology is needed.

The use of dark features (or dark launches) could help to alleviate the problem of working in small batches when developing a large feature. This type of a release technique allows deployment of features to the production without the appearance of incomplete features to the customer. This kind of mitigation strategy was reported to be in use for this kind of a challenge in one of the case companies [S1]. In the same study the author suggested that even though the concept of working in small batches is not new, there is a lack of research of how the size of the feature affects the deployability of a feature when using continuous delivery [S1].

Table 21: Distribution of main reasons

Reason	**Challenges**	**%**
Immaturity	22	37%
Unsuitability	12	20%
Complexity	5	9%
Dependency	16	27%
Security	4	7%

No standardized process plan. Lack of industry based standard process descriptions which are especially adapted for continuous delivery causes ambiguities among the people who are trying to follow the continuous delivery practices but carry on with the old process models. Also new developers may be a bit lost with continuous delivery practices because of a lack of standards. Even simple things are not clear for all organizations, for example when a build fails, how should the team react: who should fix, what should be fixed and when should it be done? One of our primary studies suggested an advice which could help organizations with process planning: adopt social rules that complement the practices in the continuous deployment which must be adhered to [S2].

However, keeping the build green is one of the principles of continuous delivery approach (because of continuous integration). The strategy for stopping the entire team to concentrate on the fixing issues must be planned and in place to keep the build green. The team should collectively aim for this goal and it should be the primary goal as stated in one of our primary studies [S2].

The team is usually aware of the status of the current project, possibly with the help of continuous integration server and issue/project tracking tools, but one question seems to be that how do we share the status of a project and propagate information about new features across the organization in a continuous basis? This should somehow be taken into account in the whole organization when planning the process model for releasing software more rapidly. Teams should be responsible for accepting continuous flow of changes and also propagating the changes rapidly to other teams [S6].

Also to be successful in continuous delivery the use of feedback data should be efficient. Efficiently using the feedback is not yet understood in all organizations. Including the utilization of feedback to the process model might help the situation. One of our primary studies suggested, that in order to enhance the use of feedback, a strategy to monitor customer behavior through some data analytics platform should be implemented [S2]. Authors stated that with such a strategy, tools, and process it is possible to steer the development of features in a way that might be more inline with customer's expected needs. They also noted that experiments should be as small as possible and made as quickly as possible.

Lack of motivation and skills. In adopting continuous delivery practices there might be problems if the top-level management is not motivated. Also the motivation and skills are important among those whose job is to implement things. Education is needed to improve skills. Experienced resources may be helpful. Top-level management motivation is seen as very important because establishing the deployment pipeline and creating a sufficient level of automated testing might be costly and time consuming. In the worst case, if the management or other personnel have not been clearly instructed about the benefits of continuous delivery there might be even resistance towards it.

An author in one of our primary study noted that you have to make sure that the top-management is implementing a strategy to push the need of continuous delivery all the way through the lower organization units and setting a goal for achieving continuous delivery [S2]. They also noted that it is also important to ensure that there is a low or nonexistent learning curve for team members in adopting continuous delivery practices when introducing it to the existing systems and process, for example when integrating the new practices to the old workflow [S2].

Change resistance. Fear of unemployment because of increasing automation, increased pressure towards software developers, and new roles (or decreasing specialization) as continuous delivery embraces cross-functional teams are issues that organizations might end up struggling with. However, some of the reasons behind the resistance might be irrelevant. For example the fear of unemployment because of increasing the amount of automation in quality assurance (instead of manual testing) might be an unnecessary fear. As noted in one of the primary studies, planning good automated tests requires a professional QA mindset [S8]. They also continued that planning requires hours of reading stories, performing research into the background of the story and creating different angles to test cases and if there is some time left, QA could invest in monitoring and analyzing the system.

One suggestion was that shifting from personal to team thinking would help to decrease the pressure towards single developer and thus reduce change resistance. So, take responsibility as a team of delivering the software to customers without negative effects to the customers [S2]. Also improving communication between developers and managers is said to reduce the increased pressure towards developers [S2].

One mentioned challenge related to cross-functional teams was that in some cases there is a need for deep specialization (for example in the level close to the hardware in embedded systems domain). In these kinds of situations it is good to ensure that there is effective communication taking place between modules and that CI practices are implemented and work as a coordinating mechanism both within and across the teams, as suggested in the study considering challenges in adopting DevOps in embedded systems domain [S6].

Unavailability of mature tools. According to our data there is no comprehensive, robust, and open (no vendor lock-in) continuous delivery platform available. Creating a custom solution for a continuous delivery platform requires multiple different tools, lots of customization, and development of available tools. Especially some special fields may be missing solutions for their needs, for example automation of infrastructure for software where hardware optimization is required [S3].

One way to overcome this is to develop own continuous delivery platform solution which might be costly as in one case company [S1]. They also noted that building such a comprehensive platform might involve many different tools and technologies and because of that avoiding vendor lock-in could be challenging.

Immature automated testing. To move towards continuous delivery of software a high level of test automation is needed. If an organization is undeveloped (i.e. is immature) in this area, the adoption of continuous delivery will be challenging and time consuming (especially for large and mature software). Not only the development of testing suite but also the management of and running the tests will be resource intensive. Also if there is not enough expertise in the area of testing there will be problems as continuous delivery relies heavily on automation; you need to know exactly how to test and what are the expected results. Sometimes acceptance testing is not advanced enough to run fast enough (for example in web software development). Sometimes different browser engine implementations or device fragmentation introduces challenges to test automation.

Improvement of testing suite may be hindered also because of domain constraints. They might even set barriers to test automation. In some cases the acceptance testing phase just takes too long (i.e. embedded systems in factory automation). Embedded systems consist of software and a device which might introduce problems in performance testing; it might be hard to implement testing infrastructure with enormous amount of devices (barrier to test automation).

There might also be challenges when software developers are not diligent enough to follow the practices. Ignoring test suite results (for example in case of flaky tests [S8]) might ruin the goal of continuous delivery which is *"delivery of working software faster"* [HF10]. Diligent testing practices must be adopted to ensure that testing is thorough since deploying software is dependent on tests passing on the continuous integration server [S2]. Tests, gates, and checks are there for a reason [S8]. So there should be investments to enhance testing: testing practices, testing automation, hardware resources, and ensuring the thoroughness of testing [S2] [S5].

Table 22: Main reason distribution across domains

Main reason	W	C	E	S	M
Immaturity	60%	4%	25%	7%	4%
Unsuitability	54%	15%	31%	0%	0%
Complexity	16%	50%	33%	0%	0%
Dependency	50%	11%	22%	11%	6%
Security	25%	25%	50%	0%	0%

5.2.2 Unsuitability

By unsuitability we mean cultural, practical, and structural aspects of the software development organization and software that are not suitable when adopting continuous delivery practices. Reasons for unsuitability behind the social challenges are organization silos and unsuitable culture. Fast and continuous pace that continuous delivery builds upon requires seamless collaboration and experimental mindset without too far into the future predicted plans that are unchangeable. This is a challenge especially for the business (i.e. traditional business models) but also for the technological parts of software. Non-modular architecture and incompatible practices that are impractical or problematic to automate are the reasons behind the challenges in the technical side for these kinds of challenges. To overcome the problem of unsuitability major changes must be made which will span across the organization.

Unsuitability was the third most frequent (20%) reason behind the perceived challenges (see Table 21). As we can see in Table 22 the distribution of challenges in the main reason of unsuitability across the domain was following the distribution of immaturity regarding the web software (54%), customer specific web software (15%) and embedded systems domain (31%) but the distribution did not have so much variation. However, there were no observed challenges in scientific or mobile software domain that were related to unsuitability.

Organization silos. According to one of our primary studies [S2] in order to fully succeed in continuous delivery and deployment it requires involvement of multiple organizational units with the common goal for continuous delivery. This might pose a challenge especially if there exists barriers between units. If the structure of organization is hierarchical this might add tension between the units. One advice was that the organization should restructure itself in order to break down barriers and promote a collaborative culture [S1].

Organization silos will also pose a challenge in sharing the current status of projects across the organization. Typical silos are between the following three units: research and development, marketing and sales, and the management. However, silos can also be formed inside a company function for example between different research and development units. Also if the units operate in a different pace this might cause problems in collaboration.

One study [S8] proposed a mitigation strategy to this problem where they encouraged the product team to participate in monthly product meetings to discuss released and to be released features, to take input on the near-term roadmap, and to share the status of the project to other stakeholders. To enable this they also use an annual roadmap but not make any detailed plans for the future, only for the current and next quarter. According to one primary study also the deployments should be more carefully discussed with related stakeholders [S2].

Unsuitable culture. If a company has a strong tradition of a conservative way of doing things, for example rigorous testing and validation processes, there might be resistance towards more experimental mindset of continuous delivery where functionality may be released although it is not fully completed. This is also problematic for businesses as they are lacking suitable business models for this kind of progression. Traditionally, business promotes fixed releases and fixed pricing according to promised features that are already available.

For these experiments that are made in run-time infrastructure, tools, data collection, and early feedback collection must be in place. These functions usually require some sort of architectural modifications if they do not exist already according to a study about moving towards R&D as a innovation experiment system [S10].

Also one suggestion was that in case a business is tackling the mindset and business model problems, an identification of lead (pilot) customers, with whom you can start to build a culture and capability for continuously delivering a software, might help [S10]. With these pilot customers you can review the business model to support continuous delivery practices and align all business functions with the R&D organization. Also these pilot customers can be used as role models to other customers.

Non-modular architecture. A large monolithic application that is not amenable to continuous delivery practice might pose a challenge for adoption of continuous delivery practices because of the problems in automating the delivery of software in such an architecture. According to one of our studies [S1] there is a huge number of non-amenable applications in the industry. Also, typical for such applications, a huge code base and high complexity might pose a challenge to continuous delivery as it might take too much time to build and compile the code, and create a package, to deliver software in a fast pace.

One suggestion was to transform architecture towards more modular architecture which allows upgrading the system in smaller and independent parts or improve capability for continuous delivery some other way [S13]. Of course this must be justified as the value for this kind of a change is not as evident as with other new features.

Incompatible practices. Incompatible practices also cause problems in the technical side. Branching in the VCS, updating version logs, and rolling back deployments to previous version are examples of such questioned practices. Continuous delivery promotes that there should be only one branch of the software in the version control system which is under development. Some companies are used to developing multiple branches at the same time for example large and long-lasting feature branches. This might pose a challenge for companies with such practices. Smaller and short-living feature branches are preferred [S2].

Also because of smaller and faster releases it might become as a surprise that managing the version log requires more effort. There is no point in discussing every release in detail with every customer. Fast pace in moving forward will also question the strategy to rollback a release [S11]. Is it a sufficient strategy when moving rapidly towards more frequent releases?

In some companies there are manual or explorative testing phases before the release. There might be support for such a testing because quality assurance testers might find bugs that were not caught by the automatic testing suite. This might hinder the rapid releasing of software that continuous delivery is aiming at.

5.2.3 Complexity

Complexity was the main reason for the challenges that were related to the configuration of the software products. This was especially problematic in the products that had multiple versions available and in software that was very complex to configure because of environment differences. In our analysis the complexity was related only to the technical side of challenges. Complexity caused problems especially in software testing automation and deployment of the software. Avoiding decisions that would increase the complexity would be recommended. Harmonization of software environments would help to overcome the problem of complexity.

Challenges that originated from complexity were not so common. Only 9% (second least) of the challenges were complexity based as we can see in Table 21. However, half of the complexity based challenges were found from customer specific web software domain (see Table 22). Also in the embedded systems domain, the complexity was the major reason behind the challenges (33%) while web software domain had only some (16%). There were no challenges that were complexity based in scientific or mobile software domain.

Complex configuration. Configuration differences caused challenges in testing and deployment phases but also increased the burden in sharing information between developers (managing diversity of configurations). If there is multiple different environments with multiple different configurations there will be multiple complex configuration variations that will be hard to test completely and thus will cause bugs to slip past the testing phase. A typical example is differences in testing/development and production environment. Another example is products that have multiple versions available (for example multiple user specific versions) with different configurations. Deployment time complexity was reported to increase with the configuration variations. One example was diversity of network client configurations when deploying new software to network devices [S7]. Configurations should be updated during the automatic upgrade of software (i.e. in the deployment phase).

Multiple version product. Multiple version products caused problems especially with the configuration variety but also with the documentation. There were reported challenges in documentation when there is a hosted version and customer specific versions out, the documentation is probably not identical. One mitigation strategy for documentation challenges was to use a wiki-type of tool to maintain it [S2].

Sometimes the ability to gather only the needed modules for specific users to use will produce multiple different variations of the software which may bring challenges to the completeness of testing.

5.2.4 Dependency

By dependency we mean dependency of the software system, process, or organization on some internal or external stakeholder in order to work correctly. From a social point of view traditional marketing and non-adaptable customers are the reasons for the challenges in this theme. Problems from the technical point of view are the legacy systems and third-party dependencies. These kinds of problems are often not easily removable. Our suggestion to cope with the problems based on dependency is to increase the level of communication and openness.

Dependency based challenges were the second most common (27%) after the immaturity (37%) as we can see in Table 21. There were challenges observed from every domain regarding the dependency. Usually these kinds of challenges were found from web software (50%) or embedded systems domain (22%) (see Table 22). On rare occasions these kinds of challenges were found from mobile software domain (6%) while both customer specific web software and scientific software domains had relatively low percentage of challenges (11%).

Traditional marketing. Traditional marketing strategies are not suitable for software that is developed in the continuous delivery approach as there may not be a clear version which can be promoted. As such it may pose a challenge in the relationship between marketing and development units as there is an uncertainty regarding when the expected features are ready. One mitigation strategy was to increase the level of transparency between the development team and sales and marketing unit [S8].

However, when you are releasing these small and possibly experimental features there should be a way to promote those changes to the users as soon as they are released. In this way users can find new features and produce usage data (and feedback) that can be gathered to help the direction of development. One advice for such rapid discovery of new features was to create a blog which is available to customers where you can post about product's changes and present new features [S2].

Non-adaptable customers. From a customer's point of view it is not easy to buy software where the features are not clearly stated and there might be intentions for experimentation. This, accompanied with the customer's assumption that the rapidly changing software is not thoroughly tested (the fear of introducing bugs) will challenge business even more. Also it might not be easy to demonstrate the added value of rapidly changing software to the customers especially if the updating requires some actions from them (learning and adopting the introduced changes or even installation). Because of this, there was an advice to invest in good UI design [S12].

The most challenging issue is the compliance with standards which require certain steps in the release process that are not possible in the continuous delivery approach. These kinds of policies are barriers for starting to use a software that is developed in the continuous delivery way. One simple mitigation strategy especially for continuous deployment was to apply continuous deployment only to non-business critical software [S2]. A more forward looking suggestion was the development of business and pricing models that support this kind of short-cycle innovation with the help of customer usage data [S10].

Legacy systems. Legacy and outdated systems are causing challenges in adopting continuous delivery practices. Usually these are related to the automation of software delivery. Legacy integrations, outdated operating systems with missing libraries, and PC based infrastructure are holding back the automation attempts. If some libraries are missing, create custom packages and set them available for private repository [S3]. It would be better to update an old PC based infrastructure to a modern virtualized cloud environment [S5]. Virtualization and containerization would also help to test binary compatibility issues [S3].

Some customers may have strict hardware or environment policies which may lead to outdated operating systems. According to one of our primary studies [S6] especially in the embedded systems domain the legacy code dependencies will cause compatibility issues in long life-cycle software products.

Third-party dependencies. Dependencies on third-parties are problematic as they cause compatibility issues if not properly handled. Integrations to third-party APIs, components, and plugins will require extra amount of care to successfully maintain the status of fully working software across releases. One advice was that there should be enough hardware resources provided for testing and that sufficient amount of testing around database and third-party plugins should be ensured [S2]. In case there are too many plugins to manage, the amount of maintained and compatible plugins could be reduced [S2].

There are also domains that require a third-party to be involved in the process of deployment of software such as mobile software domain. Application stores usually have some kind of a verifying process for the software to be released that will take from few days to weeks to complete. The same goes for software that is supposed to be installed to customers' environment but there is also a customer-end quality assurance phase. Sometimes customer environments are also inaccessible from the software development team point of view which will pose a challenge for debugging and monitoring software which is a problem for continuously learning from usage data and deliver new functionality.

5.2.5 Security

This main reason contains safety, high-level availability and confidentiality as sub reasons behind the challenges. These kinds of challenges were only on technical side according to our synthesis. The access restrictions to some systems and the seamless upgrading of software without losing user data or reducing availability are examples of these kinds of challenges. Security based challenges are not easy to overcome and they may require some kinds of trade-offs. It may be that the solutions to security originated challenges will increase the amount of complexity.

Security was least often the reason behind the challenges (only 4%) (see Table 21). Half of the challenges based from security were found from embedded systems domain (see Table 22) while the rest were evenly distributed between web software and customer specific web software domains (25% each). No security based challenges were found from scientific or mobile software domains.

High level of safety, security or availability. If there is a requirement that users should be able to use the system with extremely high availability and releases are made in a fast and continuous pace there is a requirement for seamless upgrades. Implementation of seamless upgrades are not so trivial and the difficulty depends on the type of software under development. For example in embedded systems it may not be feasible to duplicate the system in order to achieve seamless upgrades in contrast to web software development where this kind of blue-green deployment technique may be used [S2].

Sometimes if a seamless upgrade fails, it might even be dangerous for users (e.g. medical and health systems) and in some cases even the business may be damaged if some interruptions occur. In these critical domains there is also a fear for decrease in quality and thus a fear of risking user's safety or business. In turn, in case of a not so critical domain the potential increase of bugs in the production environment was reported to be overridden by the benefits of continuous delivery when the bugs are immediately fixed or rollback is made to the previous working build.

Also high level of security might prevent continuous delivery to be successful. Access restrictions may be blocking development and debugging of systems.

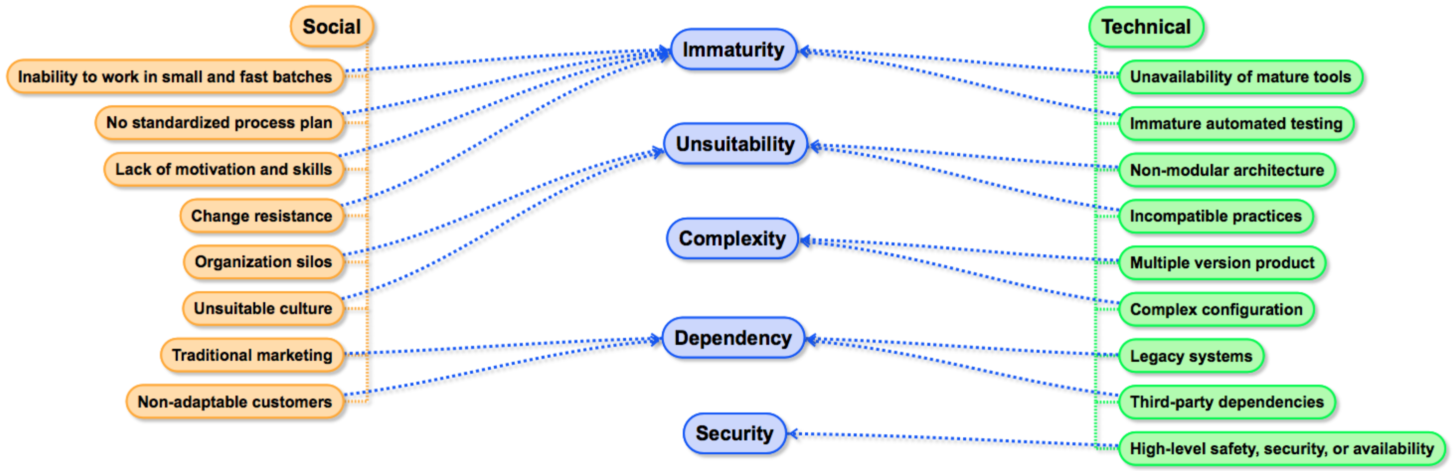


Figure 7: Reasons behind challenges

5.3 Domain Analysis

As noted in the introduction of challenge analysis, we could not find any reliable way to figure out what were the specific challenges for some specific domain but now that we have analyzed the main reasons behind the challenges, we can take a look at how the main reasons were distributed in a domain (see Table 23). However, it is good to notice that especially considering the scientific and mobile software domains the amount of cases were very low. Because of the low amount of data for scientific and mobile software domains we excluded them from the summary of domain analysis (section 5.3.5).

5.3.1 Web Software

The main reason that was clearly above others in web software domain was immaturity. Almost half (49%) of all the perceived challenges in the web software domain were related to immaturity. Dependency (26%) and unsuitability (20%) were the main reasons behind almost all of the rest of the challenges. Minority of challenges were related to complexity (3%) and security (3%).

Table 23: Distribution of main reasons inside a domain

Main reason	**W**	**C**	**E**	**S**	**M**
Immaturity	49%	11%	35%	50%	50%
Unsuitability	20%	22%	20%	0%	0%
Complexity	3%	33%	10%	0%	0%
Dependency	26%	22%	20%	50%	50%
Security	3%	11%	10%	0%	0%

5.3.2 Customer Specific Web Software

In the customer specific web software domain the complexity was the most common reason behind the challenges (33%) followed by unsuitability and dependency (both 22%). Immaturity and security were the least common main reason behind the challenges in this domain (both 11%).

5.3.3 Embedded Systems

In embedded systems domain the main reason behind the challenges that was most common (35%) was immaturity. Unsuitability and dependency were the second most common main reasons behind the challenges (both 20%). Also complexity and security based challenges were present in some cases (both 10%).

5.3.4 Scientific And Mobile Software

Immaturity (50%) and complexity (50%) were the reasons behind the perceived challenges in both scientific and mobile software domain. No challenges that were based on unsuitability, complexity or security were found. However, the amount of cases were negligible regarding these two domains (one case per domain).

5.3.5 Summary Of Domain Analysis

Most of the variation between the main reason distribution in domains (web software, customer specific web software, and embedded systems) was among immaturity, complexity, and security. In web software domain, the immaturity based challenges were emphasized while complexity and security based challenges were not. In turn in customer specific web software domain the complexity was the most common main reason behind the challenges. In embedded systems domain the immaturity was again the most emphasized but not so much as in web software domain. Also in embedded systems domain the complexity and security were more often the main reason behind the challenge.

The occurrence of challenges that were based on unsuitability and dependency were quite similar between the web software, customer specific web software, and embedded systems domains. It varied from 20% to 26% so we could say that unsuitability and dependency are common reasons behind the challenges for all domains and are occurring on a quite similar frequency. One thing to notice is that security based challenges were less perceived in every domain.

5.4 Proposed Research Areas

Clear research proposals to observed challenges were found only from a few different studies. Some of the research proposals were considering a much higher level of abstraction than a single challenge so we decided to present them all here in their own section.

Process of adopting continuous delivery. There was an expressed need for research that would identify existing incompatible processes (business, development, operations, etc.) within an organization that hinder the adoption of continuous delivery practices [S1]. After the identification, the existing processes should be developed furthermore and alternative options should be verified to suit continuous delivery [S1]. Also regarding the process of adopting continuous delivery practices, there was an expressed need for proper management program in an organization to ensure that the adopting process is not destructive for it [S2]. From a customer point of view, development of an engagement model with lead customers (i.e. pilot customer) to facilitate continuous deployment would be needed [S9]. An engagement model could also help with the expressed need for development of business models and pricing models that support short-cycle innovation processes [S10]. Innovation is based on customer usage data and thus, customers are needed.

Understanding and developing strategies to overcome challenges. Further research is also needed to understand the challenges organizations face when adopting the continuous delivery in more depth and to develop strategies and practices to tackle them more efficiently [S1]. A few more specific research topics were already presented regarding observed challenges such as: how the size of some feature affects the deployability of a feature when using continuous delivery [S1], need for widely accepted standards, open APIs, and active plugin ecosystems for existing or future continuous delivery platforms [S1], and understanding unsuitable architecture characteristics and identifying and developing the best strategies or practices to tackle them [S1].

6 Discussion

In this chapter we discuss the results, analysis, and validity of our study. First, we compare our results to related works and after that we discuss the validity of our study regarding the different parts of the research.

6.1 Related Work

In a systematic mapping study of continuous deployment of software intensive products and services [RHL+16] the found challenges were quite similar to our findings. However, only four of our primary studies were also included in the mapping study of Rodriguez et al. The particular study was focusing mainly on analyzing the frequency of findings however, they also noticed that there were difficulties adopting continuous deployment especially in the embedded domain.

Transforming organization, culture, and personnel mindset towards continuous deployment was considered as a challenge. We also observed the same difficulties and also analyzed that these types of challenges existed mainly because of immaturity and unsuitability reasons. These challenges were mainly on the social side in our categorization which supports the observation made in the mapping study that *"...human factors, including personality and cognitive aspects, play a fundamental role in truly achieving continuous delivery"* [RHL+16].

They also observed that even though customers are more satisfied there were reports of challenges in customer adoption of the continuous deployment approach. Customers were reported to be reluctant towards new releases because of poor quality of releases and a learning curve of a new functionality. Also privacy and security issues were concerned related to monitoring (in customer environments) and usage data collection. All these issues were also observed from our primary studies. The customers' adoption of fast cycle of update and feedback (C19) was also one of the common and frequently observed challenge.

One observation was that QA efforts were increased due to the difficulties in management of test automation infrastructure [RHL+16]. This is in line with our findings that there was insufficient level of automated testing (C34) which was one of the most frequently observed challenge and a common one among the domains.

Rodriguez et al. reported that the embedded systems domain was problematic regarding the adoption of the continuous deployment practice. Some domain constraints such as hardware equipment and physical assets were causing challenges for the automation process. Architecture, resources, and security (safety, security, and privacy) were mentioned as sources for the challenges. We also collected the challenges involved in adopting continuous delivery practices in embedded systems domain and similar things were observed. We also noticed that security based challenges were mainly found from the embedded systems domain (along with customer specific web software domain).

As we also observed, Rodriquez et al. reported that there were difficulties in managing various configurations and run-time environments. We also observed that these kinds of complexity based challenges were mostly occurring in customer specific web software and embedded systems domains. Also, lack of trust in quality was mentioned as one of the less frequently occurred challenges and we also observed the same challenge but interpreted it to be more of a problem of sharing the status of current build across an organization and stakeholders (C24: Sharing of status). In our study, this challenge was also a rare observation. This challenge type was observed in two cases in embedded systems domain, from two different papers ([S9] and [S10]) but the authors were the same. However, there was also one observation of the fear about product quality (C56: Product quality may decrease) but that was more of a fear than lack of trust. This was also quite rare as we observed it only from one case.

Rodriquez et al. reported other less frequently occurred challenges that we did not observe such as: lack of trust in quality, natural tensions between the desire to deliver functionalities quickly and the need for reliable products, release planning and managing the roadmap, and risks associated with gathering user feedback from a limited population [RHL+16].

Overall, especially the most frequent type of challenges in Rodriguez et al. paper (the process of transformation towards continuous deployment, customers' unwillingness to receive continuous product updates, increased quality assurance (QA) efforts, and difficulties applying CD in the embedded domain) were also observed in our study. Customers' unwillingness to receive continuous product updates and increased quality assurance (QA) efforts were also observed as common challenges in our study. Customers' unwillingness to receive continuous product updates was a similar finding to our challenge of customers' adoption of fast cycle of update (and feedback) (C19). Increased quality assurance efforts was related to our observed challenge of insufficient level of automated testing (C34). Also difficulties applying CD in the embedded domain were present in our observations, but of course our primary studies were mainly considering web software and embedded systems domains. After all, both of these studies confirm the existence of these challenges as significant ones, especially because we had only four same primary study papers.

A semi-systematic literature review and a case study on rapid releases and software testing by Mäntylä et al. [MAK$^+$14] observed from the literature that there are problems with high reliability and lack of high test coverage, automation cannot solve all challenges related to reliability and test coverage, customers might get weary of yet another update, and the technical quality of the product may decrease (increase in technical debt).

We also observed the problems with high reliability and the lack of high test coverage but there were no clear indications that automation cannot solve all challenges related to reliability and test coverage, except in the embedded systems domain where some domain constraints like inability to do blue-green deployments in factory settings prevents such things. So, to some extent, we agree with the fact that automation cannot solve all problems with reliability and test coverage. One major and a common challenge we observed was the customers' adoption of fast cycle of update and feedback which is in line with the finding of Mäntylä et al. that customers might get weary of yet another update.

Concerns regarding the increase in technical debt because of having less time available for each release, presented in the literature according to Mäntylä et al., was not observed in our study. The same observation was made by Mäntylä et al. in their case study reflection. However, we did have one observation about product quality decrease (C56: Product quality may decrease) but that was considering more the increasing amount of bugs, not technical debt.

Only two of the primary studies in the paper by Mäntylä et al. were in common with our primary studies. We can confirm the findings in the study by Mäntylä et al. especially relating to the problems with high reliability (C49: Seamless upgrades), lack of high test coverage (C34: Insufficient testing) and customers getting weary of yet another update (C19: Cycle of update and feedback). We can partially confirm the observation from Mäntylä et al. that automation cannot solve all problems related to reliability and high test coverage, especially in the embedded systems domain because there are some domain specific constraints that might block the automation. For the last issue Mäntylä et al. observed, the increase in technical debt, we did not find any support.

In summary considering these three studies, the most frequently perceived common challenges regarding the adoption of continuous delivery practices in software development projects are insufficient level of automated testing and customers' adoption of fast cycle of update and feedback.

6.2 Data Collection

Our main interest was to find out the challenges in adoption of continuous delivery practices in software projects and to create a synthesis about the data. In our opinion, to gather data for such a broad synthesis, viable options are either a multiple case study or a systematic literature review. A multiple case study would be a primary study type of a paper while systematic literature review would be a secondary study a type of paper [PFMM08]. Because multiple case study, especially a comprehensive one, would require a large number of cases we decided to create a systematic literature review.

The fact that there are multiple primary studies instead of one multiple case study reduces the bias the authors might introduce (either accidentally or intentionally) which is also a good thing for systematic literature reviews. Of course, the fact that systematic literature reviews make use of other authors' work will introduce challenges in the interpretation of the data. To ensure the possibility to interpret the findings in a correct way, we collected not only the challenge description itself but also the category, mitigation strategy and research proposals for every challenge if they were available.

A challenge for us was to define the scope of the systematic literature review considering the continuous delivery approach. Albeit we defined our scope to be broad, there are still examples of even broader scopes related to continuous delivery and related practices. For example, systematic mapping study of continuous deployment [RHL$^+$16] and a semi-systematic literature review considering rapid releasing of software [MAK$^+$14], both had much broader variation of search terms compared to ours.

Based on the distribution of the amount of papers published across the years (see Figure 4) we can say that the limitation by the publication year of the study (published in 2010 or later) was to some extent a successful decision as most of the papers were published in the year 2015. We can imagine that more papers will be expected related to the area of this study in the following years. One could argue that there are some earlier papers missed due to the limitation but we wanted that the primary studies' authors and organizations would have a clear understanding about the continuous delivery and related approaches. As we previously expressed the different interpretations of continuous delivery and related terms in section 2.1 this was not an unnecessary concern. This was one way for us to ensure that the authors of primary studies understood the concept, though maybe not a perfect one.

The amount of primary studies was quite low which may affect the validity of our results and analysis. Also a few papers had same authors which may suggest that there might be some duplicate findings or some biases. Papers S5 and S6 were written by almost the same authors. The same applies to papers S9, S10, and S11. Considering these primary studies there was a possibility to observe the same challenge multiple times.

One problem for our data collection was that it was not always easy to identify if a suggestion to overcome a challenge was a mitigation strategy or research proposal. However, this study presents both, mitigation strategies and research proposals, to the reader so there is a possibility to consider them further.

6.3 Quality of Primary Studies

To answer our second main research question, we provide an insight to the quality of primary studies (RQ2.1). Quality scores for each primary study is available at Table 11 and in more detail in Table 12. We did evaluate the quality of our primary studies with the help of quality checklist questions (see Table 7) which resulted in providing a quality score for every primary study. These scores were used in the analysis of common challenges in section 5.1.

We combined the evidence level of the primary study with the observed challenge. The evidence level of a challenge was based on the primary study (where the challenge was observed) with the highest evidence level. Almost all of the challenges associated with medium or low evidence level got a frequency of one and did not span across different domains. Based on that we could say that it has at least in some respect fulfilled its purpose of evaluating the evidence for our findings. In other words, all of the challenges that spanned more than one domain did have a high evidence level.

Considering the quality checklist questions the first one got the lowest points which was the one questioning if the primary study was based on research or if it merely was a "lessons learned" report based on expert opinion. We had quite a strict evaluation that gave the action research type of papers zero points while case studies got only half a point. This was based on a decision that to earn full points the research should be based on a controlled environment. However, in software engineering (and related) research empirical evidence may be scarce [Kit04] considering the systematic literature reviews and possible primary studies.

Based on the sum of our last three quality score questions (see Table 7) we may say that the threats to validity of the research is the most weakest point among the studies in our primary study pool. Those questions were considering the relationship between the researchers and the participants, credibility of findings, and the limitations of the study. Especially in the action research type of studies these things were hardly discussed which might be natural to those types of papers.

6.4 Thematic Synthesis

Thematic synthesis is an approach which is used to identify recurring themes or issues from multiple studies [CD11]. It is based on thematic analysis which is traditionally used to analyze data in primary studies. The aim for both is to interpret and explain these themes, and draw conclusions. As described in method recommendations of thematic synthesis [TH08] [CD11] we first did the label coding, then created descriptive themes, and finally went beyond the data with analytical themes (or higher-order interpretations). Weaknesses for thematic synthesis and thematic analysis is said to be the uncertainty of how the synthesis is done [DWAJ$^+$05] [SOV12]. One reason for this is that the synthesis process may be either driven by data or by theory [SOV12].

We were trying to approach the synthesis fully by relying only on data, but of course we were under the influence of our data which had already some theories applied. Hence, it is difficult to describe the approach in a reliable way. We state that the intention for our synthesis approach was to be fully based only on data in every phase. However, especially in the interpretation of higher-order themes, the original data was less emphasized and our own created labels were the primary sources of our analysis. So, the further we proceeded in our synthesis, the further we went from the original data. We also provided the reader with the original findings (see Appendix D) and our observed challenges (i.e. labels, see Appendix B) to allow the examination of our chain of deduction.

Issue that may pose a threat to our synthesis process is that the context from different cases were a bit differing (software and business domains). However, we did observe them in our data collection phase and later on in the synthesis phase. Also the understanding of the key terms was not always clear, or yet similar between studies, so there might be some obscurity in challenges especially related to terms continuous delivery and continuous deployment. However, as we noted in the continuous delivery chapter (see section 2.1) the difference is not so remarkable.

6.5 Perceived Challenges

In our analysis we defined a challenge to be a common one when the challenge was observed from more than one domain. The problem is that because some of the challenge occurrences did not have domain information, we might have missed some common challenges. This was also one of the reasons why the domain specific challenges were difficult to find. Also another issue was that in some challenges there were multiple findings from the same case that we ended up categorizing them in a same challenge and hence, there was higher number of occurrences in that particular domain. A good example of this is the challenge C19 (cycle of update) which had findings 102, 103, 106, and 107 that were collected from the same case CS27.

There was also a problem with web software and customer specific web software domains. We had two cases which were considering web software domain and customer specific web software domain. Every time a challenge was found from this kind of a case we had to interpret to which domain the challenge belonged to. For example in challenge C19 (cycle of update) the findings 102, 103, 106, and 107 were considering a case which domain was defined as "web software / customer specific" and three of those findings were interpreted as originating from web software domain and one as a customer specific web software domain. Practically, the customer specific web software domain cases were similar to web software domain and embedded systems domain cases. Accordingly, if the challenge was identified as originating from the customer specific web software domain and there were more than one occurrences (with known domain), the challenge was usually also found from web software or embedded systems domain (or both).

Interestingly, even though a larger part of the challenges observed were in the technical side of our categorization (34 of 59), half of the common challenges were actually considering the social aspects (procedural or organizational). So, we can strengthen the observation made in continuous deployment systematic mapping study that human factors play a significant role in achieving continuous delivery [RHL$^+$16].

6.6 Challenge Reasons

Security based challenges were least observed challenges among our primary studies. Nevertheless, we are afraid of drawing a conclusion that it would be so. Maybe such challenges are not encountered because continuous delivery is not achieved at a sufficient level yet (maybe because of some other type of challenges). Or maybe these kinds of issues are not so openly discussed because after all, security based problems, and especially the announcement of such issues, could probably make an organization vulnerable.

Complexity was the main reason behind the challenges in customer specific web software domain. Maybe that was because in the two cases that were identified as a such domain (along with the web software domain) these kinds of challenges were often mentioned to occur due to customer specific web software. This does not mean that we disagree with the results but just to note that in our analysis the amount of other types of challenges may be underestimated in customer specific web software domain. Hence, some of the web software domain challenges are probably challenges also in the case of customer specific web software domain.

7 Conclusion

Our goal in this study was to figure out the challenges organizations encounter when adopting continuous delivery and to provide a broad view to analyze commonalities and differences between them. The motivation for this was that with the help of these perceived challenges we could better understand what are the barriers in moving towards the continuous delivery approach and thus, create a foundation for future research to tackle these problems (another goal). We were aware of primary studies that were addressing the challenges (for example [OAB12]) but noticed a lack of more comprehensive broad research related to the challenges in adopting continuous delivery (and related approaches). Hence, we decided to contribute to the knowledge by collecting and analyzing these perceived challenges from multiple different studies. To achieve our goal we made a systematic literature review that, unlike other similar studies [MAK$^+$14] [RHL$^+$16], focused only to the challenges. We applied thematic synthesis approach to figure out the challenges and analyze them further.

7.1 Contribution

In our study we observed various challenges from multiple cases taking into account the software domain where the challenge occurred and the frequency of analyzed challenge. Nearly sixty different challenges were identified and categorized as a social or technical issues. Social challenges were either related to procedural or organizational issues. With the help of software domains, frequencies, and evidence levels (which were based on our quality analysis) we analyzed which of the challenges were more common among the cases. We also evaluated the quality of our primary studies based on our quality checklist. Most of the papers had a high level of quality. However, in our quality analysis we noticed that the validity of the research was the weakest point among our primary studies. The weakest points were given to action research type of studies. However, we want to point out that the approach set by the quality checklist may have resulted in lower scores for the action research type of studies compared with the case study type of studies.

If a challenge was observed in many different domains and got a high evidence level, we defined it as a common challenge. With this definition we found 14 common challenges that were evenly distributed to social and technical sides. Four of the common challenges had a high frequency: cycle of update (and feedback) (C19), insufficient testing (C34), seamless upgrades (C49), and diversity of client configurations (C50). Of these, cycle of update was the only challenge that was considered as a social (organizational) issue. So most of the common challenges that had a high frequency were considering technical issues. With the help of these categorized challenges we also tried to find out if there could have been some evidence about specific challenges for example to certain domain (or regarding the organization size) but failed to find any. However, after we analyzed the reasons behind the challenges we did find some signs that the software domain might affect the challenges that might be faced.

As we continued our thematic synthesis process even further, we identified 17 different reasons behind the challenges. These themes were still processed further and resulted in five different main reasons (high-order themes). The main reasons behind the challenges according to our analysis were immaturity, unsuitability, complexity, dependency, and security. Most of the challenges were related to the immaturity followed by the dependency based challenges. With the help of these main reasons, we analyzed how the challenges were distributed across the domains in the domain analysis. The domain analysis was mostly considering only three of our inspected domains (web software, customer specific web software, and embedded systems) because the last two domains (mobile software, scientific software) did not have enough findings to provide any reasonable results. As a result, the challenges that were based on unsuitability and dependency were observed in a quite similar way from all of the domains, so we could say that in that way the domains had something in common. However, they were never the most emphasized reasons behind the challenges. Complexity was emphasized in customer specific web software domain. In web software domain and embedded systems domain the immaturity was the main reason behind the challenges.

With these results we fulfilled our first main goal of this study which was aiming to find what are the perceived challenges of adopting continuous delivery practices in a software development project and why (RQ1).

7.2 Future Research

Another main goal (or research question) in our study considering the challenges in adopting the continuous delivery approach in software development projects was to provide new directions for future research with the help of observed mitigation strategies and research proposals (RQ2). Mitigation strategies were provided for over sixty percentages of the challenges but only a few different research proposals were collected from the primary studies (see section 5.4). Hence, there should be lots of research topics still to be discovered. We propose some follow-up questions to some of the mitigation strategies and provide our view for future research.

One of our primary studies suggested that the adoption of social rules that complement the continuous deployment of software (which must be adhered to) could be helpful when developing process standards [S2]. These social rules should be identified. After the identification of social rules, proposals for process enhancements or new processes can be made. This could probably be combined with the research proposal which stated the need for process models that suits continuous delivery [S1].

A need for development of business and pricing models that support short-cycle innovation (with the help of customer usage data) was also presented [S10]. For that, existing business and pricing models should be reviewed and improved to better suit continuous delivery practices. Probably also new business and pricing models could be developed.

There were very few mitigation strategies or research proposals for complexity based problems. Especially the complex configuration was identified as a reason for challenges which did not have any mitigation strategies or research proposals. The reasons for such complex configurations should be studied and the strategies (if any) to reduce the complexity to facilitate the adoption of continuous delivery practices should be considered. It would be wise to take into account the software domain because the needs might differ. Considering all of the observed challenges, there were a lot fewer solutions or strategies presented to mitigate technical problems. The same applied to research proposals.

Overall, more research is needed from different domains regarding the challenges. We identified five different domains of which only two had a significant amount of research made (web software domain and embedded systems domain). It would be welcoming to target research also in other software domains such as mobile software, customer specific software domain (such as desktop applications and customer hosted web software), and more less common software domains such as scientific and high-performance computing domain. It would be important to distinguish the different domains where continuous delivery is adopted, as there are slightly different kinds of problems in every domain, as we can note from this study.

Especially detailed case studies would be useful but also action research type of reports are needed. It would be good to remember that when doing these types of research it is very important that the description of the environment should be in an accurate level. When the environment is described in a detailed level the results can be utilized in a correct manner later in other research studies, for example in systematic literature reviews to gather broad synthesis.

Considering the results from this study and results from the related works, the most frequently perceived common challenges regarding the adoption of continuous delivery practices in software development projects are insufficient level of automated testing and customers' adoption of fast cycle of update and feedback. These are the topics that we recommend researchers to pay extra attention to. We believe that both topics will raise many research questions. For example there is a possibility to focus on a certain area of the challenges to deepen the knowledge as has been done in the systematic literature review focusing on customer involvement in continuous deployment [YSRK+16].

Just to bring up a few examples regarding the insufficient level of test automation: what is a sufficient level of test automation for moving towards continuous delivery, what are the drawbacks if the level of test automation is not sufficient, how could organizations discover the insufficient level of test automation before transitioning to continuous delivery practices, and what is the best strategy to move towards continuous delivery (or deployment) in case the test automation level is not sufficient (i.e. prioritization of test automation development). Considering the customers' adoption of fast cycle of update and feedback there were already valid research proposals presented (e.g. lead/pilot customer model, see 5.4).

To extend this study we would like to create a questionnaire (based on the found challenges) for organizations which would indicate the difficulty level of adoption of continuous delivery practices. With the help of the questionnaire organizations could evaluate the cost of such a transformation and probably also discover areas that would need some preparations before the transformation. Also the questionnaire could help organizations in planning the transformation towards continuous delivery (i.e. what are going to be the pain points in the transformation).

References

[ABCS10] Ali, M. S., Babar, M. A., Chen, L., and Stol, K. J.: *A systematic review of comparative evidence of aspect-oriented programming.* Information and Software Technology, 52(9):871 – 887, 2010, ISSN 0950-5849.

[All] Alliance, Agile: *Continuous deployment.* `https://www.agilealliance.org/glossary/continuous-deployment/`, visited on 2016-09-03.

[BA99] Beck, K. and Andres, C.: *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 1999, ISBN 0201616416.

[BCT+06] Budgen, D., Charters, S., Turner, M., Brereton, P., Kitchenham, B., and Linkman, S.: *Investigating the applicability of the evidence-based paradigm to software engineering.* In *Proceedings of the 2006 International Workshop on Workshop on Interdisciplinary Software Engineering Research*, WISER '06, pages 7–14, New York, NY, USA, 2006. ACM, ISBN 1-59593-409-X.

[BF14] Bourque, P. and Fairley, R. E., IEEE Computer Society: *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014, ISBN 0769551661, 9780769551661.

[BKB+07] Brereton, P., Kitchenham, B., Budgen, D., Turner, M., and Khalil, M.: *Lessons from applying the systematic literature review process within the software engineering domain.* Journal of Systems and Software, 80(4):571 – 583, 2007, ISSN 0164-1212. Software Performance, 5th International Workshop on Software and Performance.

[BPT09] Barnett-Page, E. and Thomas, J.: *Methods for the synthesis of qualitative research: a critical review.* BMC Medical Research Methodology, 9(1):1–11, 2009, ISSN 1471-2288.

[CD11] Cruzes, D. S. and Dyba, T.: *Recommended steps for thematic synthesis in software engineering.* In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, Sept 2011.

[CSA15] Claps, G. G., Svensson, R. S., and Aurum, A.: *On the journey to continuous deployment: Technical and social challenges along the way.* Information and Software Technology, 57:21 – 31, 2015, ISSN 0950-5849.

[DD08] Dybå, T. and Dingsøyr, T.: *Empirical studies of agile software development: A systematic review.* Information and Software Technology, 50(9–10):833 – 859, 2008, ISSN 0950-5849.

[DDH07] Dyba, T., Dingsoyr, T., and Hanssen, G. K.: *Applying systematic reviews to diverse study types: An experience report.* In *null*, pages 225–234. IEEE, 2007.

[DMG07] Duvall, P., Matyas, S. M., and Glover, A.: *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series).* Addison-Wesley Professional, 2007, ISBN 0321336380.

[DWAJ$^+$05] Dixon-Woods, M., Agarwal, S., Jones, D., Young, B., and Sutton, A.: *Synthesising qualitative and quantitative evidence: a review of possible methods.* Journal of Health Services Research & Policy, 10(1):45–53B, 2005.

[Fit09] Fitz, T.: *Continuous deployment*, 2009. `http://timothyfitz.com/2009/02/08/continuous-deployment/`, visited on 2016-09-01.

[Fow13] Fowler, M.: *Continuousdelivery*, May 2013. `http://martinfowler.com/bliki/ContinuousDelivery.html`, visited on 2016-09-03.

[FS14] Fitzgerald, B. and Stol, K.: *Continuous software engineering and beyond: Trends and challenges.* In *Proceedings of the 1st*

International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, pages 1–9, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2856-2.

[GWT$^+$14] Galster, M., Weyns, D., Tofan, D., Michalik, B., and Avgeriou, P.: *Variability in software systems - a systematic literature review.* Software Engineering, IEEE Transactions on, 40(3):282–306, March 2014, ISSN 0098-5589.

[HF10] Humble, J. and Farley, D.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010, ISBN 0321601912, 9780321601919.

[HG08] Higgins, J. PT and Green, S.: *Cochrane handbook for systematic reviews of interventions*, volume 5. Wiley Online Library, 2008.

[HRN06] Humble, J., Read, C., and North, D.: *The deployment production line.* In *Proceedings of the Conference on AGILE 2006*, AGILE '06, pages 113–118, Washington, DC, USA, 2006. IEEE Computer Society, ISBN 0-7695-2562-8.

[KA14] Krusche, S. and Alperowitz, L.: *Introduction of continuous delivery in multi-customer project courses.* In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 335–343, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2768-8.

[KBBL09] Kitchenham, B., Brereton, P., Budgen, D., and Li, Z.: *An evaluation of quality checklist proposals: a participant-observer case study.* In *Proceedings of the 13th international conference on Evaluation and Assessment in Software Engineering*, pages 55–64. British Computer Society, 2009.

[KC07] Kitchenham, B.A. and Charters, S.: *Guidelines for performing systematic literature reviews in software engineering.* In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. 2007.

[Kit04] Kitchenham, B.: *Procedures for performing systematic reviews.* Keele, UK, Keele University, 33(2004):1–26, 2004.

[MAK+14] Mäntylä, M. V., Adams, B., Khomh, F., Engström, E., and Petersen, K.: *On rapid releases and software testing: a case study and a semi-systematic literature review.* Empirical Software Engineering, 20(5):1384–1425, 2014, ISSN 1573-7616.

[OAB12] Olsson, H. H., Alahyari, H., and Bosch, J.: *Climbing the "stairway to heaven" - a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software.* In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399, Sept 2012.

[PC12] Poppendieck, M. and Cusumano, M. A.: *Lean software development: A tutorial.* IEEE Software, 29(5):26–32, Sept 2012, ISSN 0740-7459.

[PFMM08] Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M.: *Systematic mapping studies in software engineering.* In *12th international conference on evaluation and assessment in software engineering*, volume 17, pages 1–10. sn, 2008.

[PR08] Petticrew, M. and Roberts, H.: *Systematic reviews in the social sciences: A practical guide.* John Wiley & Sons, 2008.

[Reu] Reuters, Thomson: *Endnote basic.* `http://endnote.com/product-details/basic`, visited on 2016-09-18.

[RHL+16] Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P, Verner, J. M., and Oivo, M.: *Continuous deployment of software intensive products and services: A systematic mapping study.* Journal of Systems and Software, pages –, 2016, ISSN 0164-1212.

[RHWP15] Rahman, A. A. U., Helms, E., Williams, L., and Parnin, C.: *Synthesizing continuous deployment practices used in software* *development.* In *Agile Conference (AGILE), 2015*, pages 1–10, Aug 2015.

[SDJ07] Sjoberg, D. I. K., Dyba, T., and Jorgensen, M.: *The future of empirical methods in software engineering research.* In *Future of Software Engineering, 2007. FOSE '07*, pages 358–378, May 2007.

[SNP15] Smeds, J., Nybom, K., and Porres, I.: *DevOps: A Definition and Perceived Adoption Impediments*, pages 166–177. Springer International Publishing, Cham, 2015, ISBN 978-3-319-18612-2.

[SOV12] Snilstveit, B., Oliver, S., and Vojtkova, M.: *Narrative approaches to systematic review and synthesis of evidence for international development policy and practice.* Journal of Development Effectiveness, 4(3):409–429, 2012.

[TH08] Thomas, J. and Harden, A.: *Methods for the thematic synthesis of qualitative research in systematic reviews.* BMC Medical Research Methodology, 8(1):1–10, 2008, ISSN 1471-2288.

[Woh14] Wohlin, C.: *Guidelines for snowballing in systematic literature studies and a replication in software engineering.* In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 38:1–38:10, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2476-2.

[WP13] Wohlin, C. and Prikladniki, R.: *Editorial: Systematic literature reviews in software engineering.* Inf. Softw. Technol., 55(6):919–920, June 2013, ISSN 0950-5849.

[YSRK+16] Yaman, Sezin Gizem, Sauvola, Tanja, Riungu-Kalliosaari, Leah, Hokkanen, Laura, Kuvaja, Pasi, Oivo, Markku, and Männistö, Tomi: *Customer Involvement in Continuous Deployment: A Systematic Literature Review*, pages 249–265. Springer International Publishing, Cham, 2016, ISBN 978-3-319-30282-9.

A List of Primary Studies

S1: Chen, L.: *Continuous delivery: Huge benefits, but challenges too.* IEEE Software, 32(2):50–54, Mar 2015, ISSN 0740-7459.

S2: Claps, G. G., Svensson, R. S., and Aurum, A.: *On the journey to continuous deployment: Technical and social challenges along the way.* Information and Software Technology, 57:21 – 31, 2015, ISSN 0950-5849.

S3: Bayser, M. de, Azevedo, L. G., and Cerqueira, R.: *Researchops: The case for devops in scientific applications.* In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1398–1404, May 2015.

S4: Gmeiner, J., Ramler, R., and Haslinger, J.: *Automated testing in the continuous delivery pipeline: A case study of an online company.* In Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on, pages 1–6, April 2015.

S5: Karvonen, T., Lwakatare, L. E., Sauvola, T., Bosch, J., Olsson, H. H., Kuvaja, P., and Oivo, M.: Software Business: 6th International Conference, ICSOB 2015, Braga, Portugal, June 10-12, 2015, Proceedings, chapter *Hitting the Target: Practices for Moving Toward Innovation Experiment Systems*, pages 117–131. Springer International Publishing, Cham, 2015, ISBN 978-3-319-19593-3.

S6: Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H. H., Bosch, J., and Oivo, M.: *Towards devops in the embedded systems domain: Why is it so hard?* In 2016 49th Hawaii International Conference on System Sciences (HICSS), pages 5437–5446, Jan 2016.

S7: Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V. P., Itkonen, J., Mäntylä, M. V., and Männistö, T.: *The highways and country roads to continuous deployment.* IEEE Software, 32(2):64–72, Mar 2015, ISSN 0740-7459.

S8: Neely, S. and Stolt, S.: *Continuous delivery? easy! just change everything (well, maybe it is not that easy).* In Agile Conference (AGILE), 2013, pages 121–128, Aug 2013.

S9: Olsson, H. H., Alahyari, H., and Bosch, J.: *Climbing the "stairway to heaven" - a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software.* In 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, pages 392–399, Sept 2012.

S10: Olsson, H.H., Bosch, J., and Alahyari, H.: *Towards R&D as Innovation Experiment Systems: A Framework For Moving Beyond Agile Software Development*, pages 798–805. 2013.

S11: Olsson, H. H. and Bosch, J.: *Towards Agile and Beyond: An Empirical Account on the Challenges Involved When Advancing Software Development Practices*, pages 327–335. Springer International Publishing, Cham, 2014, ISBN 978-3-319-06862-6.

S12: Rissanen, O. and Münch, J.: *Transitioning Towards Continuous Delivery in the B2B Domain: A Case Study*, pages 154–165. Springer International Publishing, Cham, 2015, ISBN 978-3-319-18612-2.

S13: Smeds, J., Nybom, K., and Porres, I.: *DevOps: A Definition and Perceived Adoption Impediments*, pages 166–177. Springer International Publishing, Cham, 2015, ISBN 978-3-319-18612-2.

B Challenges

Id	Challenge	Description	Mitigation strategies	Research proposals	Findings
Social / Procedural					
C1	Incompatibility of traditional processes	Many traditional processes are too slow for continuous delivery. There is usually stages involved in the process where unnecessary delays are introduced (e.g. change advisory board or manual acceptance testing) which might delay the process with many days. Total cycle time of a feature might become significantly longer, for example from a few days to weeks. The smaller the change is the more significant the delay will become.	In transitioning towards continuous deployment the internal action is to involve product management in the short, agile cycle of product development.	There is a need for a research that would identify these incompatible processes (business, development, operations, etc.) within an organization that hinder adopting continuous delivery practices. After the identification processes should be developed furthermore and alternative options should be verified to suit continuous delivery.	2, 81
C2	Working in small batches	Continuous delivery embraces development in small batches. There might be challenges to break them in to smaller batches. It is also challenging to ensure that the incomplete changes are not available to customers. Some companies have a tradition of long lived feature branches.	The use of dark features (or dark launches) will help to alleviate the problem of working in small batches when developing a large feature. This type of release technique allows deployment of features to the production without the appearance of incomplete feature to the customer.	Even though the concept of working in small batches is not new, there is a lack of research of how the size of the feature affects the deployability of a feature when using continuous delivery.	5, 6, 96
C3	Utilizing shorter feedback loop	There might be challenges on how to utilize efficiently the feedback that is provided by the shorter feedback loops to justify, direct or discard the development of some feature.	Implement a strategy to monitor customer behavior through some data analytics platform. With such a strategy, tools, and process it is possible to steer the development of features in a way that might be more in line with customer's expected needs. Experiments should be made as small as possible and as quickly as possible.	-	7

C4	Lack of standard practices	Lack of understanding of the continuous delivery process, especially for the novice developers, is seen as a challenge. For some of the teams in a specific case organization there was only some sort of documentation available associated with the continuous delivery process or it did not even exist. However, some developers felt that there were not so many differences to their original workflow when they were adopting the continuous delivery practices. Still there is a demand for industry standards for the software development processes regarding to the continuous delivery practices.	Adopt social rules that complements the continuous deployment of software which must be adhered.	-	21, 36
C5	Keeping the build green	Merge conflicts and assuring the deployability of every software version may pose a challenge.	Strategy for stopping the entire team to concentrate on the fixing issues must be planned and in place to keep the build green.	-	28
C6	Propagation of changes across teams	If systems are complex (i.e. embedded systems) there might be challenges on changes across company (from team to team) on continuous basis.	Teams should be responsible for accepting continuous flow of changes and also propagating the changes rapidly to other teams.	-	113
C7	Ambiguity in coordination	When some incident occurs the question is who should fix, what, and when?	To continuously deliver a software the build should always be green. Team should collectively aim for this goal and it should be the primary goal.	-	11
Social / Organizational / Resources					

C8	Lack of motivation	Because successful adoption of continuous delivery practices needs collaboration from different organizational units the lack of motivation for continuous delivery might pose a challenge. Management commitment is crucial because of motivation but also because establishing deployment pipeline is time consuming and costly which is usually made in the initial phase of moving towards continuous delivery. Management might even be resistant towards continuous delivery or changes to existing processes which will be a challenge for adopting continuous delivery practices.	Make sure that top-management is implementing a strategy to push the need of continuous delivery all the way through lower organization units. Set a goal for achieving continuous delivery.	-	10, 33, 53, 63

C9 Increased pressure	When adopting the continuous delivery practices software developers will become more responsible for deployment process. In addition, the idea that some code could be released to the production environment immediately will increase the pressure for the software developers. In case there is no manual QA phase the pressure will be increased even more. Software developers must be *"diligent in writing tests and monitoring the system"* [S8]. Also maintaining the code quality is an increasingly important part of the software developer's daily work if there is no manual quality assurance personnel. For that the practices and static code analysis tools should be firmly in place. Software developer's reputation might suffer if she deploys a broken build or some other way impairs the customer's user experience. It might even affect negatively the business. Also the possibility to decide what to deploy and when might create confusion in cooperation with internal (management, sales, etc.) and external (e.g. customer) stakeholders. All things considered the responsibilities and pressures will be increased for the software developer.	Improving communication between developers and managers.	Need for proper management program for adopting the continuous deployment process in an organization to ensure that the process is not destructive for it.	12, 62, 75, 100
C10 New roles	Team members have to take new roles related to different tasks when adopting continuous delivery practice.	Shift from personal to team thinking. Taking the responsibility as a team of delivering the software to customers without negative effects to the customers.	-	13
C11 Lack of knowledge	It requires enough knowledge and experience from team members to fully succeed in the adoption of continuous delivery practices.	Ensure that there is low or nonexistent learning curve for team members in adopting continuous delivery practices when introducing it to existing systems and process for example integrating the new practices to old workflow.	-	9

C12 Deep specialization	Some modules (especially modules close to the hardware) might require some level of specialization. That may pose a challenge for cross-functional teams that continuous delivery needs.	Ensure that there is effective communication taking place between modules and that CI practices are implemented and work as a coordinating mechanism both within and across the team.	-	42
C13 Fear of unemployment	Especially the QA team might get irrational fear of unemployment as there are supposed to shift mindset towards automated testing instead of manual testing.	Planning good automated tests requires professional QA mindset. Planning requires hours of reading stories, performing research into the background of the story and creating different angels to test cases. If there is some time left from the old way of working, QA could invest in monitoring and analyzing the system.	-	76
Social / Organizational / Marketing and sales				
C14 Marketing versionless product	Traditional marketing strategies are not suitable for marketing a product that has no versions.	Make a blog and post about product's changes.	-	14
C15 Uncertainty of ready features	Because there might exist uncertainty within development team about when some feature will be ready, there may be challenges in relationship between marketing and development teams.	Increase the level of transparency between the development team and sales and marketing unit.	-	73
C16 Selling experimental functionality	From customer perspective the idea of buying partially developed functionality with intentions to experiment might be challenging.	-	Develop business and pricing models that support this kind of short-cycle innovation with the help of customer usage data.	86

C17 Customer policies	Releasing software in small increments, in fast pace, and possibly with some experimental mindset may not necessarily meet the expectations of all customers. Large companies might hesitate to buy a software which might seem not to be thoroughly tested because of the risk or fear that there might be increased amount of bugs in production. Some internal policies may prevent a company from buying such a "not thoroughly tested" software. Also compliance with standards (national or international) will pose a challenge for continuous delivery. Usually such standards require that patient and user safety is guaranteed by not introducing adverse side effects with the changes.	-	-	18, 55
Social / Organizational / Customer adoption				
C18 Feature discovery	Continuous delivery enables software to be delivered and deployed to customers in faster pace but it is also important to aid users to find these new features.	Make a blog and post about product's changes.	-	19

C19 Cycle of update (and feedback)	Not all of the customers welcome continuous delivery type of development method simply because of a too frequent update pace. It may be difficult to demonstrate the added value of such a fast pace to all customers especially if it might affect somehow negatively the customer organization. Negative effects may arise when the customer organization has to react somehow to the change for example to learn new features, to learn new UIs (especially if the existing UI is already hard to learn), adjust somehow to changes (organization or processes), do acceptance testing, or to be involved in the installation process. Especially in embedded systems domain deploying a new software may involve numerous activities in customer sites (updating dependencies in multiple places) and thus be very tedious to customer organizations. Some actions (e.g. customer acceptance testing) need to be even scheduled with the customer which hinders continuous delivery. Also just the possibility of new bugs may be big enough reason to be reluctant to update, especially for organizations who do experience that the software is working correctly and there is no need for new features at the moment.	Apply continuous deployment only to non-business critical software. Invest in good UI design.	-	20, 37, 48, 50, 54, 102, 103, 106, 107
Social / Organizational / Hierarchy				
C20 Coordinating units	There are strong indications that to fully succeed in the continuous delivery practice requires involvement of multiple organizational units. That poses a challenge of coordination and collaboration of multiple organizational units to achieve the common goal of successful continuous delivery.	The deployments should be more carefully planned in a way that supports continuous delivery of multiple systems that are interdependent.	-	8

C21 Barriers between units	Releasing activities may involve operations and communication needs between many different units of an organization. If organization hierarchy promotes competing goals for different units there might be too much tension between them.	Restructure the organization to break down barriers and promote collaborative culture.	Further research is needed to understand challenges the organizations face when adopting the continuous delivery in more more depth and to develop strategies and practices to tackle them more efficiently.	1
C22 Different pace between units	There might be challenges in the collaboration between units (or company functions) if they operate in a different pace. For example development might run on continuous releases while marketing runs with six months release cycle. Business runs usually on *"dates and dollars"* [S8] with annual and quarterly goals and marketing events.	Use annual roadmap but do not make detailed plans for the future. Plan for current and next quarter but no further. Some candidate initiates can exist for the rest quarters. Participate in monthly product meetings and discuss released and to be released features. Take input on the near-term roadmap. Share the status of the project to other stakeholders. Use historical data to estimate future throughput of the team.	-	39, 79
Social / Organizational / Culture				
C23 Releasing experimental functionality	In companies where there is a strong tradition of rigorous testing and validation processes there might be a resistance towards experimentation of functionality that may not be fully completed.	Infrastructure and tools for testing run-time variations, data collection and early provision of feedback must be in place. These functionalities require usually some sort of architectural modifications if they do not exist already.	-	85
C24 Sharing of status	If the quality status of the projects are not visible to other business units they may not be able to react to customer requests related to status, quality, or problem situations. Also other units do not know what is the effect of some build to the product if it is not clearly communicated. There might also exist a lack of trust in quality of the product if such information is not available.	Mechanism for the rollback of releases and to deploy components instead of whole system might ease the lack of trust in quality.	-	82, 83

C25 Lack of business model	There is a lack of business model that suits continuous delivery type of development. It is hard to implement continuous delivery practices, as it flows continuously, if business acts in a more conservative way, promoting fixed releases and fixed pricing according to features promised and usually frozen upfront.	Identify lead (pilot) customers with whom you can start to build a culture and capability for continuously delivering a software. Use lead customers as role models to other customers. Review business model to support continuous delivery practices and align all business functions with R&D organization in order to facilitate continuous delivery.	-	34, 84, 105
Technical / Tools				
C26 No comprehensive platform suite	Comprehensive and robust highly customizable solutions for continuous delivery are scarce or do not exist or might introduce vendor lock-in.	Develop own continuous delivery platform solution which might be costly. Also building comprehensive platform might involve many different tools and technologies. Avoiding vendor lock-in is challenging.	Widely accepted standards, open APIs, and active plugin ecosystems for existing or future platforms will alleviate the challenge.	3
C27 Branching in VCS	There should be only one branch of the software in the version control system which is under development. This poses a challenge when developing larger features.	Develop larger features on a separate branch and merge as soon as possible to mainline. Split large features to smaller batches.	-	22
C28 Cannot use optimal hardware resources	In scientific computing it is important to use optimal resources (CPU vs GPU or other co-processors) depending on the calculation. Usually the information is only available at deployment so the deployment scripts should make the decision about the best variation of configurations. This might need even recompilation of the code.	-	-	31
Technical / Architectural				

C29 Incompatible architectures	For example large monolithic applications that aren't amenable to continuous delivery practice are a great challenge. There might be a huge number of such applications in the industry.	Transform architecture towards more modular architecture which allows upgrading the system in smaller and independent parts or improve capability for continuous delivery some other way. Also motivation for a such possibly expensive transformation is needed especially because the value is not immediately evident as it might seem for new features.	There is a need for understanding the characteristics of such incompatible architectures and to develop strategies to overcome and figure out the steps of how to move towards continuous delivery.	4, 109
C30 Legacy system integrations	If there is some legacy system integrations they might pose a challenge to the adoption of continuous delivery practices in full scale.	-	-	64
C31 Code base size	If the code base size and complexity is too high it might pose a challenge for continuous delivery practice as it takes too much time to build, compile and package code.	-	-	66
Technical / Documentation				
C32 Maintaining documentation	If the software has different versions out (for example in customer's environments and software that is versionless but hosted in separated environments for each customer) there might be a challenge of managing the documentation for these different versions.	Use wiki type of tool to maintain the documentation.	-	15
C33 Maintaining change logs	Because of smaller and faster version releases it might become as a surprise that managing the version log requires more effort. There is no point in discussing every release in detail with every customer.	-	-	104
Technical / Testing				

C34 Insufficient testing	There may be challenges in having enough and maintaining existing automated acceptance testing. It will be very laborious to build automated testing suite from the scratch that is sufficient. This might be especially laborious for mature software. Also there might be challenges in prioritizing the tests that should be created to validate a release. Testing suite should also be maintained which needs resources. Automated acceptance testing must be in place in deployment pipeline before deployment to production is possible with continuous delivery.	Diligent testing practices must be adopted to ensure that testing is thorough since deploying software is dependent on tests passing on the continuous integration server. Invest in testing practices, testing, testing automation and ensure its thoroughness.	-	17, 35, 38, 47, 60, 89, 90, 95
C35 Testability of outputs	In physical simulation the results are complex and even the experts are not able to say whether the result is right or wrong. There is a possibility to say whether the results are in line with a theory but deciding based from the result if the calculation was wrong or right is not always possible. The situation might be even more complex in the context of cognitive systems.	-	-	32
C36 Long acceptance tests	In embedded systems some factory acceptance testing might take several months and it is done after the development is completed what can be considered as a barrier for continuous delivery. However, test suite that takes hours, for example a test suite run in a web software development, delays instant releases and hence it is a hindering issue for continuous delivery practice.	-	-	43, 65

C37 Configuration differences	In a software which is installed specifically for each customer, usually there are different environments with different configurations and external dependencies, probably interfered with other vendors. That poses major challenges for automated acceptance testing. These kinds of configuration differences lead to multiple and complex variations which are very difficult to test reliably. Hence the testing is usually done with similar configurations. This leads to the problem that there will be many faults to be found in the acceptance testing phase on the customer end. Same problem occurs also in internal development process if there exist differences between environments.	-	-	45, 46, 68
C38 Automated UI testing	Web browsers introducing different implementations of standards and mobile game user interfaces with fragmented device base may pose a challenge for automated acceptance testing.	-	-	61, 71
C39 Performance testing	If the subject of testing is a component of some network infrastructure, load testing might require enormous amount of hardware.	-	-	69
C40 Manual testing	Existing manual or explorative testing phases before release might hinder rapid releases of software that continuous delivery is aiming at. There might be support for manual testing because QA testers might find bugs that automated tests did not catch.	-	-	70, 72
C41 Differences in environments	If development and testing environments do not simulate or are not similar to production it may pose a threat to validation of the software (testing) and thus continuous delivery. This is also a problem for sharing information and responsibilities.	-	-	110

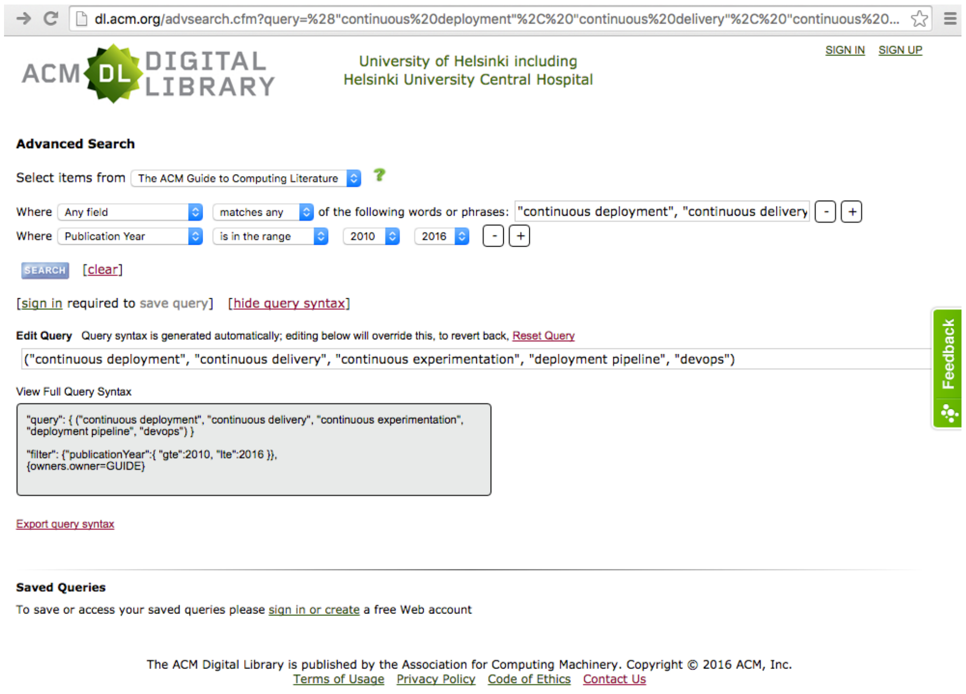
C42 Ignoring test suite	False assumption that there is nothing wrong even though the tests fail (wrong data etc.) might lead to bad situations and probably hides real problems with the code. Same goes with flaky tests.	Never ignore failing tests. Always dig into the failure and fix what's wrong. To release quick doesn't mean you should rush. Use flaky tests tools that inspect and handle flaky tests to get real results. Flaky tests must be fixed, eliminated or handled correctly.	-	74, 78
Technical / Resources				
C43 Infrastructural requirements	To properly handle cross-product dependencies and testing there must be suitable infrastructure in place and enough resources for the deployment pipeline processing to keep the pace fast enough.	Provide enough hardware to continuous delivery pipeline components to continuously run the deployment pipeline.	-	25
C44 Hardware based infrastructure	Hardware/PC based infrastructure is a great barrier for adopting continuous delivery practices.	Update the product architecture from PC platform to a virtualized cloud environment.	-	40
Technical / Dependencies				
C45 Cross-product dependencies	If a software product has tightly coupled cross-product or module dependencies it may pose a challenge for deployment process as there is a great need for rigorous integration testing across these products. Also there may be challenges if the products or some parts of product are in different stage of development. The same applies for API integrations. And in addition to the rigorous testing, the API changes must be planned and discussed in advance with stakeholders. The support for automatically updating the integrations may *"require unduly amount of work considering the results"* [S12].	Provide enough hardware resources to testing infrastructure.	-	26, 67, 91, 97
C46 No packages available in repositories	No package available for certain operating system version to satisfy a library dependency.	Create custom packages and set them available for private repository.	-	29

C47 Outdated environments	Customers may have strict hardware/environment policies which may lead to outdated operating systems. This is a challenge for compatibility and testing.	Use containers/virtual machines (i.e. Vagrant) to test binary compatibility to customers' environments (i.e. if customer is using RHEL5, use centOS5 as a base for software testing).	-	30
C48 Long product lifecycle	In the embedded systems domain the lifecycle of installed software might often be quite long with lots of legacy code involved. When new features are added the software may have dependencies to old software made years ago. Ensuring the compatibility between versions might be challenging.	-	-	49
Technical / Deployment				
C49 Seamless upgrades	There may be complications related to seamless upgrades that continuous delivery demands. Extra resources (instances of software) might be needed to perform zero-downtime upgrades and customer data has to be preserved correctly. This is also a great problem for embedded systems domain where it is critical to have no downtime and two parallel systems would be inconvenient to use. For example some factories must stop the whole production line for a day or so to run the update process. This is usually also a big cost for the customer. Same applies for a web software, if seamless upgrades are not achieved the user has to stop working or might lose data in the progress. There should be no interferences with ongoing usage in case of frequent update pace.	Use of two parallel systems to perform upgrades to shift user from an old to a new version of the software (blue-green deployment).	-	27, 51, 56, 88, 108

C50 Diversity of client configurations	Diversity of configurations is also a great challenge for the application deployment phase. For example diversity of client network configurations increases the deployment time complexity when deploying new releases of software to network devices. Diversity of configuration is also a challenge when there is legacy product to be updated and the configuration is very complex. Configurations should be updated automatically in the deployment process. Managing the diversity of configurations is also problematic.	-	-	57, 80, 92, 93, 94
C51 Third-party distribution channels	If there is a third party involved in the distribution process which also contributes or manages somehow the software updates it will pose a challenge for continuous delivery practice. For example mobile games and AppStore, or some other device (for example mobile solutions) software update which is distributed through third party (telephony companies).	-	-	58/59 (duplicate)
C52 Efficient rollback mechanism	The only way to rollback a deployment is to degrading to previous system version which is not so efficient and not sufficient when moving towards more frequent releases.	-	-	87
Technical / Security and safety				
C53 Critical systems	Performance, security (and business) critical systems are a challenge for continuous delivery.	-	-	41
C54 Not enough access	Not having enough access to environments is a challenge for continuous delivery.	-	-	112

C55	Limited view on run environment	Development companies might have limited view on the environment (configuration, monitoring) the software is installed to (customers environment) especially in the embedded systems domain. Monitored data may be accessible only when testing or fault diagnostic.	-	-	44, 52
C56	Product quality may decrease	Because the deployments occur in a more faster pace there might be an increase in the amount of bugs in the production environment.	Fix bugs immediately and deploy the fix to the production environment as soon as possible. Roll back to previous working state if software is not in a fully functioning state.	-	16
Technical / Integrations					
C57	Partner plugin integration	Constantly changing version and code affects integration issues to partner plugins. If plugin is not continuously tested for the integration and maintained, the plugin will become broken.	Offer customers a smaller amount of plugins which are maintained and compatible with the latest versions.	-	24
C58	Changes in database schema	Minor changes in code may introduce an unplanned database schema change. This goes beyond internal development in case of plugin ecosystem to external partners.	Ensure that there is sufficient amount of testing around the database to prevent deployments to affect the database. Use rollbacks to reset issues.	-	23
C59	Multiple versions out	If deployed software product is built from different components and there is multiple customers with own environments there might be multiple versions out at the same time because components can be in different versions in different customer installations.	-	-	99, 111

C Search Screens



dl.acm.org/advsearch.cfm?query=%28"continuous%20deployment"%2C%20"continuous%20delivery"%2C%20"continuous%20...

University of Helsinki including
Helsinki University Central Hospital

SIGN IN SIGN UP

Advanced Search

Select items from: The ACM Guide to Computing Literature

Where: Any field | matches any | of the following words or phrases: "continuous deployment", "continuous delivery

Where: Publication Year | is in the range | 2010 | 2016

SEARCH [clear]

[sign in required to save query] [hide query syntax]

Edit Query Query syntax is generated automatically; editing below will override this, to revert back, Reset Query

```
("continuous deployment", "continuous delivery", "continuous experimentation", "deployment pipeline", "devops")
```

View Full Query Syntax

```
"query": { ("continuous deployment", "continuous delivery", "continuous experimentation",
"deployment pipeline", "devops") }

"filter": {"publicationYear":{ "gte":2010, "lte":2016 }},
{owners.owner=GUIDE}
```

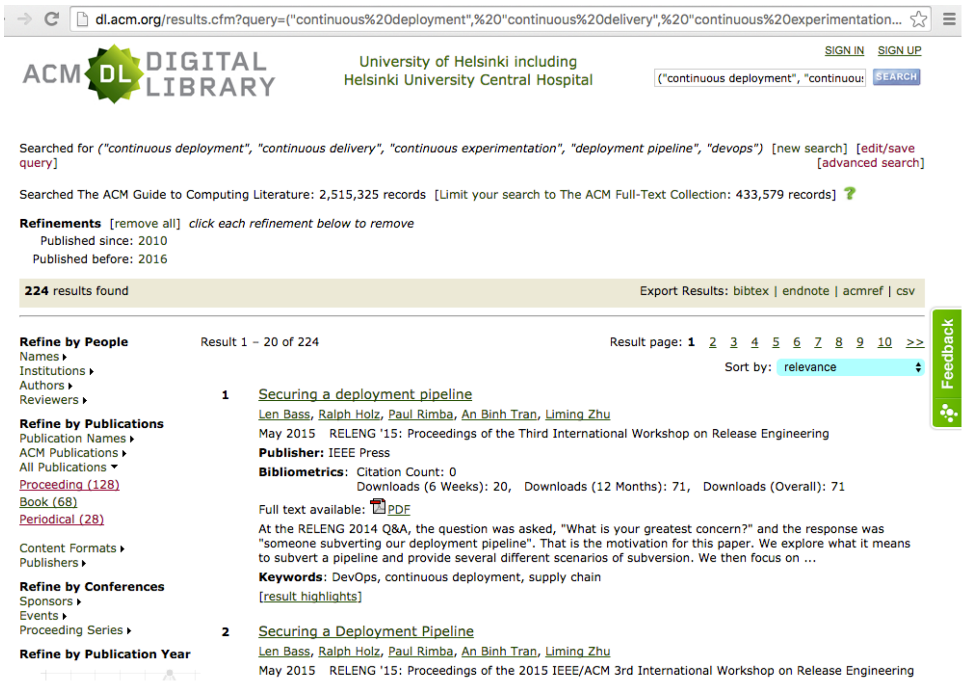
Export query syntax

Saved Queries

To save or access your saved queries please sign in or create a free Web account

The ACM Digital Library is published by the Association for Computing Machinery. Copyright © 2016 ACM, Inc.
Terms of Usage Privacy Policy Code of Ethics Contact Us

Figure 8: ACM advanced search screen



dl.acm.org/results.cfm?query=("continuous%20deployment",%20"continuous%20delivery",%20"continuous%20experimentation...

ACM DL DIGITAL LIBRARY
University of Helsinki including Helsinki University Central Hospital
SIGN IN SIGN UP
("continuous deployment", "continuous
SEARCH
Feedback

Searched for *("continuous deployment", "continuous delivery", "continuous experimentation", "deployment pipeline", "devops")* [new search] [edit/save query] [advanced search]

Searched The ACM Guide to Computing Literature: 2,515,325 records [Limit your search to The ACM Full-Text Collection: 433,579 records]

Refinements [remove all] *click each refinement below to remove*
Published since: 2010
Published before: 2016

224 results found Export Results: bibtex | endnote | acmref | csv

Refine by People
Names
Institutions
Authors
Reviewers

Refine by Publications
Publication Names
ACM Publications
All Publications
Proceeding (128)
Book (68)
Periodical (28)

Content Formats
Publishers

Refine by Conferences
Sponsors
Events
Proceeding Series

Refine by Publication Year

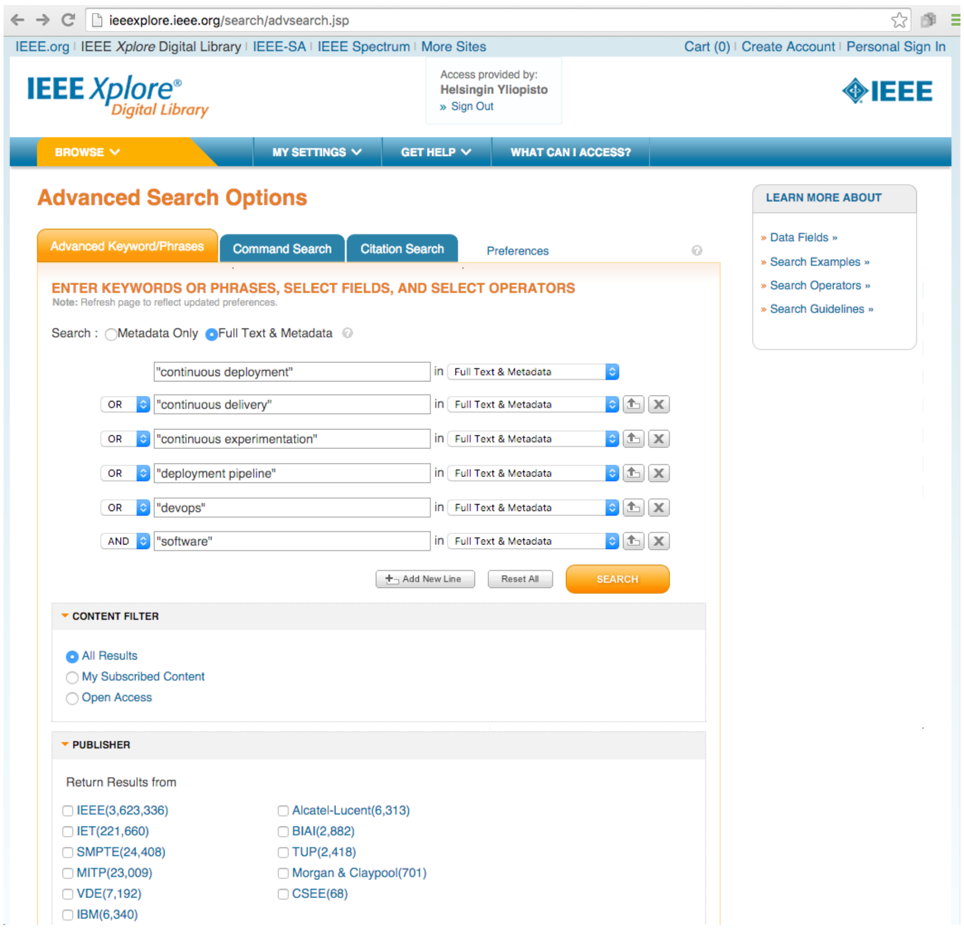
Result 1 – 20 of 224 Result page: **1** 2 3 4 5 6 7 8 9 10 >>

Sort by: relevance

1 Securing a deployment pipeline
Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, Liming Zhu
May 2015 RELENG '15: Proceedings of the Third International Workshop on Release Engineering
Publisher: IEEE Press
Bibliometrics: Citation Count: 0
Downloads (6 Weeks): 20, Downloads (12 Months): 71, Downloads (Overall): 71
Full text available: PDF
At the RELENG 2014 Q&A, the question was asked, "What is your greatest concern?" and the response was "someone subverting our deployment pipeline". That is the motivation for this paper. We explore what it means to subvert a pipeline and provide several different scenarios of subversion. We then focus on ...
Keywords: DevOps, continuous deployment, supply chain
[result highlights]

2 Securing a Deployment Pipeline
Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, Liming Zhu
May 2015 RELENG '15: Proceedings of the 2015 IEEE/ACM 3rd International Workshop on Release Engineering

Figure 9: ACM search results screen


ieeexplore.ieee.org/search/advsearch.jsp

IEEE.org | IEEE *Xplore* Digital Library | IEEE-SA | IEEE Spectrum | More Sites — Cart (0) | Create Account | Personal Sign In

Access provided by: Helsingin Yliopisto » Sign Out

BROWSE | MY SETTINGS | GET HELP | WHAT CAN I ACCESS?

Advanced Search Options

Advanced Keyword/Phrases | Command Search | Citation Search | Preferences

ENTER KEYWORDS OR PHRASES, SELECT FIELDS, AND SELECT OPERATORS

Note: Refresh page to reflect updated preferences.

Search : Metadata Only [✓ Full Text & Metadata]

	"continuous deployment"	in	Full Text & Metadata
OR	"continuous delivery"	in	Full Text & Metadata
OR	"continuous experimentation"	in	Full Text & Metadata
OR	"deployment pipeline"	in	Full Text & Metadata
OR	"devops"	in	Full Text & Metadata
AND	"software"	in	Full Text & Metadata

Add New Line | Reset All | SEARCH

CONTENT FILTER

[✓ All Results] My Subscribed Content Open Access

PUBLISHER

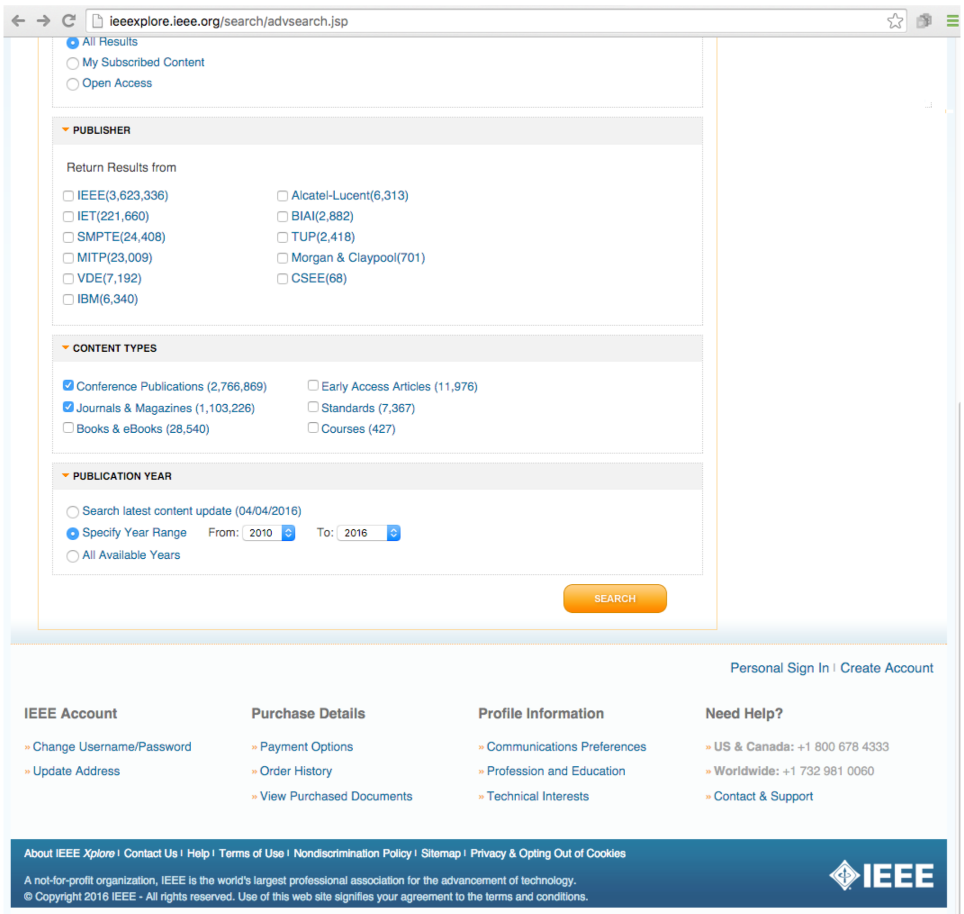
Return Results from

IEEE(3,623,336)	Alcatel-Lucent(6,313)
IET(221,660)	BIAI(2,882)
SMPTE(24,408)	TUP(2,418)
MITP(23,009)	Morgan & Claypool(701)
VDE(7,192)	CSEE(68)
IBM(6,340)	

LEARN MORE ABOUT

» Data Fields »
» Search Examples »
» Search Operators »
» Search Guidelines »

Figure 10: IEEE advanced search screen, part 1



ieeexplore.ieee.org/search/advsearch.jsp

[✓ All Results]
My Subscribed Content
Open Access

PUBLISHER

Return Results from

☐ IEEE(3,623,336)	☐ Alcatel-Lucent(6,313)
☐ IET(221,660)	☐ BIAI(2,882)
☐ SMPTE(24,408)	☐ TUP(2,418)
☐ MITP(23,009)	☐ Morgan & Claypool(701)
☐ VDE(7,192)	☐ CSEE(68)
☐ IBM(6,340)	

CONTENT TYPES

☑ Conference Publications (2,766,869)	☐ Early Access Articles (11,976)
☑ Journals & Magazines (1,103,226)	☐ Standards (7,367)
☐ Books & eBooks (28,540)	☐ Courses (427)

PUBLICATION YEAR

Search latest content update (04/04/2016)
[✓ Specify Year Range] From: 2010 To: 2016
All Available Years

SEARCH

Personal Sign In | Create Account

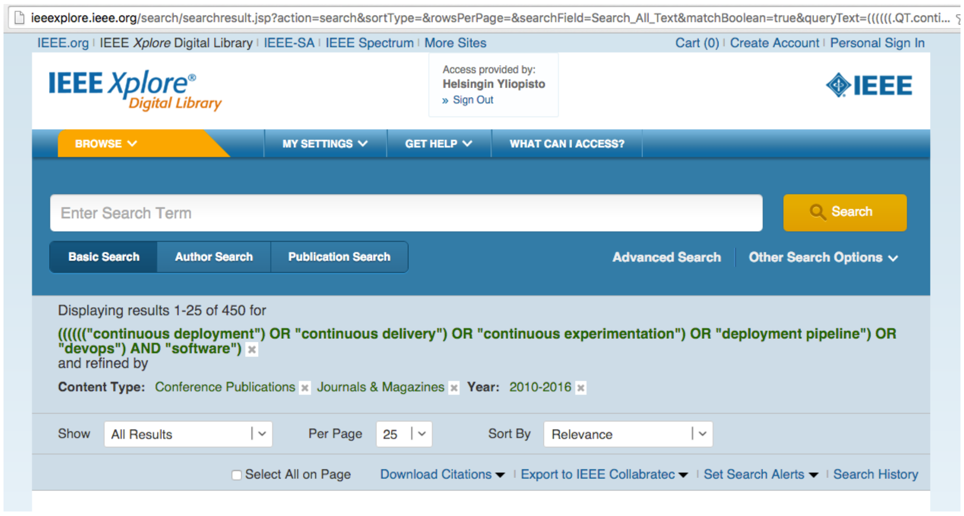
IEEE Account	Purchase Details	Profile Information	Need Help?
» Change Username/Password	» Payment Options	» Communications Preferences	» **US & Canada:** +1 800 678 4333
» Update Address	» Order History	» Profession and Education	» **Worldwide:** +1 732 981 0060
	» View Purchased Documents	» Technical Interests	» Contact & Support

About IEEE *Xplore* | Contact Us | Help | Terms of Use | Nondiscrimination Policy | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest professional association for the advancement of technology.
© Copyright 2016 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

IEEE

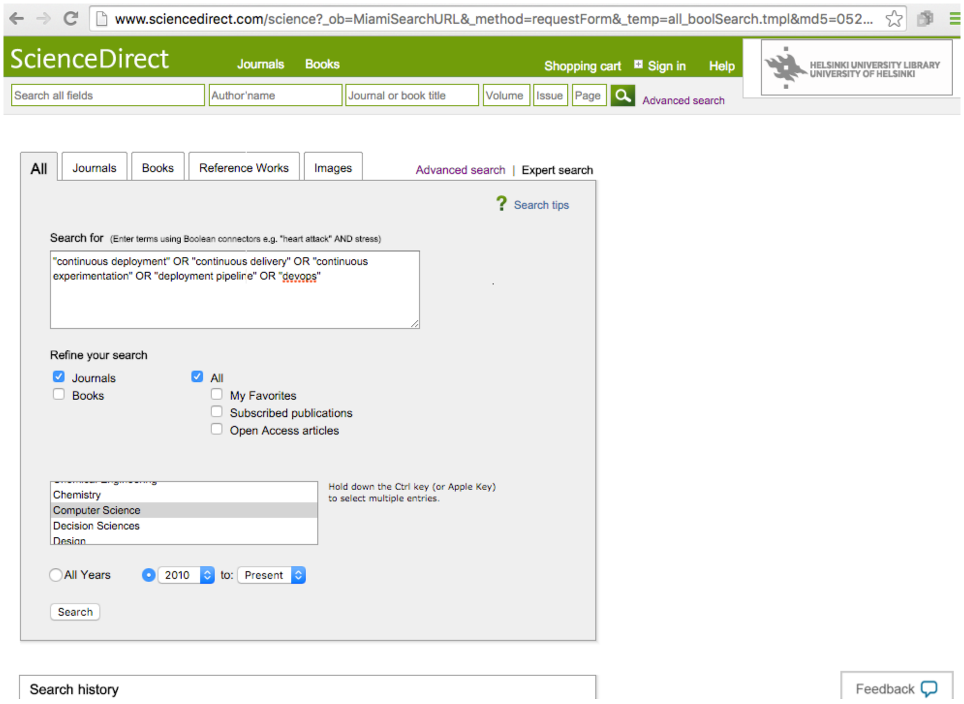
Figure 11: IEEE advanced search screen, part 2


ieeexplore.ieee.org/search/searchresult.jsp?action=search&sortType=&rowsPerPage=&searchField=Search_All_Text&matchBoolean=true&queryText=((((((.QT.conti...
IEEE.org | IEEE Xplore Digital Library | IEEE-SA | IEEE Spectrum | More Sites
Cart (0) | Create Account | Personal Sign In
IEEE Xplore® Digital Library
Access provided by: Helsingin Yliopisto » Sign Out
IEEE
BROWSE
MY SETTINGS
GET HELP
WHAT CAN I ACCESS?
Enter Search Term
Search
Basic Search
Author Search
Publication Search
Advanced Search
Other Search Options
Select All on Page
Download Citations
Export to IEEE Collabratec
Set Search Alerts
Search History

Displaying results 1-25 of 450 for
((((("continuous deployment") OR "continuous delivery") OR "continuous experimentation") OR "deployment pipeline") OR "devops") AND "software")
and refined by
Content Type: Conference Publications Journals & Magazines **Year:** 2010-2016

Show: All Results | Per Page: 25 | Sort By: Relevance

Figure 12: IEEE search results screen



Search for (Enter terms using Boolean connectors e.g. "heart attack" AND stress)
"continuous deployment" OR "continuous delivery" OR "continuous experimentation" OR "deployment pipeline" OR "devops"

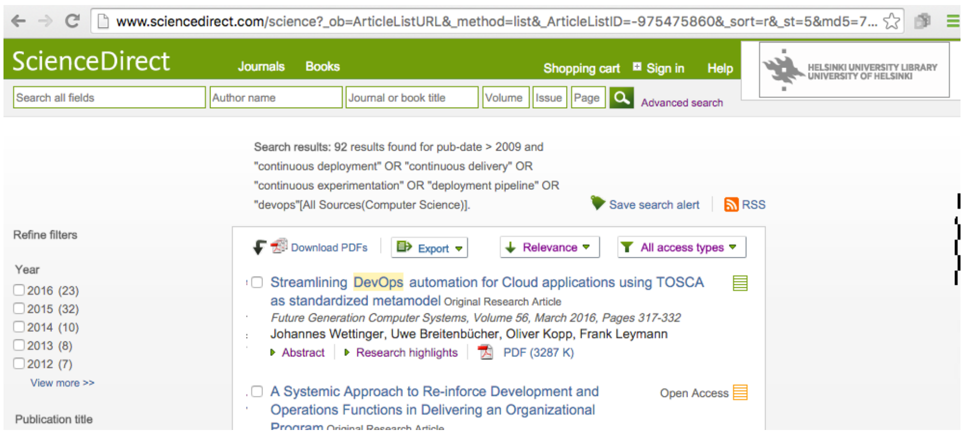
Refine your search
- [✓] Journals
- [ ] Books
- [✓] All
- [ ] My Favorites
- [ ] Subscribed publications
- [ ] Open Access articles

Chemistry
Computer Science
Decision Sciences
Design

Hold down the Ctrl key (or Apple Key) to select multiple entries.

○ All Years ● 2010 to: Present

Figure 13: ScienceDirect expert search screen

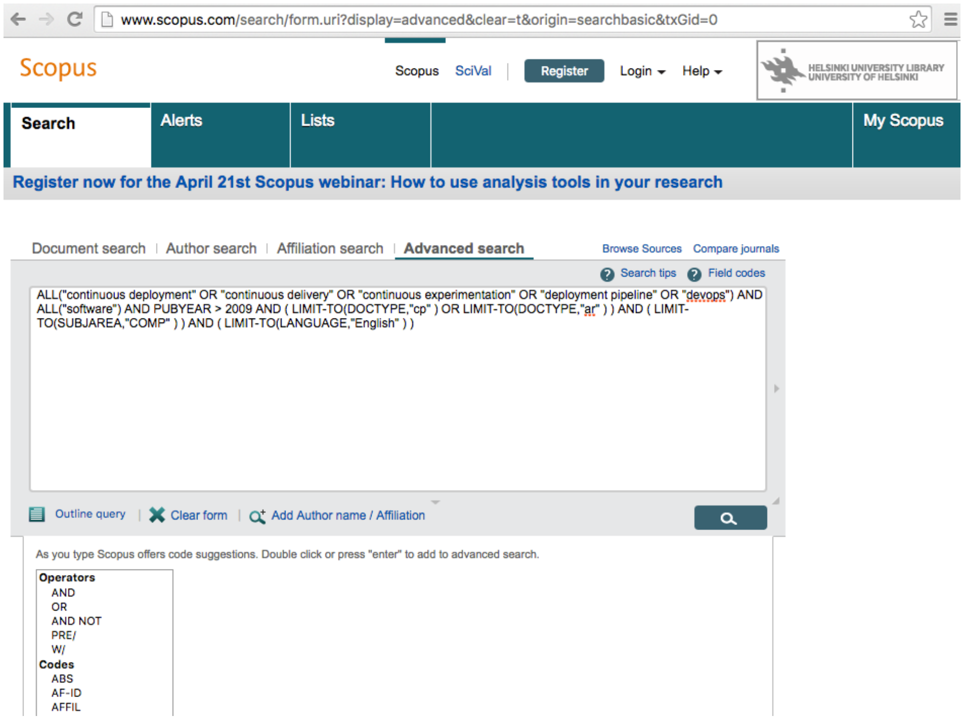


Search results: 92 results found for pub-date > 2009 and "continuous deployment" OR "continuous delivery" OR "continuous experimentation" OR "deployment pipeline" OR "devops"[All Sources(Computer Science)].

Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel Original Research Article
Future Generation Computer Systems, Volume 56, March 2016, Pages 317-332
Johannes Wettinger, Uwe Breitenbücher, Oliver Kopp, Frank Leymann

A Systemic Approach to Re-inforce Development and Operations Functions in Delivering an Organizational Program Original Research Article

Figure 14: ScienceDirect search results screen

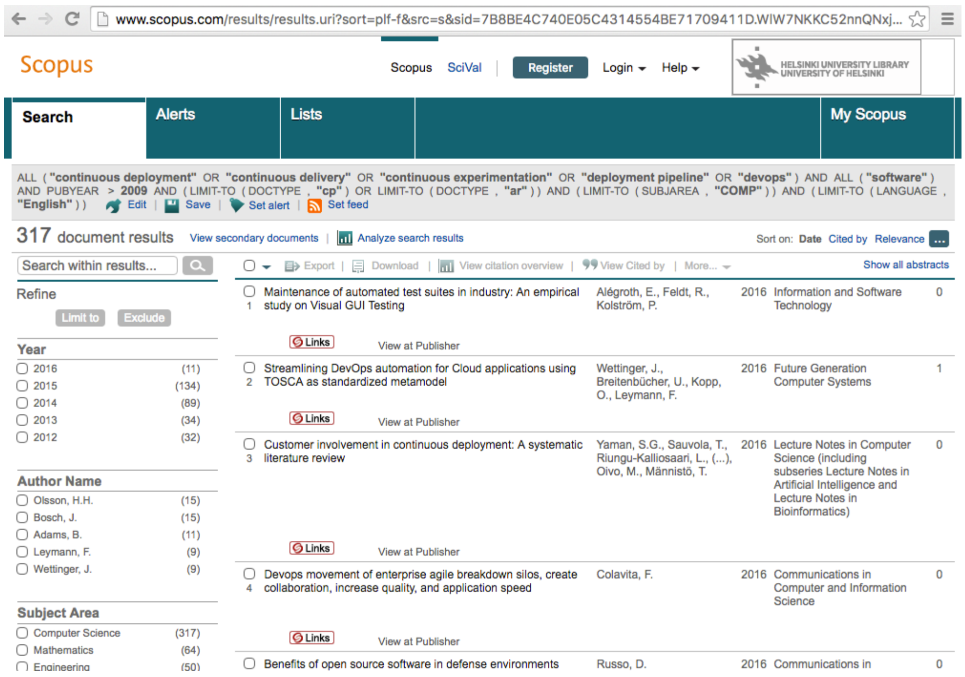


Register now for the April 21st Scopus webinar: How to use analysis tools in your research

```
ALL("continuous deployment" OR "continuous delivery" OR "continuous experimentation" OR "deployment pipeline" OR "devops") AND
ALL("software") AND PUBYEAR > 2009 AND ( LIMIT-TO(DOCTYPE,"cp" ) OR LIMIT-TO(DOCTYPE,"ar" ) ) AND ( LIMIT-
TO(SUBJAREA,"COMP" ) ) AND ( LIMIT-TO(LANGUAGE,"English" ) )
```

As you type Scopus offers code suggestions. Double click or press "enter" to add to advanced search.

Figure 15: Scopus advanced search screen



www.scopus.com/results/results.uri?sort=plf-f&src=s&sid=7B8BE4C740E05C4314554BE71709411D.WIW7NKKC52nnQNxj...

Scopus
Scopus SciVal | Register Login Help
HELSINKI UNIVERSITY LIBRARY UNIVERSITY OF HELSINKI
Search
Alerts
Lists
My Scopus

ALL ("continuous deployment" OR "continuous delivery" OR "continuous experimentation" OR "deployment pipeline" OR "devops") AND ALL ("software") AND PUBYEAR > 2009 AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English"))

317 document results

View secondary documents | Analyze search results
Sort on: Date Cited by Relevance ...
Search within results...
Export | Download | View citation overview | View Cited by | More...
Show all abstracts
Refine
Limit to Exclude

Year

2016	(11)
2015	(134)
2014	(89)
2013	(34)
2012	(32)

Author Name

Olsson, H.H.	(15)
Bosch, J.	(15)
Adams, B.	(11)
Leymann, F.	(9)
Wettinger, J.	(9)

Subject Area

Computer Science	(317)
Mathematics	(64)
Engineering	(50)

1	Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing	Alégroth, E., Feldt, R., Kolström, P.	2016	Information and Software Technology	0
2	Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel	Wettinger, J., Breitenbücher, U., Kopp, O., Leymann, F.	2016	Future Generation Computer Systems	1
3	Customer involvement in continuous deployment: A systematic literature review	Yaman, S.G., Sauvola, T., Riungu-Kalliosaari, L., (...), Oivo, M., Männistö, T.	2016	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)	0
4	Devops movement of enterprise agile breakdown silos, create collaboration, increase quality, and application speed	Colavita, F.	2016	Communications in Computer and Information Science	0
	Benefits of open source software in defense environments	Russo, D.	2016	Communications in	0

Figure 16: Scopus search results screen

D Findings (Raw Challenges / Form III)

Findings were collected with the Form III and they are presented in the next page as included spreadsheet.

Challenges (Form III)

CH	CS	Category	Title	Description	Mitigation strategy	Research proposal
1	1	Organizational	Barriers among teams	Release activities involve many divisions of the company. Each has its own interests, ways of working, and perceived territories of control. Tension existed between divisions due to competing goals. For example, we needed root access to the servers, and another team controlled this permission. Arriving at a solution involved much consultation and negotiation over six months.	To address the organizational challenges, the leadership team restructured the organization to break down barriers among teams and promote a collaborative culture. The situation has improved since.	Although literature on organizational change exists, little, if any, research focuses on introducing continuous delivery to an organization. Further research on this topic—for example, understanding the challenges in more depth and developing strategies and practices to tackle them more effectively—will significantly help an organization's smooth adoption of continuous delivery.
2	1	Process	Traditional processes do not fit for continuous delivery	Many traditional processes hinder continuous delivery. For example, a feature that's ready for release normally must go through a change advisory board. This can delay the release for up to four days. If a feature takes only a few days from conception to being ready for release, this four-day period accounts for too much of the feature's total cycle time.	-	Research is needed to identify these processes (covering areas of business, software development, operations, and so on) and develop and verify alternatives that suit continuous delivery.
3	1	Technical	No comprehensive continuous delivery platform tool	A robust, out-of-the-box, comprehensive, and yet highly customizable solution for continuous delivery doesn't exist yet.	So, we developed our own solution, which was costly. When we're building the continuous delivery platform, we use many different tools and technologies as building blocks. Avoiding vendor lock-in is challenging.	Work on developing widely accepted standards, de▾fining open APIs, and building an active plugin ecosystem will help alleviate the challenge.
4	1	Technical	Applications that are not suitable for continuous delivery	Dealing with applications that aren't amenable to continuous delivery (for example, large, monolithic applications) is also challenging. A huge number of such applications exist in the industry.	-	Research is needed on understanding their characteristics and identifying and developing the best strategies or practices to tackle them.
5	2	Technical (being 'lean')	Development of large features prevents continuous deployment (working in small batches)	The ability to break up large features into small batches, and to ensure that incomplete features do not get sent to customers, are challenges that Atlassian is actively working to overcome.	Organization adopted the strategy of using 'dark features' when deploying large features. Dark features are features that are too large to develop in a small period of time, and that are instead developed in small batches that only appear visible to the customer when the entire feature has been developed.	While the concept of working in small batches is not new, there is no literature to date that has specified the size of the features that can be deployed using continuous deployment.
6	2	Technical (being 'lean')	Development of bigger change to software prevents continuous deployment (working in small batches)	Another challenge faced by Atlassian related to small batches was incomplete changes. If a small batch of code is developed and deployed to customers, but is not fully complete, customers will potentially see the incomplete change and may interact with the software product in undesirable ways.	For Atlassian, using small batches to deploy low risk features to customers was the starting point and, as their confidence grew, larger features were deployed using dark features.	-
7	2	Social (being 'lean')	Efficiently utilizing shorter feedback loop	To effectively utilise the shorter feedback loop that results from implementing these lean software development and DevOps techniques...being able to analyse data when justifying the deployment of a new feature	...Atlassian has implemented a strategy to monitor customer behaviour through a data analytics platform. By understanding how their customers use their software products, Atlassian can change what software they develop in response to customers' actions, thus eliminating waste and minimising inventory. In addition, Atlassian conducts experiments by developing features and deploying them to certain subsets of customers to test whether these new features are likely to be used by other customers. Deploy minimal versions of features quickly using experiments, then use the results to update or remove depending on the customers' interactions with the feature.	-
8	2	Social (team coordination)	Coordinating multiple teams requires extra effort	...adopting the continuous deployment process "requires involvement of different organisational units in order to fully succeed". This holds true for Atlassian, which has required an increased amount of collaboration to take place between teams for products that adopt continuous deployment.	Engage in an increased amount of planning in terms of planning around deployments with multiple systems that are interdependent.	-
9	2	Social (team experience)	Inexperienced team	Having an experienced team is critical in the successful adoption of continuous deployment.	Integrate the automated deployment of software using continuous deployment into the existing continuous integration workflow of developers to ensure there is no, or a low learning curve.	-
10	2	Social (company-wide effort)	A lack of motivation for adopting continuous deployment	Cross-team collaboration could not be achieved without top-management implementing a strategy to push the need to implement the continuous deployment process. A technical lead from Team A described a challenge in adopting the continuous deployment process as: "It is something that I would find very hard to do just from a, you know, intrinsic motivation like from within the product team, because it touches so many things."	Ensure that top-management implement a strategy to push the need to implement the continuous deployment process. This will make the implementation of the continuous deployment process a goal for specific product teams to achieve.	-
11	2	Social (coordination)	Ambiquity in coordinating of fixing things	Another challenge of team coordination when using continuous deployment was: who should fix, what, and by when?	As described by a software developer from Team A, "there is a much bigger incentive for the team to keep the build green because it does give us the bonus of having it continuously deployed".	-

12	2	Social (changes in responsibilities)	More pressure to software developers	Software developers may feel an increased amount of pressure to have code ready to be deployed immediately. After adopting the continuous deployment process, software developers also become responsible for deployments, which meant that the quality assurance team did not overlook quality of the deployed code. This change in responsibility has placed the burden for code quality on the software developers, which in turn has resulted in increased pressure on these developers.	Atlassian mitigates this issue by improving communications between developers and managers.	Need for a proper management program for adopting the continuous deployment process in a company to ensure that the continuous deployment process is not destructive for teams that adopt it.
13	2	Social (changes in responsibilities)	Changing team roles	Team members have to adapt to different tasks in new roles due to working in a continuous deployment environment.	The team, as a whole, must work closer together to allow the frequent deployment of software to customers without negatively affecting customers.	-
14	2	Social (changes in responsibilities)	Marketing 'versionless' product	Marketing continuous deployment products requires marketing a versionless product, which requires alternative marketing strategies.	Blog a product's changes.	-
15	2	Social (changes in responsibilities)	Technical product writing	Maintaining multiple documentation for each product which may have multiple offerings (e.g. a customer-hosted version and a versionless Atlassian-hosted online version), where features may be released in one offering, but not in the other.	Use Atlassian's own product, Confluence (which is a wiki), to document software products.	-
16	2	Technical	Product quality may decrease	The quality of the software product may decrease since bugs may slip through and be deployed to customers since deployments occur more frequently.	This challenge is outweighed by the benefits of continuous deployment. It is mitigated by fixing bugs quickly, and by roll back software to a previous version if any code changes leave the software in a less than fully functioning state.	-
17	2	Technical	Testing	Having production quality tests and maintaining the process of 'code review'	Adopt diligent testing, i.e. ensure testing is thorough since deploying software is dependent on tests passing on the continuous integration server.	-
18	2	Social	Customer policies	Nonetheless, since it is easier to deploy bugs when using to deploy software, larger companies are seemingly hesitant to purchase such software. For example, in some cases, large companies have internal policies which state that they cannot use software that is not thoroughly tested.	-	-
19	2	Social	Customers may not notice the newly added features	Continuous deployment enables software products to be constantly updated, but it does not assist in introducing these updates to customers. One product manager from Team B found it especially important, ''you know we deploy continuously and like it just goes over their [a customer's] head, so feature discovery is actually, I think, our biggest challenge right now for customers''.	Show the new features in blogs that can be viewed by customers.	-
20	2	Social	Customer adoption – Not all customers are pleased to receive updates of features.	Atlassian has found it difficult to show customers when they have deployed new features. Atlassian has found when using continuous deployment that not all of their customers welcome this method of development. A software developer from Team B commented that ''...it's affecting them negatively a little bit, but it's only a small subset of customers, because just in general it runs fine''. This has inhibited Atlassian from being able to demonstrate to customers the added value of the ability to introduce new features into their OnDemand software products.	Apply continuous deployment for non-business critical software development	-
21	2	Social (process)	No process documentation	A lack of understanding of the continuous deployment process by novice developers due to inconsistent documentation and a lack of industry standards. The Atlassian developers also pointed out that there was little or no process documentation associated with the continuous deployment process in their teams. However, in most cases they also stated that they did not feel that there were any changes to their original workflows when their teams adopted continuous deployment.	Adopt 'social rules' which must be adhered to when deploying software	-
22	2	Technical	Source code control	Having one single branch of code to maintain one working version of the software product in a continuous deployment environment.	Develop each feature for a software product on a separate branch and, when completed, merge that feature into the main branch of code.	-
23	2	Technical	Changing database schemas	Minor changes in code create unplanned changes in database schema. Changing database schemas issues go beyond internal development and extend to partners who provide plugins for Atlassian.	Ensure there is rigorous testing around the database of a software product to help prevent any deployments affecting the database. Roll backs can also help reset issues.	-
24	2	Technical	Partner plugins integration issues	Only a small fraction of plugins are available due to code integration issues involved with a continuously changing software product. By constantly changing versions and the code of their continuous deployment based software, some partner plugins no longer work with Atlassian's latest software updates, and it has become infeasible for partners to keep updating their plugins to work with Atlassian's constant updates.	Offer customers a smaller fraction of plugins that are known to be compatible with the Atlassian product.	-

25	2	Technical	Infrastructural requirements	It requires proper hardware and software to handle the continuous deployment process and its related problems e.g. cross-product dependencies.	Provide more hardware resources to a product's continuous integration servers to allow the software product to be continuously integrated as often as necessary, thus allowing the software product to be deployed at any time.	-
26	2	Technical	Cross-product dependencies	To handle cross-product dependencies, Atlassian has changed its product upgrading process by removing the product that is being upgraded from OnDemand, and then re-integrating the product once it has been upgraded. Due to the complexity of cross-product dependencies, several interviewees believed this was the main challenge for the company when adopting continuous deployment. A software developer from Team A explained, ''we had to overcome a lot of obstacles because the OnDemand product is kind of a set of tightly coupled applications''.	Provide more hardware resources to a product's continuous integration servers to allow the software product to be continuously integrated as often as necessary, thus allowing the software product to be deployed at any time.	-
27	2	Technical	Seamless upgrades	Complications in implementation of seamless upgrades due to resource limitations, zero downtime deployment and customer data preservation.	Use two parallel running systems (the old and the new system) to upgrade the user to the next version of software.	-
28	2	Technical	Continuous integration process challenges	Challenges with the continuous integration process include merge conflicts, having Atlassian's software in an everready state of deployment, and determining the deployability status of software builds.	Employ a strategy of stopping the entire team (i.e. as many people as necessary) from doing what they are doing to fix issues to make sure the software is in an ever-ready state of being.	-
29	3	Technical	No packages available to satisfy dependencies	...it is common to encounter a situation where the dependencies of a library are not satisfied by any package that is available in the repositories of the particular Linux version required to be used. As an example, it is quite challenging to install a new version of the PETSc package together with a new version of the g++ compiler that supports the C++11 standard on a RHEL 5 machine.	When a package for a specific tool or library is not available, we create custom ones and maintain a private repository for those packages.	-
30	3	Technical	Outdated environments at clients	In many cases, they (clients) do not have the freedom to choose their operating system. They have workstations that are tightly controlled by corporate IT who are extremely conservative, having them use very outdated versions of Linux.	One platform, usually the one that the client is using, is chosen as the base for this automation. For instance, if the client uses RHEL5 then, we use Vagrant to set CentOs5 virtual machines that are binary compatible to test our environment automation and the deployment and execution of our software.	-
31	3	Technical	?	Depending on the hardware, numerical calculations can be carried out either by CPUs, GPUs or other co-processors such as the Xeon Phi 11 or FPGAs (Field Programmable Gate Arrays). The information about the exact execution infrastructure would only be available at deployment so the IaC scripts would have to select the correct variation and apply device specific configurations, maybe even recompile code to run optimally on the local system.	-	Extending the software product line approach to IaC would allow us to approach the development of infrastructure code in a more systematic and reusable way.
32	3	Technical	Testability of outputs	...in physical simulation the results are often quite complex and it is not feasible to calculate them by hand. Even experts are not able to say with precision if a result is right or wrong, they are able only to assess if the result is consistent with the theory. In other words, there is a continuous spectrum of results where the limits between right and wrong are fuzzy. In cognitive systems, the situation can be even more challenging.	-	-
33	4	Organizational	Management commitment	Establishing a continuous delivery pipeline is a time consuming and costly endeavor. Thus, first of all we would like to emphasize that such a step can only be made with the firm commitment of a sponsor from the top management. Furthermore, when major parts of the pipeline are established, development is initially – for a short period of time – slowed down before the investment starts to pay off. The commitment of the top management is also crucial to survive this phase.	-	-
34	5	Organizational	Lack of business model	-	-	-
35	5	-	Test automation	-	More investments in test automation and CI build system	-
36	5	-	Lack of practices	Lack of common practices for continuous deployment	-	-
37	6	-	Adjusting stakeholders to shorter feedback cycle	How to ajust and align internal and external stakeholders to shorter development cycles	-	-
38	6	-	Inadequate level of test automation	-	-	-
39	7	-	Pre-defined milestones in other units	Some company functions still work according to pre-defined milestones, and those functions still support a six-month release cycle	-	-

40	7	-	PC based infrastructure	-	...current product architecture must be updated from a PC platform to a virtualized cloud computing platform.	-
41	8	-	Perfomance- and safety critical system blocks continuous deployment	-	-	-
42	10	Culture of continuous improvement	Too much specialization	Module teams tend to require some level of specialisation to develop software modules for the lower layers of the system stack, particularly closer to the hardware. People working for a certain module don't know enough what's happening outside their modules. "I would prefer that we would have only cross-functional teams that will work in end-to-end solution in the bigger picture. … There's too little interaction between the different modules. Cross-functional teams with more responsibility on the end-to-end aspect of the feature for each team would be a great benefit." (Senior developer, Company B)	Development of new features in silos of modules requires effective communication and underscores the importance of agile software development in a large scale context and CI practices as coordination mechanisms both within the team and across the team when building an end-to-end product	-
43	11	Configuration management of test environments	Long acceptance tests	A factory acceptance test takes several months after start-up after the completion of system development	-	-
44	10 11	Configuration management of test environments	Limited view on configuration	Companies have limited view of how customer environments are configured.	-	-
45	10 11	Configuration management of test environments	Difficulty to construct test environments	For these companies, each customer environment has its own configuration, with several elements provided and configured by other vendors. That creates complexity and makes it difficult for companies to repeatedly and reliably construct test environments that are also representative of a wide range of possible customer configurations.	-	-
46	9 10 11 12	Configuration management of test environments	Many faults in acceptance testing phase	Tend to discover many faults in the system during customer acceptance tests in the field. This is due to having a variety of tests performed in a similarly configured test environment.	-	-
47	10	Configuration management of test environments	Lack of fully automated acceptance tests	Lack of fully automated acceptance test coverage, requiring a lot of time to be spent on manual acceptance and regression testing.	-	-
48	9 10 11 12	Deployment process automation	Deploying requires numerous activities	In the embedded systems domain, deploying new software functionality to customer sites involves numerous activities that also require gaining consent from customers. In most cases, the complex systems cannot be updated easily because specific versions of software need to be updated in multiple places.	-	-
49	11	Deployment process automation	Long lifecycle of products and thus support for multiple versions	Embedded systems also have a long lifecycle with large amounts of legacy code that cannot be updated easily. Very often, customers acquire new product features for their existing systems, which may have old software versions from releases made several years earlier. The long lifecycle of products requires companies to ensure high compatibility between new software features and the existing software features at customer sites. Ensuring system compatibility is a challenging task.	-	-
50	9 10 11 12	Deployment process automation	Customers do not want to upgrade	...that customers do not like to upgrade existing systems if they are working correctly.	-	-
51	11	Deployment process automation	Lack of technology to automatically deploy without downtime	...we observed a lack of technology to automatically deploy new features repeatedly and reliably without downtime in complex and critical embedded systems.	-	-
52	9 10 11 12	Monitoring in the production environment	Not possible to monitor customer systems	It is impossible to monitor customer systems, especially after product launches, unless the customer sends direct reports to the company. The four companies in the study rarely had access to monitored data from customer systems, except during customer trial tests or when doing fault diagnostics during maintenance.	-	-
53	13	Social	Resistance to change	One interviewee mentioned that the company's management downright resisted changes to the current development practices and was unwilling to switch to continuous deployment. Several other interviewees also said they felt their organizational culture wasn't particularly receptive toward new ideas and continuous deployment, which was seen as a challenge.	-	-
54	13	Social	Customer preferences	An interviewee from a telecom provider mentioned that the company tried to ramp up its release frequency from quarterly releases but that the receiving end wasn't prepared to handle a shorter release cycle. Eventually, dissuaded by the experience, the company reverted to its previous schedule.	-	-

55	14	Domain constraint	Compliance to standards	The interviewee from company A pointed out that the company emphasized compliance to national or international medical standards. The company needed to ensure that changes had no adverse side effects, guaranteeing patient and user safety.	-	-
56	15	Domain constraint	No zero-downtime deployment capability	The interviewee from company B, which works on automation control systems, reported that the company might have to stop the whole process for a day or so or run the control system updates during weekends. Such a setting prohibits instant deployment because each update must be scheduled. Also, the costs related to factory downtime can make automation software providers think twice before pushing a new release to production systems.	-	-
57	13	Domain constraint	Diversity of clients	Deploying software releases directly to network devices is challenging because network configurations might vary among clients.	-	-
58	13	Domain constraint	Third party distribution channels	The distribution channels that provide software to customers might also slow down deployment. When a software development company doesn't fully control the distribution channel, a third party is responsible—at least partly—for making a product available.	-	-
59	16	Domain constraint	Third party quality assurance	The interviewee from company L, which develops mobile games, noted that its company's releases couldn't be instant because an application store took a week to review the submissions and publish the product.	-	-
60	13	Developer trust and confidence	Lack of automated testing	According to the interviews, many companies saw the need to improve their automated testing; they viewed the existence and volume of such testing as paramount for continuous deployment. The lack of automated testing was a major barrier the companies faced on their path to more continuous software releases.	-	-
61	13	Developer trust and confidence	Automated user interface tests	Some companies, especially those working with the quirks of Web browsers or mobile games, found automating user interface tests particularly challenging.	-	-
62	13	Developer trust and confidence	Developer responsibility	Also, developers' reputations are on the line: deploying a broken build to customers could strain the relationship between parties and create an unwanted user experience. Any lack of confidence in an application's quality is amplified by the knowledge that any and all changes are immediately deployed.	-	-
63	17	Developer trust and confidence	Lack of resources	The interviewee from company G, which develops Web-based products, noted that the company simply didn't have the time to update the existing setup to better support continuous deployment. In cases such as this, developing software trumps infrastructure setup and configuration.	-	-
64	18	Legacy code considerations	Legacy system integrations	Although legacy code wasn't the most common barrier, the interviewee from company I, which develops Web products, mentioned that it was a challenge for one project. The company couldn't employ fully continuous deployment because of legacy system integration.	-	-
65	19	Duration, Size, and Structure	Long test execution	The interviewee from company J, which develops Web frameworks, mentioned that the full test suite took a good hour and a half to finish. Under these circumstances, instant releases are harder to achieve because the release process takes at least as long as test execution, given that all the tests run after the code changes are committed.	-	-
66	13	Duration, Size, and Structure	Long execution of compile, build and creating packages	The code base's size also affects the time to deploy software releases. Company K, which works with a user interface framework, had to compile, build, and assemble deployable packages out of 8 Gbytes of source code. The interviewees from companies A and F also identified the code base's size and complexity as challenges to continuous deployment.	-	-
67	20	Duration, Size, and Structure	Dependencies to other projects	One project he was working on had been split to subprojects, each of which had to be compiled and built separately before a release could occur. In this project, the interrelations between the subprojects weren't restricted to building. Development tasks could have dependencies from one subproject to another, and parts of the project could be at different development stages.	-	-

68	20	Different Development and Production Environments	Configuration differences between environments	Company F’s interviewee commented that the company did extra checks because the production system used a different database than was used in development. Also, the code could work differently in the production system, causing unforeseen defects.	-	-
69	21	Manual and Nonfunctional Testing	Perfomance testing requires lots of hardware for embedded software	For network infrastructure services, load testing can be resource intensive and requires plenty of hardware, as the interviewee from company C, a large telecom provider, noted.	-	-
70	13	Manual and Nonfunctional Testing	Explorative testing	...besides the possible performance and security tests, they performed a round of manual or exploratory testing, trying out the product on actual devices with or without a test plan.	-	-
71	16	Manual and Nonfunctional Testing	Acceptance testing requires lots of different devices	The interviewee from company L, which develops games, stated that the multitude of devices and fragmented base of hardware partly stopped the company from doing continuous releases.	-	-
72	19	Manual and Nonfunctional Testing	Automated acceptance testing do not catch all defects	Company E’s interviewee explained that the company’s product had a public application interface and was used on various devices. Although the release frequency was rapid with weekly releases, the company had to alert a third-party organization a day before the release date. That organization then tested the product on different devices before release.The company couldn’t skip this phase because exploratory testing could reveal defects that automated tests didn’t catch.	-	-
73	22	-	Sales and marketing were not quite sure when anything would be released	So, since Sales and Marketing could not rely on feature a, b and c coming out with the next release in x weeks, they asked the product owner. But the product owner did not really know exactly when the next feature was going to come out. They would give vague and unsatisfying answers to Sales and Marketing when asked. As you can imagine this was not great for their relationships.	We eventually learned, that we need to increase the level of transparency with Sales and Marketing. We found it easier to maintain a calendar communication cadence with stakeholders. As a company we have annual planning and quarterly planning. Each product line has a monthly council to gather stakeholder feedback and to keep them current on what is being being built and what is coming next.	-
74	22	-	Ignoring the test suite	We were having problems deploying a significant code change because our test deployment kept failing in the pipeline. We decided that it was because there was bad data on the test system. This was incorrect. The actual problem was that our new code was incompatible with the old data format. To get the tests to pass we deleted all the historical test data and the new code made it out to production. Then everything stopped working... the new code was not compatible with the production data.	The lesson learned from this is that your tests, gates and checks are there for a reason. Always run them with old data and never circumvent the process without a clear understanding of what you are doing. The ability to release quickly does not mean you should rush without full understanding.	-
75	22	-	Added responsibility to developers	The transition to a continuous delivery model can make stakeholders uncomfortable. The ability to push code directly to production does come with added responsibility - developers must be diligent in writing tests and monitoring the system.	-	-
76	22	-	Fear of unemployment among QA team	Members of our QA team were particularly worried about not having time to test code before it shipped. They needed to switch to a mindset that trusts the automatic tests to perform this task. This naturally led to the fear that they would be automated out of employment.	As mentioned earlier, before a story is coded we execute a “test planning” phase. This needs a professional QA mindset. QA does not just appear blindly at this stage. They will have spent hours reading stories in the backlog, performing research into the background of the story and brain storming testing angles. With the decreased load from test automation some of our QA team have adopted a TestOps role. TestOps monitor our application with tools like Splunk and Ganglia to discover changes deep within the system. DevOps is a commonly talked about role in our community and we suspect that TestOps will become a familiar role in engineering teams of the future.	-
77	22	-	Adding concurrency to already implemented GUI testing suite was hard	At one time our GUI test suite took nine hours to run. Since a passing test run is a prerequisite for a production release this mean the fastest we can ever release is nine hours. When test runs take this long people begin to ignore them.	To resolve this one must break down the suite, optimize long running tests and parallelize. Our advice here, learnt from pain, is to thread early and often. Adding concurrency de facto is far more time-consuming than working on it up front.	-

78	22	-	Flaky tests	If tests are nondeterministic people will stop listening. Tests that pass or fail based on race conditions or test execution ordering are essentially useless. We call these ”flaky” tests. Flaky tests cannot be tolerated. We fell foul of a set of flaky tests when making a cross cutting change to improve performance of our application. The GUI tests were still slow at this time and failed on an nine hour overnight run. We ignored the failing GUI test run because they regularly flaked out. After a day of high stress fixing the defective production systems we ran a “PER” retrospective and realized that the GUI tests were indicating this problem.	This lesson encouraged us to invest heavily in our GUI test framework. Flaky tests cannot be tolerated. We wrote a flaky finder application that hammers new test to see if they contain determinism or concurrency bugs and, more recently, we built Flowdock integrations with bots that report build health and can be queried for statistics on the test suites.	-
79	22	-	Business runs on dates, not on continuous flow	The business still runs on dates and dollars. We set annual goals, quarterly goals and target market events. But if the delivery teams are running continuous flow how do you reconcile this?	The product team still maintains an annual roadmap but the future is less defined now. By this we mean, the roadmap is detailed for the current quarter, defined for the next and has candidate initiatives for the rest of the year. We still have monthly product council meetings where we discuss recently released work, work in progress and take input on the near-term roadmap. All our planning and in progress work is organized using Rally Software’s product. We have built dashboards that make it easy for stakeholders to check in on progress and provide vision to what features are coming soon. Using historical data on team throughput we can estimate with a reasonable accuracy when a feature will be shipped.	-
80	23	Organizational	Network configuration and upgrade complexity	The interviewees at company D all mention the complexity of the network and the many different configurations that their customers have. A very common challenge is when a customer wants a new feature but has an old version of the product to which this new feature has to be configured. Similarly, an upgrade of any kind is considered stressful by customers, something that is highlighted by the release manager: “it is more difficult to guarantee minimal network impact if the configuration of the product is complex”. From the interviews it is clear that customers still regard upgrades and new features as a challenge due to the risk of interfering with legacy.	-	In transitioning towards continuous deployment the external action is to develop a new engagement model with lead customers to facilitate for continuous deployment
81	23	Organizational	Internal verification loop is too long	Another barrier is the internal verification loop which needs to be shortened and automated in order to meet up with the requirements that continuous deployment raises. As mentioned by one of the product line maintenance managers, more automated tests are needed in order to increase speed and frequency of delivery.	In transitioning towards continuous deployment the internal action is to involve product management in the short, agile cycle of product development.	-
82	23	Organizational	No information about build quality status	Also, several interviewees highlight the importance of improving the quality on each build and to increase awareness on what effect each build has on the overall software package. In this, the teams would benefit from knowing more about the quality status of the development projects, i.e. the current quality of features, the number of errors etc. If such knowledge could be better established, teams could respond faster and act more pro-actively towards customers.	-	-
83	24	-	Lack of trust in quality	As mentioned by one of the product line maintenance managers in company D, the development teams would benefit from knowing more about the status of the development projects, i.e. the current quality of features, the number of errors etc. If such knowledge could be better established, teams could respond faster and act more pro-actively towards customers.	-	Mechanisms to roll back unsuccessful deployments, and to deploy components of the system rather than the entire system, are needed.
84	24	-	Interface towards customers	Also, the interface towards customers is considered a barrier since adjustments of business models are needed. In our study, a number of interviewees mention the difficulty in acting agile and promote continuous deployment of functionality at the same time as the business model gives a conservative impression in promoting fixed releases and fixed price models assuming that requirements are frozen upfront.	Lead customers need to be identified with whom the R&D organization can start building a continuous deployment culture and capability. These lead customers serve as role models to other customers. The business model needs to be reviewed so that there are mechanisms that support continuous deployment of functionality. All corporate functions, such as for example the release organization, need to be aligned with the R&D organization in order to facilitate continuous deployment.	-

85	25	-	Resistance to release experimental functionality	Besides benefits, the barriers relate to the resistance that might be the case for releasing “experimental functionality” to customers. In the companies, the tradition of rigorous test and validation processes is well anchored and the culture is that all testing and validation activities are to be performed before functionality is released to any customer.	-	Customers need to be involved in providing early feedback on new functionality. To collect, analyze and capitalize on customer feedback is important and the establishment of mechanisms that allow for quick response to customers is the major initiative to undertake. Infrastructures need to be established in order to support run-time variation of functionality that allow for innovation experiments with customers. Also, a variety of data collection mechanisms are necessary which means an extension of current architectures.
86	25	-	Resistance to buy experimental functionality	Also, from a customer perspective, the idea of having partially developed functionality released with the intention to “experiment” might be a challenging task to pursue.	-	Business models and pricing models need to support short-cycle innovation processes based on customer usage data.
87	26	-	Difficulties in establishing efficient rollback mechanisms	The challenge is to have efficient rollback mechanisms to manage potential problems with the deployment of new software. Our respondents report on a situation in which the only way to rollback is to de-grade to the previous system version.	-	While this is common practice it is not considered a sufficient mechanism when moving to more frequent deployment of software.
88	27	Technical	Downtime is critical for certain customers	According to the Dialog product owner, downtime causes end-users being unabale to perform their job. Downtime can also interrupt ongoing customer tasks, possibly losing critical data in the progress.	Currently the deployment time for both projects is negotiated with the customer to prevent these cases, and the version deployments are done when the system can be closed for a short period of time.	-
89	27	Technical	Automated testing has to be built on top of a matured software product	The developers perceive automated testing and test environments to be the largest technical task. The developers state that building a sufficient test automation is a very laborious process especially due to the maturity of the software, and are concerned with the maintainability of the test suite.	-	-
90	27	Technical	Prioritizing of automated tests to be implemented for matured software product is not clear	The management is not sure what to test with automatic acceptance testing to validate a version.	-	-
91	27	Technical	Software is often integrated to multiple third party applications	Both of the case company’s software products are integrated to various third party applications and APIs. Changes to the interfaces communicating with these applications must be planned and discussed in advance. Based on the interview results, automatically updating the integrations requires an unduly amount of work considering the results.	-	-
92	27	Technical	Software is often accompanied by multiple external components	It is also common for B2B applications to have external components that have to be configured when the software is installed or the APIs to these components changed. The configurations for these external components either have to be manually updated, or automated as well.	-	-
93	27	Technical	There exists multiple different configurations due to having multiple customers with different specifications	One of the main differences between B2B and B2C domains is the production environment. Both of the case company’s products are used in multiple different customer environments. This introduces a problem of managing different configurations per customer environment and software instance.	-	-
94	27	Technical	Transferring the software product to diverse customer-owned environments requires different deployment configurations	One of the main differences between B2B and B2C domains is the production environment. Both of the case company’s products are used in multiple different customer environments. This introduces a problem of managing different configurations per customer environment and software instance.	-	-
95	27	Procedural	User acceptance testing environment is a requisite for production release	The basic deployment pipeline in the case company first includes a deploy to a user acceptance testing server, which is then tested manually by either the team or the customer. Only after the version has been acceptance tested and validated to work properly, can the production version be released. Continuous deployment to production is seen very risky due to the applications playing a major role in running the customers business.	-	-

96	27	Procedural	The development process drifts towards small feature branches from long-lived feature branches	Both of the case company’s products are developed with a branching model, where feature branches are first thoroughly developed and then integrated to the master branch. With continuous delivery the long-lived feature branches should be changed to short-lived and relatively small feature branches to allow exposing new functionality faster to the customers, and receive feedback faster. While the small feature branches might be common for companies with a relatively new software products, companies that have been developing products for a long time might be more devoted to the practice of long-lived feature branches.	-	-
97	27	Procedural	Triggering the compilation and deployment of a modular project to maintain integrity is hard	The software applications in B2B often are large and modular applications, as is the case in the case company. The point when a deployment is triggered has to be designed to maintain the integrity of the application. As the deployment process is currently manually triggered by first releasing a version, a suitable time can be chosen each time. When a production deployment is triggered in continuous delivery, each module has to be in the correct state in order to produce a coherent version.	-	-
98	27	Procedural	The software has to be deployed to multiple customers	Both of the case company’s products are used by multiple customers, each having their own environments. As the deployments are currently done manually, the customers receiving each deployment can be manually chosen. However, with a continuous delivery process whenever a feature or a new release is ready to be delivered, it can either be deployed to a single customer or to every customer.	-	-
99	27	Procedural	Versioning is affected by having different customer profiles of the product	Multiple customer environments affects versioning of the software product. In the case company, each customer has a unique configuration of the product, with possibly different versions of certain components. According to Jan Bosch, in an Innovation Experiment System environment only a single version exists: the currently deployed one. Other versions are retired and play no role [3]. However, with multiple environments, multiple different versions of the software are necessary at least in the early phase.	-	-
100	27	Procedural	Responsibility of deploying moves towards developers	Continuous delivery also drifts response towards the developer, and the developers decide what is ready to be released. Currently in the case company the product owners and team leaders are responsible for negotiating the deployment date with the customer, and they also inform the developers that a new version is required. If the developer can single-handedly deploy a feature, the management can quickly lose track on the features available to customers. This also requires the developers to deeply understand the details of the version control system and automated testing.	-	-
101	27	Procedural	Management and sales loses track of versions	Due to increased developer responsibility and varying interval of version updates continuous delivery causes, a team leader expresses concern that the delivery process complicates tracking when deployments are performed, and when features are finished. This also concerns other parties working in the cus- tomer interface, such as sales.	-	-
102	27	Customer	Some customers are reluctant towards new versions	Some customers of the case company are reluctant towards new releases. One of the reasons for this reluctancy is that new releases occasionally contain new bugs.	-	-
103	27	Customer	Customers are trained to use a certain version, and modifications confuse the users	In the case company, customers have been trained to perform certain tasks with a certain user interface. The customer might perform these tasks daily, once every two weeks or even less frequently. If the UI changes often, the customers feel lost and initially take more time to perform the tasks. This causes frustration in the users, and visible changes generally increases the reluctancy customers have towards new versions, unless the changes are significantly improving the user experience.	“The user interface should be easy to use. Now it’s relatively hard to learn. If customers have just learned to perform a task, and we change the UI, the feedback is terrible.” -Product Owner	-

104	27	Customer	Changelogs are especially important, since as versions are released faster the cus- tomers become less aware on what has changed	Listing the changed features in changelog entries is especially important when releases are made more often. While the changes become smaller the faster versions are released, customers become less aware of when the version will be updated and when features have changed. Currently the version deployments are negotiated with the customers, and when the deployments are made more often, discussions regarding version releases may be reduced or even ceased.	-	-
105	27	Customer	Pilot customer is required for developing the continuous delivery process	A way to identify the best practices in continuous delivery is to develop the continuous delivery process with a pilot customer. Pilot customer is a company willing to help the company to quickly learn what works and what needs to be improved. The interviewees expressed a desire to first test the continuous delivery process with a single customer that is willing to receive updates in a continuous manner, since the engagement model inevitably differs from the current model.	-	-
106	27	Customer	Acceptance testing is performed by both the company and the customers, and requires a lot of resources from the customers	The acceptance testing is performed in varying ways. Some customers require to perform manual acceptance testing before the product can be deployed into production. Other customers trust the developers to perform the acceptance testing. The technical implementation therefore should make it possible to con- tinuously deploy versions to the user acceptance testing environment, and by the push of a button to the production environment. However, if the versions are deployed to user acceptance testing environment very often, customers might feel encumbered by the amount of required testing.	-	-
107	27	Customer	Production deployment schedule has to be negotiated with the customer	The customers also have to be informed whenever a new version is available to the user acceptance testing environment.	-	-
108	27	Customer	Ongoing critical tasks by users cannot be interrupted by downtime	Customers might be using the software when a new version is deployed, and the deployment process shouldn't interfere with ongoing usage.	-	-
109	28	Technological Enablers	Monolithic architecture	The architecture of the system is closely coupled with how the system is developed, tested, and deployed for use. A monolithic architecture can be a bottleneck to rapid continuous build, test, and deployment. Transforming the architecture or improving the capability of the continuous deployment system is needed to overcome this impediment.	It was mentioned during the interviews that a more modularized architecture allows for upgrading smaller parts of the system independently and, for example, shorter wait times for build, test, and deployment results. As the interviews suggest, overcoming this impediment can be particularly challenging if the value of such technical change is not evident. Without clear value in architectural improvements, these are easily postponed, for example, in favor of work on new software features.	-
110	28	Technological Enablers	Development and testing environments do not reflect production environments	Some interviewees perceived differences between development, testing, and production environments as a possible impediment. Difficulty to simulate production environments in testing environments create a risk that software is not properly validated before it is deployed to production. Differences between development, testing, and production environments can be problematic not only for continuous delivery and deployment, but also for sharing responsibilities.	-	-
111	28	Technological Enablers	Multiple production environments	Based on the perception of some of the interviewed people, multiple production environments and differences between them could be a possible impediment for continuous delivery. Different needs of environments cause complexity. Automating and having common tools and processes becomes challenging. Even different access rights can cause issues. The main perceived difficulties that multiple production environments cause are related to deployments and configurations.		-
112	28	Technological Enablers	Multiple production environments	Even different access rights can cause issues.	In the interviews, it was for example stated that when fixing production problems, it is essential to have free enough access.	
113	11	Culture of continuous improvement	Propagation of rapid changes across company	As embedded systems are very complex, there is oftentimes no mechanism to propagate rapid changes made by one team to other teams across the company on a continuous basis.	To enable fast feedback loops, systems development needs to encompass new ways of working whereby teams bear the responsibility for accepting a continuous flow of rapid changes and also propagating their changes rapidly to other teams.	-