Department of Computer Science

Adoption problems of modern release engineering practices

Eero Laukkanen

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 220/2017

Adoption problems of modern release engineering practices

Eero Laukkanen

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 of the school on 21 December 2017 at 12.

Aalto University
School of Science
Department of Computer Science
Software Process Research Group

Supervising professor
Prof. Casper Lassenius, Aalto University, Finland

Thesis advisors
D.Sc. (Tech.) Juha Itkonen, Nitor Delta Oy, Finland
Prof. Maria Paasivaara, IT University of Copenhagen, Denmark

Preliminary examiners
Prof. Tommi Mikkonen, University of Helsinki, Finland
Prof. Dag Sjøberg, University of Oslo, Norway

Opponent
Prof. Jan Bosch, Chalmers University of Technology, Sweden

Aalto University publication series
DOCTORAL DISSERTATIONS 220/2017

ISBN 978-952-60-7713-0 (printed)
ISBN 978-952-60-7714-7 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-7714-7

Unigrafia Oy
Helsinki 2017

Finland

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Eero Laukkanen

Name of the doctoral dissertation
Adoption problems of modern release engineering practices

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 220/2017

Field of research Software Engineering

Manuscript submitted 13 June 2017 **Date of the defence** 21 December 2017

Permission to publish granted (date) 21 August 2017 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

Release engineering means the process of bringing the individual changes made to a software system to the end users of the software with high quality. The modern release engineering practices emphasize using build, test and deployment automation and facilitating collaboration across functional boundaries, so that it is possible to achieve both speed and quality in the release engineering process. While some companies have been successful in adopting modern release engineering practices, other companies have found the adoption to be problematic.

In this dissertation, we aim to understand what prevents organizations from adopting modern release engineering practices. First, we conducted a systematic literature review to survey the literature about the adoption problems, their causes and solutions. In addition, we conducted four case studies which included qualitative interviews and software repository mining as data collection methods. In the first case study, we investigated the adoption problems in a distributed organization. The case study was extended in a follow-up case study, in order to see how the release stabilization period could be reduced after the adoption efforts. In the third case study, we focused on the consequences of the stage-gate development process and how it explained the adoption problems. Finally, we compared two organizations with different organizational contexts working with similar products in the fourth case study, in order to compare the effects of different organizational contexts.

This dissertation identifies, that adopting modern release engineering practices decreases the time needed for stabilizing a software system before it can be deployed. Problems during the adoption of modern release engineering practices are categorized under problem themes of technical adoption and social adoption. Technical adoption problems include build automation (slow, complex or inflexible build), test automation (slow, unreliable or insufficent tests) and deployment automation (slow, complex or unreliable deployment) problems, and social adoption problems include organizational adoption (lack of resources or coordination) and individual adoption (lack of motivation or experience) problems. These primary problems can be explained with three identified explanations: system design (system not testable or deployable) explains technical adoption problems, organizational distribution (more difficult communication, motivation and coordination) explains social adoption problems and limited resources explain both adoption problem themes. Organizations can use the results of the dissertation to design their software processes and practices accordingly to suit modern release engineering practices.

Keywords software engineering, release engineering, adoption, continuous integration, continuous delivery, continuous deployment, devops, case study, systematic literature review

ISBN (printed) 978-952-60-7713-0 **ISBN (pdf)** 978-952-60-7714-7

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2017

Pages 169 **urn** http://urn.fi/URN:ISBN:978-952-60-7714-7

Tekijä
Eero Laukkanen

Väitöskirjan nimi
Modernien julkaisutekniikoiden käyttöönotto-ongelmat

Julkaisija Perustieteiden korkeakoulu

Yksikkö Tietotekniikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 220/2017

Tutkimusala Ohjelmistotuotanto

Käsikirjoituksen pvm 13.06.2017 **Väitöspäivä** 21.12.2017

Julkaisuluvan myöntämispäivä 21.08.2017 **Kieli** Englanti

☐ **Monografia** ☒ **Artikkeliväitöskirja** ☐ **Esseeväitöskirja**

Tiivistelmä

Ohjelmistojen julkaisutekniikalla tarkoitetaan prosessia, jolla yksittäiset ohjelmistoon tehdyt muutokset julkaistaan loppukäyttäjille hyvällä laadulla. Modernit julkaisutekniikat korostavat käännös-, testi- ja asennusautomaatiota ja yhteistyötä funktionaalisten rajojen yli, jotta julkaisuprosessi olisi nopea ja takaisi hyvän laadun. Vaikka jotkin yritykset ovat onnistuneet modernien julkaisutekniikoiden käyttöönotossa, toiset ovat kohdanneet tekniikoiden omaksumisessa ongelmia.

Tässä väitöskirjassa pyrimme ymmärtämään mikä estää organisaatioita omaksumasta moderneja julkaisutekniikoita. Toteutimme ensin systemaattisen kirjallisuuskatsauksen, jossa kartoitimme ongelmia, ongelmien syitä ja ratkaisuja modernien julkaisutekniikoiden omaksumisessa. Tämän jälkeen suoritimme neljä tapaustutkimusta, joissa käytimme laadullisia haastatteluita ja ohjelmistovarastojen louhimista tiedonkeruumenetelminä. Ensimmäisessä tapaustutkimuksessa selvitimme omaksumisongelmia hajautetussa organisaatiossa. Jatkoimme saman organisaation tutkimista toisessa tapaustutkimuksessa, jossa tarkastelimme kuinka julkaisun vakauttamisaikaa voitiin lyhentää omaksumisen jälkeen. Kolmannessa tapaustutkimuksessa tutkimme, kuinka vaiheportti-kehitysprosessin käyttö selittää omaksumisongelmia. Lopuksi neljännessä tapaustutkimuksessa vertailimme kahta organisaatiota, joilla oli erilaiset organisatoriset kontekstit mutta samankaltaiset tuotteet, jotta pystyimme vertailemaan erilaisten organisatoristen kontekstien vaikutusta omaksumisongelmiin.

Tässä väitöskirjassa tunnistettiin, että modernien julkaisutekniikoiden omaksuminen vähentää ohjelmiston vakauttamiseen tarvittavaa aikaa ennen ohjelmiston asennusta. Modernien julkaisutekniikoiden omaksumisongelmat kategorisoitiin teknisiin ja sosiaalisiin ongelmateemoihin. Tekniset omaksumisongelmat sisältävät käännös- (hidas, monimutkainen tai joustamaton käännös), testi- (hidas, epäluotettava tai riittämätön testaus) ja asennusautomaatio-ongelmat (hidas, monimutkainen tai epäluotettava asennus), ja sosiaaliset omaksumisongelmat sisältävät organisatoriset (resurssien ja koordinoinnin puute) ja yksilölliset (motivaation tai kokemuksen puute) omaksumisongelmat. Näitä primäärisiä ongelmia selittävät kolme selitystekijää: ohjelmiston arkkitehtuuri (järjelmä ei testattava tai asennettava) selittää teknisiä omaksumisongelmia, organisatorinen hajautus (vaikeampi kommunikointi, motivointi ja koordinointi) selittää sosiaalisia omaksumisongelmia ja puutteelliset resurssit selittävät molempia ongelmateemoja. Organisaatiot voivat käyttää tämän väitöskirjan tuloksia suunnitellessaan prosessejaan ja käytäntöjään, jotta ne sopivat moderneille julkaisutekniikoille.

Avainsanat ohjelmistotuotanto, julkaisutekniikka, käyttöönotto, jatkuva integrointi, jatkuva toimitus, jatkuva asennus, devops, tapaustutkimus, systemaattinen kirjallisuuskatsaus

ISBN (painettu) 978-952-60-7713-0 **ISBN (pdf)** 978-952-60-7714-7

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Helsinki **Painopaikka** Helsinki **Vuosi** 2017

Sivumäärä 169 **urn** http://urn.fi/URN:ISBN:978-952-60-7714-7

Dedicated to my loving spouse Suvi and our daughter Minea.

Preface

I would like to acknowledge the people who made this book possible.

I thank my colleagues in the Software Process Research Group at Aalto University. Thanks to my supervisor, Professor Casper Lassenius, for providing me inspiration, direction and freedom. Thanks to my instructors, Juha Itkonen and Maria Paasivaara, for discussing the details and giving practical guidance. Thanks to Timo Lehtinen for initial support and for coauthoring a publication. Thanks to the other members of the group, Jari Vanhanen, Ville T. Heikkilä, Lauri Hukkanen and Raoul Udd, for inspiring discussions during, before and after our lunch breaks.

Thanks to Nokia, Ericsson and other organizations who were the research subjects in the dissertation. Thanks to Teemu Arvonen, who was the chief case informant at Ericsson and coauthored two publications. All the intervieweees who were interviewed receive my sincere thanks.

Thanks to the people who reviewed the work. Thanks to Tommi Mikkonen and Dag Sjøberg for performing the preliminary examination. Thanks to Jan Bosch for being the opponent of the dissertation.

This work was financially supported by TEKES (the Finnish Funding Agency for Innovation) as part of the Need for Speed research program of DIMECC (Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business).

Finally, I could not have done this book without my spouse Suvi and our daughter Minea. I thank you for the unlimited patience, love and support.

Rekola, Vantaa, October 19, 2017,

Eero Laukkanen

Contents

List of publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Eero Laukkanen, Juha Itkonen, Casper Lassenius. Problems, causes and solutions when adopting continuous delivery——A systematic literature review. *Information and Software Technology*, 82, 55–79, February 2017.

II Eero Laukkanen, Maria Paasivaara, Teemu Arvonen. Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study. In *Proceedings of the 2015 Agile Conference*, Washington, D.C., 11–20, August 2015.

III Eero Laukkanen, Maria Paasivaara, Juha Itkonen, Casper Lassenius, Teemu Arvonen. Towards Continuous Delivery by Reducing the Feature Freeze Period: A Case Study. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, Buenos Aires, Argentina, 23–32, May 2017.

IV Eero Laukkanen, Timo O.A. Lehtinen, Juha Itkonen, Maria Paasivaara, Casper Lassenius. Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Ciudad Real, Spain, 45:1–45:10, September 2016.

V Eero Laukkanen, Maria Paasivaara, Juha Itkonen, Casper Lassenius. Comparison of Release Engineering Practices in a Large Mature Company and a Startup. *Submitted to Empirical Software Engineering*, May 2017.

Author's contribution

Publication I: "Problems, causes and solutions when adopting continuous delivery——A systematic literature review"

The author was the first author and the main responsible for the study design. The author collected the data, analyzed the data and wrote the article. Other authors participated in the study design, inter-rater agreement and commented other parts of the work.

Publication II: "Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study"

The author was the first author and the main responsible for the study design. The author collected the data with the second author, analyzed the data and wrote the article. Other authors participated in the study design and commented other parts of the work.

Publication III: "Towards Continuous Delivery by Reducing the Feature Freeze Period: A Case Study"

The author was the first author and the main responsible for the study design. The author collected the data with the second author, analyzed the data and wrote the article. Other authors participated in the study design and commented other parts of the work.

Publication IV: "Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization"

The author was the first author and the main responsible for the study design. The author collected the data with the second and third author, analyzed the data and wrote the article. Other authors participated in the study design and commented other parts of the work.

Publication V: "Comparison of Release Engineering Practices in a Large Mature Company and a Startup"

The author was the first author and the main responsible for the study design. The author collected half of the data with the second and fourth author. Other half of the data was collected by the fourth author. The author analyzed the data and wrote the article. Other authors participated in the study design and commented other parts of the work.

1. Introduction

1.1 Background

Release engineering is an important part of the software development life cycle. Release engineering means the process of bringing the individual changes made to a software system to the end users of the software with high quality [1]. The process consists of version control, building, testing, deploying and releasing the software [1]. In order to survive the market pressure, companies are required to respond quickly to customer needs [2], which is enabled by modern release engineering practices [1] (see the definition in 2.1). The modern release engineering practices emphasize using build, test and deployment automation [3] and facilitating collaboration across functional boundaries [4], so that it is possible to achieve both speed and quality in the release engineering process.

1.2 Research problem and questions

While some companies, such as Facebook [5], have been successful in adopting modern release engineering practices, other companies have found the adoption to be problematic [6]. Some explanations for the problems are related to the domain of the developed software system, e.g., in the embedded systems domain [7] and mobile software domain [8]. However, other aspects have been identified as well, such as organizational and architectural aspects [9] that explain the problems. In this dissertation, we aim to understand the research problem: **what prevents organizations from adopting modern release engineering practices?**

We address the research problem by answering four research questions:

1. How are modern release engineering practices adopted?

2. What problems have been faced when adopting modern release engineering practices?
3. What explanations for the problems have been presented?
4. What solutions for the problems have been used or proposed?

The research questions follow the logic of root cause analysis [10]; in order to solve problems, we have to first understand the phenomenon (RQ1), identify the problems (RQ2), explain the problems (RQ3) and then develop solutions for the problems (RQ4).

1.3 Research methods

First, we conducted a systematic literature review (SLR) in Publication I, to survey the literature about the adoption problems, their causes and solutions. As we found that the literature was not providing enough explanations for the problems, we conducted four additional case studies. In the first case study in Publication II, we investigated the adoption problems in a distributed organization. This case study was extended in a follow-up case study in Publication III, in order to see how the release stabilization period could be reduced after the adoption efforts. In another case study, we focused on the consequences of the stage-gate development process and how it explained the adoption problems in Publication IV. Finally, we compared two organizations with different organizational contexts working with similar products in a fourth case study in Publication V, in order to compare the effects of different organizational contexts.

1.4 Structure of the thesis

The dissertation is structured as follows. First, we discuss related work in Section 2. Second, we define our research problem and introduce the research methodology in Section 3. Third, we present an overview of the results in the publications in Section 4. Fourth, we discuss the implications and validity of the results in Section 5. Finally, we conclude the dissertation and propose future work in Section 6.

2. Related work

In this section, we review related work related to the dissertation. First, we conceptualize the modern release engineering and after that, review empirical research on the subject.

2.1 Modern release engineering

Release engineering is the process where the individual changes made to a software system are assured for quality and brought to the end users of the software [1]. As shown in Table 2.1, the areas of release engineering have been studied under several research topics: software configuration management (SCM) [11], software testing (ST) [12], software release management (SRM) [13] and IT operations (ITO) [14]. What release engineering adds to the research topics is that it studies the connections between the topics and aims to improve the release engineering process holistically to avoid local optimization.

While assuring the quality during software development has been a research topic in the past, recently more focus has been set on the speed

Table 2.1. The relationships of the release engineering areas and previous research topics.

Release engineering areas	**SCM**	**ST**	**SRM**	**ITO**
Version control	X			
Deployment pipeline	X	X		
Build systems	X			
Infrastructure-as-code				X
Deployment			X	X
Release			X	X

of delivery due to market pressure [2]. Thus, when designing a release engineering process, there is a requirement to balance release confidence and velocity [15]. In this dissertation, we define that *modern release engineering practices* [1] aim to achieve both confidence and velocity, including practices such as version control [1], organizational collaboration [14], build, test and deployment automation [3], releasing individual changes [3], decoupling deployment from release [1, 16] and detecting defects during deployments [17].

The industry has used three concepts to describe the maturity level of modern release engineering practices: continuous integration, continuous delivery and continuous deployment. Next, each of them is introduced.

Continuous integration (CI) [18] states that changes to a software system are automatically build and tested. Typically a CI server, such as Jenkins [19], is used to automatically build and test changes that are submitted to the used version control system. The CI server also notifies the developers if the build or tests fail, so that problems can be solved immediately when occurred. Thus, the CI server keeps the development team aware of the status of their work [20]. In order to keep the CI feedback fast, developers should submit their changes frequently to the version control system. For example, Fowler [18] suggests changes to be submitted at least once a day.

Continuous delivery (CD) [3] extends CI by requiring that the test and deployment processes are automated to the extent that the changes can be deployed to the production environment after passing the tests. A CI server is used to automate the process, but in addition tools, such as Docker [21] and Ansible [22], are used to automate and manage the deployments to different environments. Finally, a testing tool such as Robot Framework [23] is used to automate the end-to-end acceptance tests. In CD, production deployments are not automatically triggered after passing all the tests, but instead production deployments are triggered manually, in order to allow deferring production deployments for business reasons.

Continuous deployment (CDep) [17] removes the decision whether to deploy changes to production or not, as the changes are automatically deployed if the tests pass. Thus, the only difference to CD is that the production deployments are automatically triggered after the changes have passed all the required tests. A production deployment does not always mean that a change is released, as it can be hidden with feature toggles [16].

2.2 Empirical research on modern release engineering

As advances in release engineering have been driven by the industry [1], empirical research available on the modern release engineering has been limited until recently [24]. In his dissertation, Wright [9] studied release engineering in general and concluded that improving process automation and organizational communication would prevent the most common release engineering failures. Leppänen et al. [25] surveyed Finnish software companies and discovered that the goals to adopt modern release engineering practices vary between companies. The same survey data was analyzed by Mäkinen et al. [26] to investigate tool-usage in release engineering. Some tools, such as version control, are used in every company, whereas test automation tools for acceptance testing are not used in most of the companies. Thus, it seems that some areas of the release engineering are commonly used, but not all of them.

Other related concepts to modern release engineering are rapid releases [27] and devops [4]. Rapid releases concern the release frequency, and modern release engineering practices are an enabler for rapid releases, as the time for testing with rapid releases is limited [27]. Devops is an organizational approach and focuses on the collaboration of different organizational functions, especially development and operations [4]. While organizational collaboration is not an area of release engineering, it is an enabler for better release engineering practices [9].

2.2.1 Benefits of modern release engineering practices

Benefits of modern release engineering practices have been examined in multiple studies. Claimed benefits, such as support for automated testing, improved communication, developer productivity and project predictability have been verified in an interview study [28]. An analysis of version control data has shown that projects using modern release engineering practices are more productive without a decrease in software quality [29]. Furthermore, projects using modern release engineering practices are able to release the software more often [30].

2.2.2 Adoption of modern release engineering practices

The adoption of modern release engineering practices have been characterized in two models. The stairway to heaven model by Holmström Ols-

son et al. [31] describes CI and CD as the stepping stones between the adoption of agile practices and having R&D function as an experiment system. The model states that after adopting CD, an organization can develop software through experimentation instead of traditional specification; minimal functionality is implemented and deployed first to see whether or not the expected benefits of the functionality are present when the software is actually used.

The other adoption model by Eck et al. [32] describes the implications that an organization must implement when adopting the modern release engineering practices. The model introduces three adoption stages, acceptance, routinization and infusion, in which different implications take place. The implications are categorized into four categories: purposeful adoption, processes & organization, testing and IT infrastructure. Thus, the adoption includes both social and technical implications.

2.2.3 Adoption problems of modern release engineering practices

The adoption of modern release engineering practices has been investigated in many experience reports and a few case studies, as studied in the systematic literature review of this dissertation, Publication I. While many problems during the adoption have been identified and solutions to mitigate the problems have been proposed, there is still not a clear picture of the adoption, especially in more complex software development contexts. In addition, there are no commonly accepted explanations for the problems during the adoption of modern release engineering practices.

In order to understand the adoption problems, the adoption itself has to be understood as a technological and social process [33]. In addition, the process is centered around an organization developing software systems. Eck et al. [32] studied the actions needed to adopt the practices across an organization, and found that the actions relate to the themes of purposeful adoption, processes and organization, testing and IT infrastructure. In addition to an organizational-level view of adoption, the adoption requires acceptance on an individual-level too [34].

3. Research problem and methodology

In this section, we first define the research problem and research questions of the dissertation. After that, we introduce the research methodology, including data collection, data analysis and validation methods.

3.1 Research problem and questions

As previous research has not identified the explanations for modern release engineering adoption problems, in this dissertation, we perform *exploratory* research on the adoption problems. Exploratory research is a valid research approach when there is not much information available on the subject [35]. The aim of exploratory research is to produce hypotheses that can be used in the following descriptive or explanatory research [35]. Our research can be also categorized as inductive [35], as we did not employ any theory to explain the phenomenon before conducting the studies.

The research problem of the study is **what prevents the adoption of modern release engineering practices?** In our investigation, we limit ourselves to the internals of organizations and do not attempt to understand other stakeholders', such as customers', viewpoints. The research problem is investigated by answering the research questions of the study in Table 3.1. The research questions follow the logic of root cause analysis [10]. First, in order to understand the problems, we have to understand the phenomenon itself (RQ1). Second, we have to identify problems (RQ2), in order to improve the practice. Third, we have to explain the problems (RQ3), in order to be able to solve them. Finally, we can develop solutions for the problems based on the explanations (RQ4).

We map the publications to the research questions in Table 3.1. In Publication I, we conducted a systematic literature review to investigate the

Table 3.1. The relationships of the research questions and the publications. "X" denotes that the research question was directly addressed in the publication and "(X)" denotes that the publication indirectly yielded results related to the research question.

Research Question	I	II	III	IV	V
RQ1. How are modern release engineering practices adopted?		(X)	(X)		
RQ1.1. How is the continuous end-to-end testing practice adopted?		X			
RQ1.2. How is the release stabilization period reduced?			X		
RQ2. What problems have been faced when adopting modern release engineering practices?	X	X		X	
RQ3. What explanations for the problems have been presented?	X	(X)		X	X
RQ4. What solutions for the problems have been used or proposed?	X	X			

research questions RQ2, RQ3 and RQ4. In Publication II, we conducted the first case study that examined the research questions RQ1, RQ2 and RQ4. The last three case studies were more focused. In Publication III, we focused on the reduction of release stabilization period and how it was enabled by the test automation efforts. In Publication IV, we focused on how the used stage-gate product development process explained the adoption problems. Finally, in Publication V, we focused on how the organizational context explained the used release engineering practices in two case organizations.

3.2 Research methodology

In this section, we overview the used research methodology. First, we introduce the systematic literature review methodology used in Publication I. After that, we introduce the case study methodology used in the other publications. Finally, we discuss the data collection, analysis and validation methods used in the publications.

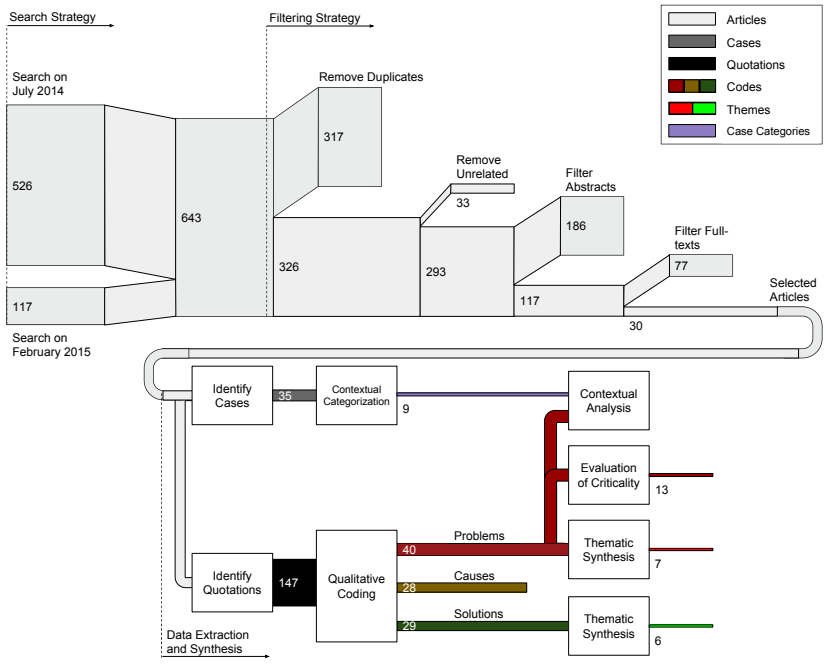

Figure 3.1. Overview of the research process in Publication I.

3.2.1 Systematic literature review

Systematic literature reviews (SLR) are used to present the available evidence on specific research questions with trustworthy, rigorous and auditable methodology [36]. Conducting a SLR consists of five steps [36]: identification of research, selection of primary articles, study quality assessment, data extraction and data synthesis.

In Publication I, we identified related literature by using the following search string in major bibliographic databases: *("continuous integration" OR "continuous delivery" OR "continuous deployment") AND software*. As the review was conducted in 2014, the concept of "modern release engineering" was not yet identified, and we used the terms continuous integration, continuous delivery and continuous deployment, which were used in both research and practice for the phenomenon. Currently, the research field has matured, and other relevant search terms, such as "devops", have emerged. The search, filtering and data extraction process is visualized in Figure 3.1. A total of 30 articles were selected for the data extraction phase. We classified 21 articles as experience reports and 9 articles as scientific articles.

We selected the primary articles by excluding articles that did not pro-

vide evidence from real-life software development contexts. In addition, we excluded cases that provided only technical implementation descriptions, but did not provide evidence from the use of the technical implementations.

We did not exclude articles based on quality assessment, as most of the articles were experience reports and at the time there were only a few proper empirical research articles on the subject. We mitigated the bias from the experience reports by focusing on experiences instead of opinions or claims.

The data was extracted and synthesized by following the thematic synthesis procedures [37]. First, quotations of text that discussed problems with the modern release engineering practices or the adoption were identified in the primary articles. Next, the problems were given descriptive codes and descriptions. In addition to problems, explanations of the problems and solutions to the problems were captured with codes. The explanations were coded as relationships between the problem codes, e.g., nondeterministic tests can explain the ambiguity of test results. Finally, the problem and solution codes were grouped under higher-order themes which describe areas where adoption problems can exist.

3.2.2 Case studies

Case study is a research strategy which can be used for investigating a contemporary phenomenon within its real-life context, especially when it is difficult to separate the phenomenon from the context [38]. The case study strategy is appropriate for investigating modern release engineering practices, as it would be difficult to simulate the phenomenon in a laboratory setting. Furthermore, our goal in this dissertation is to capture realistic adoption problems which might not occur in isolation from the context of development. It is also possible that the adoption problems are explained by the context.

In this dissertation, we conducted four case studies studying four cases which are described in Table 3.2. We used development organizations developing single products as the units of analysis. Two case studies were conducted for the case Ericsson in Publication II and Publication III. The case Nokia was studied in Publication IV and both cases BigCorp and SmallOrg were compared in Publication V.

The cases were selected to be investigated because they were revelatory [38] considering the adoption of modern release engineering practices.

Table 3.2. Case organizations studied in this dissertation. OC=Organizational Context, CS=Company Size, COS=Case Organization Size, OD=Organizational Distribution, CD=Case Domain.

Case	Ericsson	Nokia	BigCorp	SmallOrg
OC	Unit in a large company	Unit in a large company	Unit in a large company	Startup
CS	$>$ 100,000	$>$ 50,000	$>$ 20,000	50
COS	135	–	180	50
OD	Four sites in Europe	Several sites in multiple countries	Several sites in Europe and Asia	One site in North America
CD	Telecom	Telecom	Domain X	Domain X

Table 3.3. Data collected in the case study publications.

Publ.	II	III	IV	V
Cases	Ericsson	Ericsson	Nokia	BigCorp, SmallOrg
Data	27 interviews	11 interviews, version control and issue tracking data	2 workshops with altogether 15 participants	18 interviews, continuous integration data

Case Ericsson was revelatory, because it was a large distributed organization adopting modern release engineering practices. Case Nokia was revelatory, because the product development process used in the case was claimed to explain the adoption problems. Finally, comparing BigCorp and SmallOrg was revelatory, because they were developing competing products in different kind of organizations. Therefore we were able to investigate the effects of organizational context to the release engineering practices and mitigate the effect of the product context on the practices.

3.2.3 Data collection

In the case studies, the main data collection method was qualitative interviews with an interview guide approach [39], as shown in Table 3.3.

In Publication III, quantitative data from version control and issue tracking systems was collected to triangulate the qualitative results. Also in Publication V, continuous integration data was collected to illustrate the difference in discipline in the cases. Finally, in Publication IV, we used ARCA root cause analysis method [10] to investigate the adoption problems. The ARCA method is comparable to structured group interviews [39].

In the interview guide approach [39], interview questions are not determined beforehand, but instead a list of topics is used as a guide to make sure that the research questions are covered in the interviews. Otherwise, the approach is flexible and allows interviewees discuss subjects that they feel important regarding the subject of the research. In addition, questions can be asked spontaneously based on the role and experience of the interviewee, which allows investigating topics that would be difficult to plan before the interviews.

We interviewed various roles in the organizations. In Publication II, we interviewed managers, architects, developers, testers and coaches. In Publication III, we interviewed managers, testers and developers. In Publication IV, we interviewed managers, developers and testers. In Publication V, we interviewed managers, architects, developers and service team members. This allowed us to gain a holistic understanding on the subject, not being biased by single roles. In addition, we attempted to interview team members from as many teams as possible, although the organizational distribution limited the possibility to interview everyone. The interviewees were selected by the key stakeholders in the case organizations, according to the requests we made.

3.2.4 Data analysis

The interviews were audio recorded and transcribed by a professional transcription company. The ARCA workshops were transcribed by the author of the dissertation, due to having multiple people talking in the workshop and technical language. The transcriptions were first read through by the author to get immersed with the data [39]. After that, the author identified quotations from the transcriptions that were relevant to the research goals of the individual studies. The concepts in the quotations were given descriptive codes [39] to identify the same concepts in other quotations too. The codes were refined until they represented the interview data well enough and comprised a coherent whole.

Quantitative data from software repositories were analyzed in Publication III and Publication V. In Publication V, the descriptive statistics were calculated from the data, to compare the continuous integration discipline in the two case organizations. In Publication III, time series analysis [38] was done to investigate the effects of reducing the release stabilization period. In both publications, the quantitative data analysis was performed to triangulate [38] the qualitative data analysis.

3.2.5 Validation

In this section, we describe how we mitigated the threats validity, according to the validity types of case study research [40]: construct validity, internal validity, external validity and reliability. In Section 5.5, we discuss what threats we could not mitigate.

Construct validity

Construct validity assesses whether the constructs operationalized in research are the same as what is investigated according to the research questions. In the SLR, it was more difficult to assess if an article used the same definition for, e.g. CI, as we did. Nevertheless, the inclusion and exclusion steps of the SLR were performed by multiple researchers in order to reduce bias.

In the case studies, it was easier to improve construct validity, because we either asked the interviewees what they meant by CI (Publication II) or provided them a common definition (Publication IV). Furthermore, in the interviews, we used a flexible interview guide approach and open questions, which allowed interviewees to explain their experiences with their own words. Finally, we could ask them clarifications if something was ambiguous.

Internal validity

Internal validity assesses whether the explanations based on the collected data are valid. In both SLR and case studies, this was achieved with triangulation [38]. First, multiple researchers were involved in primary article selection in the SLR and in most of the interviews. Second, in the case studies, we interviewed employees in various roles, teams and sites. Third, the data analysis results were reviewed and criticized by the other authors of the publications. Finally, key stakeholders from the case organizations reviewed the final results of the case studies and made sure

that the results truthfully represented their situation.

External validity

External validity assesses whether the results are generalizable outside the studied cases. We described the context of the case organizations in order to allow generalizing the results to other similar contexts. In Publication V, we identified constructs that can be generalized outside the studied cases.

Reliability

Reliability assesses whether the data and analysis are dependent on the researchers. We documented the used data collection and analysis procedures in the publications. In the SLR, we performed inter-rater agreement on the selection of primary articles and achieved moderate agreement.

4. Overview of the results

In this section, an overview of the answers to the research questions are given based on the publications in the dissertation.

4.1 RQ1. How are modern release engineering practices adopted?

The adoption of modern release engineering practices consists of automating the build, test and deployment activities. In this dissertation, we focus on the test automation. In addition, we study how the release stabilization period can be reduced with test automation, as in the more mature modern release engineering practices, there should be no release stabilization period needed before production deployment.

4.1.1 RQ1.1. How is the continuous end-to-end testing practice adopted?

In Publication II, we investigated a case organization that had adopted the practice of continuous end-to-end testing (CET). For the case organization, it took over a year to automate half of the manual test cases of their system, as shown in Table 4.1. The adoption consisted of three phases. First, the CET system was created by two specialized teams. Second, automated end-to-end tests (AET) were created for the legacy functionality by the two teams. Third, the habit of creating AETs for new functionality was spread across the whole organization. The first two phases could be done by specialized teams, whereas the last phase required collaboration of the whole organization. While creating AETs is a technical task, spreading the practice in an organization is a social process.

Table 4.1. Timeline of the events during continuous end-to-end testing adoption in Publication II.

Date	Event
September 2013	Testing activities and test automation close to non-exist, two teams on Site A start to build the CET system and create legacy AETs
January 2014	Evaluation of test frameworks done by one team on Site A, one framework chosen for the system
March 2014	Events organized at all sites to spread the test automation mindset and give trainings for the chosen test framework
December 2014	50 % of legacy AETs created

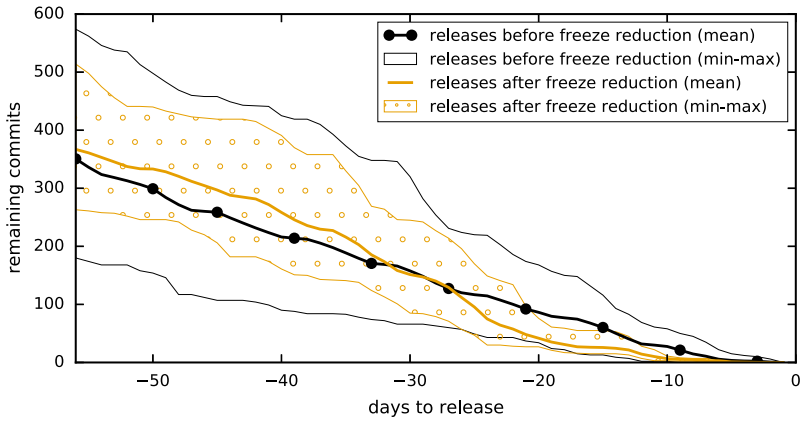

Figure 4.1. After the release stabilization period was reduced, less commits were done near the release date in Publication III.

Table 4.2. Problem themes and related problems found in Publication I.

Theme	Problems
Build Design	Complex build, inflexible build
System Design	System modularization, unsuitable architecture, internal dependencies, database schema changes
Integration	Large commits, merge conflicts, broken build, work blockage, long-running branches, broken development flow, slow integration approval
Testing	Ambiguous test result, flaky tests, time-consuming testing, hardware testing, multi-platform testing, UI testing, untestable code, problematic deployment, complex testing
Release	Customer data preservation, documentation, feature discovery, marketing, more deployed bugs, third party integration, users do not like updates, deployment downtime
Human and Organizational	Lack of discipline, lack of motivation, lack of experience, more pressure, changing roles, team coordination, organizational structure
Resource	Effort, insufficient hardware resources, network latencies

4.1.2 RQ1.2. How is the release stabilization period reduced?

In Publication III, we investigated how the same case organization as in Publication II reduced the release stabilization period after automating the end-to-end and subsystem tests. Reducing the release stabilization period made the feature freeze practice work more as intended; less commits were done near the release date, as shown in Figure 4.1. However, further reduction of stabilization period would have required deployment automation too, as setting up the test environments for the release testing took the majority of the time during the release stabilization period.

4.2 RQ2. What problems have been faced when adopting modern release engineering practices?

In Publication I, we studied the literature about the problems faced when adopting modern release engineering practices. From the 30 analyzed articles, we extracted 35 descriptions of cases where modern release en-

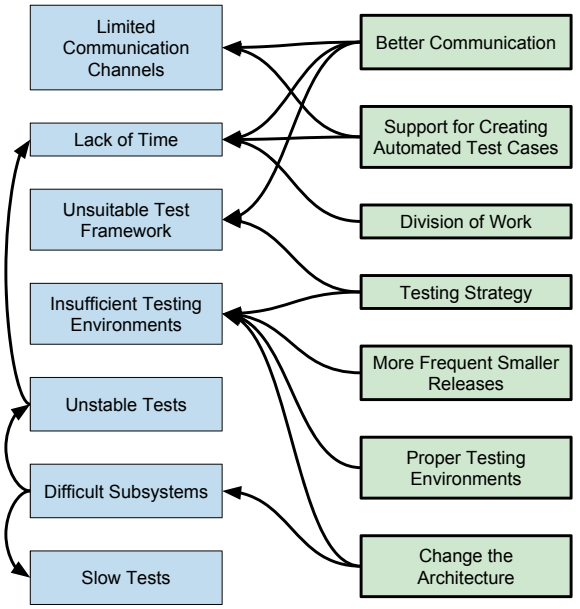

Figure 4.2. Adoption problems and their proposed solutions found in Publication II.

gineering practices were either used or being adopted. We could identify multiple problems related to the adoption, under the themes of build design, system design, integration, testing, release, human and organizational and resource (see Table 4.2).

In Publication II (see Figure 4.2) and Publication IV, we investigated the problems in two case organizations. We could recognize all the problem themes in the cases. However, the problems that the cases were mostly struggling with were in the themes of system design, integration, testing, human and organizational and resource. In addition, problems in human and organizational and resource themes were perceived to explain the problems in the other themes.

4.3 RQ3. What explanations for the problems have been presented?

In Publication I, we found that in the literature, some problems were perceived to explain other adoption problems, as shown in Figure 4.3. However, the literature does not explain, why the adoption problems are not solved in practice, even when there are plenty of solutions described in the literature. The case studies, Publication II, Publication IV and Publication V, provided explanations how the problems emerge and why they

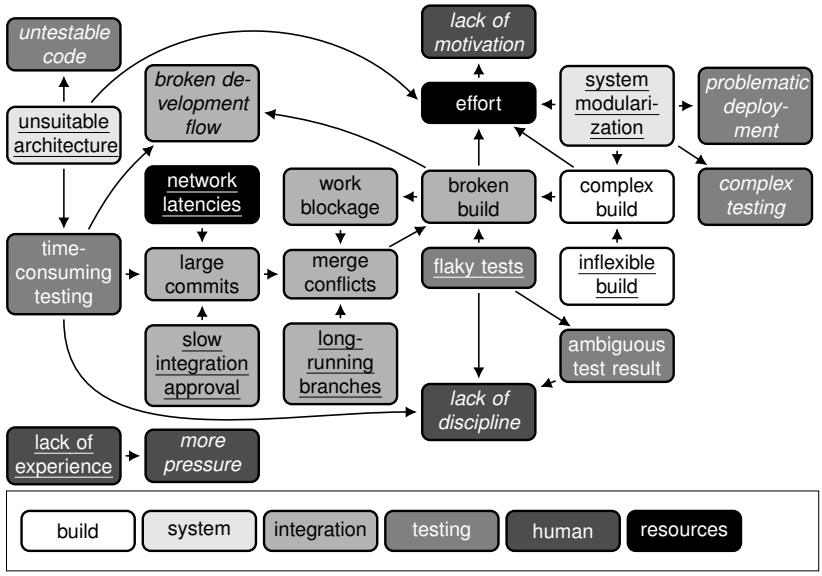

Figure 4.3. Relationships between the problems found in Publication I.

are not solved.

In Publication II, *the system design* made the automated testing and deployment more difficult. First, changes to some subsystems could not be version controlled at all, which would be needed for the automatic deployment of the changes. Second, some subsystems did not behave deterministically enough to be automatically tested. Instead, the system behavior included delays which would slow down the execution of tests and cause tests to fail nondeterministically. The subsystems were developed by third parties and could not be improved by the case organization. In Publication IV, the system design also explained the adoption problems by causing nondeterministic tests.

In Publication II, *the organizational distribution* was an explanation for the slow adoption of continuous end-to-end testing. The distribution made communication and coordination more difficult. In addition, all the adoption drivers were located on a single site and the adoption had not spread over the site borders. In Publication IV, the organizational distribution also explained the adoption problems by lack of communication and coordination, which resulted in broken builds and duplicate testing.

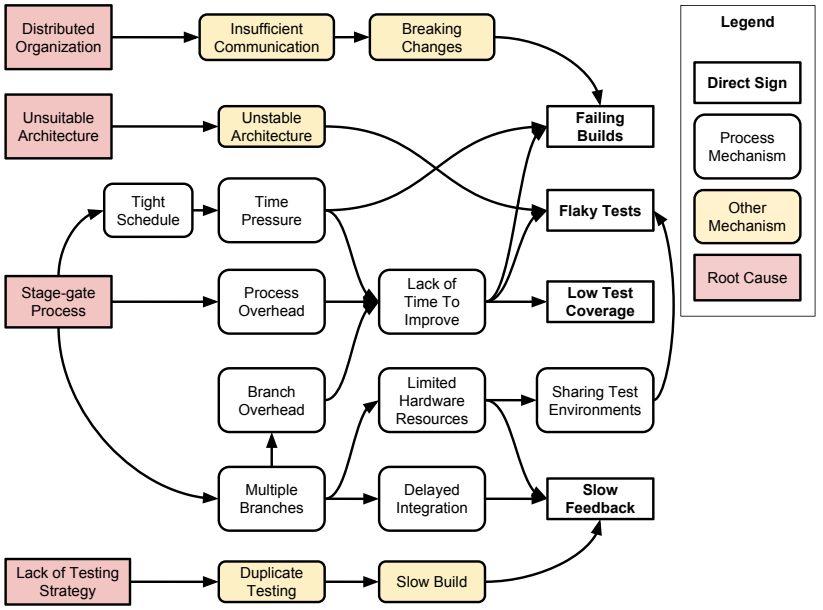

Figure 4.4. Mechanisms that explain the adoption problems in Publication IV.

4.3.1 Stage-gate product development process explaining the problems

In Publication IV, we investigated how the stage-gate product development process explained the adoption problems in the case organization. We found that the plans which were conceived early in the stage-gate process were overscoped and could not be realized on later stages, adding time pressure on new feature development and reducing time from the adoption (see Figure 4.4). Similar time pressure for new feature development was explaining the adoption problems in Publication II. In addition, the stage-gate process required the use of different development branches for different stages, which added complexity to the development work and increased the cost of adoption.

4.3.2 Organizational context explaining the problems

In Publication V, we compared the release engineering practices implemented in two organizational contexts: large mature company context and startup context. Since the case organizations were developing competing products, we could control the effect of product context and investigate the differences explained by the organizational context (see Figure 4.5). We found out that in the large mature company context, the

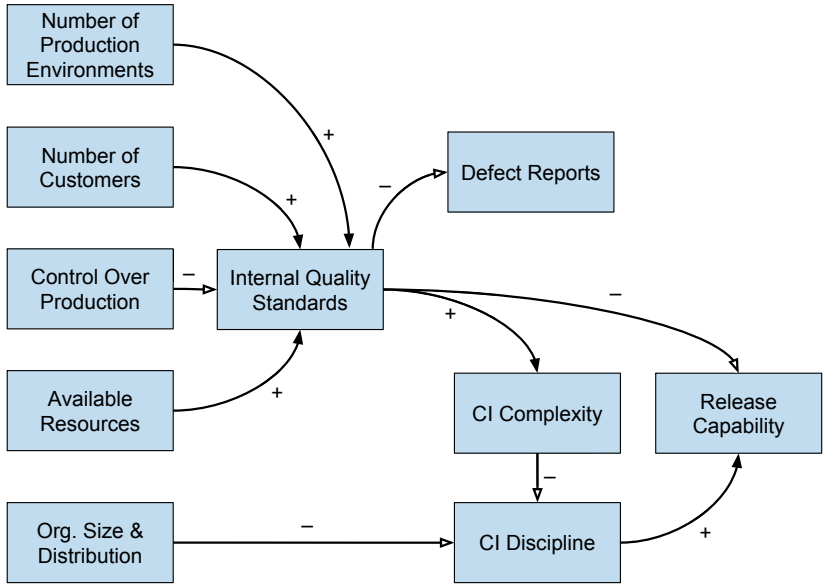

Figure 4.5. Mechanisms that explain the outcomes of the release engineering practices in Publication V.

number of customers and stakeholders inside the company increases the cost of adoption due to higher quality requirements and adoption complexity. Thus, the increased cost of adoption can prevent it from happening. In the startup context, the main explaining factor is the lack of resources which prevents further adoption. However, with flexibility, startups can mitigate the lack of resources by collaborating more tightly with customers.

4.4 RQ4. What solutions for the problems have been used or proposed?

In Publication I, we found that in the literature, most of the adoption problems can be solved with the solutions given in Table 4.3. However, since the problems described in the literature are not contextualized, it might be that the solutions are not applicable in every context. In Publication II, we identified proposed solutions for the problems present in the case organization. Especially the actions of starting the adoption with specialized teams for the technical implementation, demonstrating early value with radiators, providing trainings and support for creating the automated test cases while learning and moving the specialists to other teams after implementing the initial technical infrastructure were considered beneficial

Table 4.3. Solutions themes and related solutions found in Publication I.

Theme	Solutions
System Design	System modularization, hidden changes, rollback, redundancy
Integration	Reject bad commits, no branches, monitor build length
Testing	Test segmentation, test adaptation, simulator, test parallelization, database testing, testing tests, comprehensive testing, commit-by-commit tests
Release	Marketing blog, separate release processes
Human and Organizational	Remove blockages, situational help, demonstration, collaboration, social rules, more planning, low learning curve, training, top-management strategy, communication
Resource	Tooling, provide hardware resources

for succeeding with the adoption. Still, the actions could not be applied on every site of the organization, because there was no required organizational support and individual acceptance, which explains the slower than expected adoption rate in the case organization.

5. Discussion

In this section, we discuss the implications of the results as shown in Figure 5.1, first generally as hypotheses that can be tested in future studies, and then specifically to research and practice. After that, we discuss the threats to validity of the results.

5.1 Adopting modern release engineering practices

In this section, we will discuss the implications regarding the adoption of modern release engineering practices. We will focus on what the adoption requires and how it can be measured.

5.1.1 Adopting modern release engineering practices requires build automation, test automation and deployment automation

Adopting modern release engineering practices requires automation of build, test and deployment activities, which is described in the practitioner literature [3]. This is also visible in the results from the SLR in Publication I, as there were problem themes for build design and testing, and a problem concept "problematic deployment" under the testing theme. Finally, lack of automation and tool support has been found to hinder successful release processes [9].

In the case studies, we focused on the test automation, because the improvements in the case organizations focused mostly on test automation. Especially automating the end-to-end testing is problematic, since it was the major theme discussed in Publication II, and a survey of Finnish software companies shows that acceptance testing tools are not used in most of the companies [26]. Thus, especially the automation of the end-to-end or acceptance tests is a substantial part of adopting modern release engi-

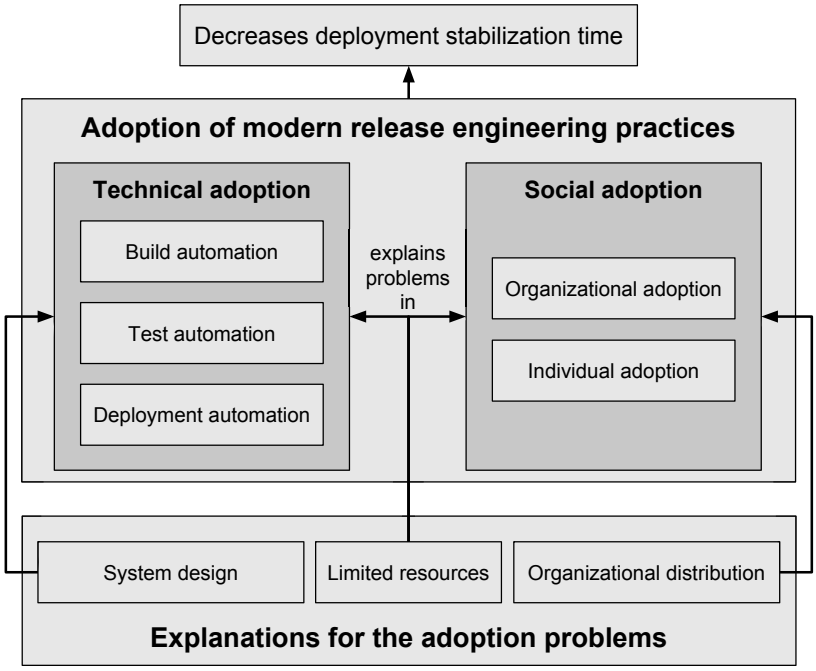

Figure 5.1. An overview of the implications of the results.

neering practices.

The interviewees did not bring out build automation as a noteworthy theme in the interviews, and we suspect that the build automation was not a substantial issue in the case organizations. However, for some other software systems with more complex compilation configurations, build automation can be a major issue [41].

Regarding deployment automation, the case organizations did not have access to production environments in order to automate deployments to production. However, deployments to test environments were automated in the case organizations. In Publication III, we identified that deployments to the release test environment were problematic and they could have been automated to improve the release testing process. But as releases were performed six times per year in Publication III, the lack of automation regarding deployments was not a critical issue. When moving towards faster release cycles, we suspect that deployment automation becomes more important.

5.1.2 Adopting modern release engineering practices requires adoption on both organizational and individual levels

Adopting complex technological innovations in organizations requires both organizational- and individual-level adoption [34]. In the context of modern release engineering practices, organizational-level adoption is required to get resources for the adoption and to coordinate the adoption. Individual-level adoption is required, because the adoption requires that individuals automate the testing of the developed system and keep the system stable during the development. In Publication IV, we could see that organizational support for the adoption was missing, as the adoption was driven by the development organization and not by the management. Thus, the organizational processes did not provide enough slack [42] and resources for the adoption.

In Publication II, there was organizational support for the adoption, as two teams were permitted to focus only on the adoption and the managers in the organization were facilitating the adoption. However, the individual-level adoption was not proceeding as fast as expected by the adoption drivers. Gallivan [34] shows that bureaucratic culture, centralized planning, cultural norms of learning and job roles and individual attributes may hinder the adoption on the individual-level. The case organization in Publication II was part of a large mature company, and it can be suspected that some of the hindering factors were present in the organization, although they were not under our investigation.

Persuasion of informal advocates, e.g. team members, can significantly speed the adoption of innovations [43]. This was also visible in Publication II, as moving early adopters to other teams had spread the adoption to other team members in those teams. However, after accepting the adoption, individuals still have to learn to use the test automation tools, which can take long time if the individual has no previous experience with test automation. Furthermore, time for learning can be limited, as the development of the system is often prioritized higher than the adoption, which was visible in both Publication II and Publication IV.

5.1.3 Adopting modern release engineering practices reduces the time needed for deployment stabilization

In Publication III, we studied how the adoption of modern release engineering practices allowed the case organization to reduce the time needed

for release stabilization. According to the results, reducing the release stabilization time did not have any negative consequences. Thus, we could deduce that the time needed for release stabilization could be used as a measure for the maturity of modern release engineering adoption.

However, when developing new features or making changes with large impacts, there is often a need to hide the changes with feature toggles [16] after they have been deployed to production environment, in order to verify the functionality in a realistic setting. Thus, there might not be *deployment* stabilization before the production deployment in CD, but there can still be *release* stabilization after the production deployment.

As in modern release engineering, the release of a feature is decoupled from the deployment of a feature [1], stabilization for deployment and stabilization for release can be decoupled too. In Publication III, the deployment and release of the system were still coupled and thus we could only speak of release stabilization regarding the case organization. However, based on the literature, we suggest using deployment stabilization as the measure for the maturity of modern release engineering practices instead of release stabilization, in order to cover scenarios where deployment and release are decoupled.

The release stabilization period was reduced from six weeks to three weeks in Publication III. From the literature, we can find release stabilization periods for other types of software: 9 weeks for Linux [44], 13 weeks for Chrome [44] and 1 week for Facebook mobile applications [8]. As explained in [8], product context can explain the need for release stabilization period, e.g., if the deployment cannot be controlled by the development organization and the number of and variance in production environments is large. Thus, the measures of deployment stabilization are not easily comparable across different software systems.

5.2 Explanations for the adoption problems

In this section, we discuss the implications regarding the explanations for the adoption problems. First, we overview the problems and then focus on specific explanations based on organizational distribution, system design, stage-gate process and organizational context. Finally, we discuss the proposed or used solutions.

5.2.1 Problems are related to build automation, test automation, deployment automation or social adoption

As discussed in the previous section, the adoption requires build automation, test automation, deployment automation and adoption on both organizational and individual levels. Thus, these areas cover the *primary* adoption problems. In the SLR in Publication I, we also identified the problem themes of system design and integration. However, system design explains the primary problems of build automation, test automation and deployment automation. In addition, integration problems can be explained by the primary problems.

5.2.2 Organizational distribution explains social adoption problems

Organizational distribution explains the social adoption problems related to organizational and individual adoption. In Publication II, the site boundaries and limited communication channels were limiting the adoption. Similarly, in Publication IV, the organizational distribution hindered the communication, which showed as surprising changes to the software system coming from other sites. However, in the literature in Publication I, the organizational distribution was not identified as a problem. Thus, the relationship between the organizational distribution and adoption problems should be investigated in further studies.

While the organizational distribution explains the adoption problems, adopting modern release engineering practices can also improve collaboration in a distributed setting [32]. Thus, if possible, the adoption should be made before distributing the development and not afterwards, as in a distributed setting the adoption itself is more difficult.

5.2.3 System design explains technical adoption problems

System design can explain problems in the technical areas of the adoption: build automation, test automation and deployment automation. In Publication I, we found cases where the system design had an effect on the testability and deployability on the system. Similarly, in Publication II and Publication IV, the system design made the automated testing and deployment of the system more difficult, as the system tests were not reliable enough and deploying the systems was not possible without downtime. The results of this dissertation align with other studies that have

investigated the relationship between the system design and modern release engineering practices [9, 45, 46, 47, 48].

5.2.4 Limited resources explains both social and technical adoption problems

In Publication IV, we found that the use of stage-gate process limited the resources for the adoption, and thus hindered the adoption in all areas. Similar finding was made in Publication II, where the pressure for new feature development decreased the amount of resources for the adoption. The problems can be due to overscoping [49], as the scope for the new feature development in both cases was stated to be high and prioritized over the adoption. Gruver et al. [50] describe a similar situation, where the scope of development was not decreased, but instead an architectural change was applied which allowed the development of same scope with less resources than previously. Thus, changes in architecture or business processes might be needed in order to allow the adoption enough resources for proceeding as planned.

In Publication V, we found that different types of organizations have different kind of requirements for the modern release engineering practices. The variance in requirements explains the cost to adopt the practices, especially regarding the test automation. Thus, in some contexts, the resources can be limited for the adoption because the requirements are set higher, whereas some contexts can have a successful adoption with less resources if the requirements are lower. We are not aware of other studies that have investigated the effects of organizational context to the modern release engineering practices.

5.2.5 There are no easy solutions for the adoption problems

In Publication I, we found solutions from the literature that can help in adopting the modern release engineering practices. Some of the reported solutions relate to the system design explanation and can help in some situations, but as seen in Publication II, solving system design problems can require replacing entire subsystems, which requires substantial resources. Wright [9] emphasizes the role of architecture when improving release engineering processes. He also makes the observation that organizations recognize the need for improvement in architecture, but implementing the improvements is often difficult.

Recently, an architectural pattern called microservices [51] has been used to solve the system design adoption problems when the system is executed in a cloud environment. The pattern implies that the developed system is split into multiple services that can be built, tested and deployed independently of each other. As speculation, the case organization studied in Publication II and Publication III could have gained benefit from the microservices approach, as the organization had control over the production environment, although only through a separate operations organization. The case organizations studied in the other publications would have gained less benefit from the approach, because the production environments were operated by their customers.

We did not find any strategies from the literature on how to solve the problems related to the organizational distribution. If possible, similar actions that were made in Publication II could be applied on all sites of the organization to spread the adoption over the site boundaries. Furthermore, collaboration between sites can be improved with the best practices in global software engineering [52]: face-to-face meetings and effective and frequent synchronous communication.

5.3 Implications to research

This dissertation provides a framework for future research on modern release engineering practices. Future research can consider only some parts of the modern release engineering practices, for example, build automation, test automation or deployment automation. In addition, future research can use the metric of deployment stabilization time to assess the maturity and improvement of modern release engineering practices. Also the difference between deployment and release stabilization can be studied.

Social adoption can be studied on its own. Adopting complex technological innovations in organizations has been studied previously [34] and applying results from that research topic to the adoption of modern release engineering practices would be a good topic for future research. However, when studying technical adoption in case studies, the social problems should be taken into consideration too, since they can explain the technical problems. Similarly, technical problems can explain social problems. For example, a problematic system design increases the complexity of the adoption.

The organizational distribution and system design explain many of the adoption problems. System design has been studied in multiple earlier investigations, but the effects of distribution are recognized only in this study to our knowledge. Thus, in future research, the distribution and system design should be taken into account when studying modern release engineering practices.

Finally, organizational context and used development processes, such as stage-gate process, can explain the adoption problems. Researchers should take these into account when comparing the practices between organizations.

5.4 Implications to practice

Based on this dissertation, the practitioners can measure their release engineering maturity with the time needed for deployment stabilization. In addition, practitioners can focus on how well their build, test and deployment processes have been automated, when assessing the maturity.

Practitioners can assess the suitability of their system design for the modern release engineering practices. In addition, when designing new systems, practitioners can consider the testability and deployability of the systems, if they aim to implement modern release engineering practices.

Practitioners in distributed organizations, with stage-gate processes or otherwise difficult organizational contexts can see that implementing modern release engineering practices in their context might be difficult. Furthermore, they can prepare for the difficulties before committing to the adoption of modern release engineering practices.

5.5 Threats to validity

In this section, we discuss the threats to validity of our results. In Section 3.2.5, we showed how the threats were mitigated. We use the validity types of case study research [40]: construct validity, internal validity, external validity and reliability. We discuss the threats to validity of the SLR also under these validity types.

5.5.1 Construct validity

Construct validity assesses whether the constructs operationalized in research are the same as what is investigated according to the research questions. In this dissertation, there is a threat to construct validity, because the terminology used for modern release engineering practices is ambiguous and used differently by different practitioners and researchers. For example, there is no clear and agreed difference between CI and CD, nor what is considered to be "continuous" enough to be accepted as CI. In addition, the definition of CD requires the software to be always releasable, but it is again questionable what is considered to be "releasable" and what is not.

5.5.2 Internal validity

Internal validity assesses whether the explanations based on the collected data are valid. We used interviews and software repository mining as our data collection methods. Including observation as a data collection method would have had better validity, as it does not have the same biases. Our data collection and analysis procedures included interpretation, which reduced the internal validity. Finally, the studied case organizations were large, and we could not interview all parts of the organizations.

5.5.3 External validity

External validity assesses whether the results are generalizable outside the studied cases. In the SLR, we identified 35 cases related to the subject. However, the cases were not presented identically in the primary articles and they did not use the same research questions as we used in the SLR. Furthermore, the articles did not provide enough context-dependent material for synthesizing and generalizing the results.

In the case studies, the case organizations were from two business domains. The dynamics of the domains are different from other domains, e.g., the web application domain where most of the successes in modern release engineering practices are reported. As reported in Publication V, it is typical in the studied domain to have release cycles of multiple months. In addition, three of the studied cases were large mature companies and one was a startup. Thus, we were not able to study a mid-sized company. The presented results are only generalizable to similar situa-

tions and contexts.

Finally, we investigated the phenomenon in three cases. There exist many more software organizations with the aims of implementing modern release engineering practices. It can be expected that the results in this dissertation can be extended by investigating further cases in different domains and situations.

5.5.4 Reliability

Reliability assesses whether the data and analysis are dependent on the researchers. The data collection and analysis phases included interpretative elements and thus, replicating the work by other researchers can produce different results.

6. Conclusions

In this section, we present the contributions of the dissertation and propose topics for future research.

6.1 Contributions of the research

This dissertation provides four contributions for the research problem: **what prevents organizations from adopting modern release engineering practices?** The contributions are the answers for the research questions of the dissertation.

6.1.1 RQ1. How are modern release engineering practices adopted?

Modern release engineering practices are adopted by automating the build, test and deployment activities. The automation activities result in decrease of deployment stabilization period. The adoption is similar to the adoption of complex technological innovations in organizations and requires adoption to happen on both organizational and individual levels.

6.1.2 RQ2. What problems have been faced when adopting modern release engineering practices?

The identified adoption problems concern primarily automation of build, test and deployment, and social aspects. Other problems explain or are consequences of these primary problems. The automated build and deployment can be too slow, complex or inflexible so that building or deploying the system takes too long or requires substantial maintenance effort. Automated tests can be too slow, unreliable or the tests might not be sufficient for being certain of the quality of the system. Finally, individuals might resist the change needed in the work practices, due to, e.g., lack of

resources or insufficient training. On an organizational-level, resources or coordination can be missing for the adoption to be successful.

6.1.3 RQ3. What explanations for the problems have been presented?

Identified explanations for the adoption problems were organizational distribution, system design, stage-gate development processes and organizational context. Organizational distribution explains the social adoption problems, because distribution makes it more difficult to communicate, motivate and coordinate the changes needed in the organization. System design explains the technical adoption problems, when the architectural requirements for modern release engineering practices, such as testability and deployability, were not taken into account when the system was designed. Stage-gate processes and organizational context explain all the adoption problems, as they can either limit the resources for the adoption or increase the need for resources by higher quality requirements.

6.1.4 RQ4. What solutions for the problems have been used or proposed?

We identified that solutions exist for both technical and social problems. Technical problems can be solved by changing the design of the system so that the system and its parts are more testable and deployable. When an organization has control over the production environment of the development system, the microservices architectural pattern can be used to allow deploying smaller parts of the system separately. Social problems can be solved by implementing case specific strategies for achieving organizational change. For example, specialized teams can concentrate on creating automated legacy test cases and supporting other teams to create test cases for new features. These solutions can require substantial resources and thus need support from the management.

6.2 Future work

We did not study the automation of the build or deployment activities, as it was not either feasible in the cases or the subject was not yet timely for the organizations. However, automation of the deployment is an important part of adopting modern release engineering practices, as we identified in

Publication III. This could be a subject in future studies.

As our work was conducted inductively, we did not employ existing theories during the data collection and analysis. However, only afterwards, we recognized the existence of adoption theories, e.g. [34], that could be employed for more structured investigation of the adoption.

Finally, the hypotheses presented in the discussion section of the dissertation can be validated further in future studies.

References

[1] B. Adams and S. McIntosh. “Modern Release Engineering in a Nutshell – Why Researchers Should Care”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Vol. 5. Mar. 2016, pp. 78–90. DOI: 10.1109/SANER.2016.108.

[2] J. Bosch. “Speed, Data, and Ecosystems: The Future of Software Engineering”. In: *IEEE Software* 33.1 (Jan. 2016), pp. 82–88. ISSN: 0740-7459. DOI: 10.1109/MS.2016.14.

[3] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional, Aug. 6, 2010. 512 pp. ISBN: 978-0-321-60191-9.

[4] Andrej Dyck, Ralf Penners, and Horst Lichter. “Towards Definitions for Release Engineering and DevOps”. In: *Proceedings of the Third International Workshop on Release Engineering*. RELENG ’15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 3–3.

[5] Tony Savor et al. “Continuous deployment at Facebook and OANDA”. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE2016. ACM Press, 2016, pp. 21–30. ISBN: 978-1-4503-4205-6. DOI: 10.1145/2889160.2889223.

[6] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. “Challenges When Adopting Continuous Integration: A Case Study”. In: *Product-Focused Software Process Improvement*. Vol. 8892. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 17–32. ISBN: 978-3-319-13834-3.

[7] Torvald Mårtensson, Daniel Ståhl, and Jan Bosch. “Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences”. In: *Product-Focused Software Process Improvement*. Springer International Publishing, Nov. 22, 2016, pp. 448–457. DOI: 10.1007/978-3-319-49094-6_30.

[8] Chuck Rossi et al. “Continuous Deployment of Mobile Software at Facebook (Showcase)”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. New York, NY, USA: ACM, 2016, pp. 12–23. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2994157.

[9] Hyrum Kurt Wright. “Release engineering processes, their faults and failures”. PhD thesis. The University of Texas at Austin, 2012.

[10] Timo O. A. Lehtinen, Mika V. Mäntylä, and Jari Vanhanen. “Development and evaluation of a lightweight root cause analysis method (ARCA method) – field studies at four software companies”. In: *Information and Software Technology* 53.10 (2011), pp. 1045–1061.

[11] Stephen P. Berczuk and Brad Appleton. *Software configuration management patterns: effective teamwork, practical integration*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[12] Srinivasan Desikan. *Software testing: principles and practice*. Pearson Education India, 2006.

[13] André van der Hoek et al. “Software Release Management”. In: *Proceedings of the 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC ’97/FSE-5. New York, NY, USA: Springer-Verlag New York, Inc., 1997, pp. 159–175. ISBN: 978-3-540-63531-4. DOI: 10.1145/267895.267909.

[14] James Roche. “Adopting DevOps Practices in Quality Assurance”. In: *Commun. ACM* 56.11 (Nov. 2013), pp. 38–43. ISSN: 0001-0782. DOI: 10.1145/2524713.2524721.

[15] G. Schermann et al. “Towards quality gates in continuous delivery and deployment”. In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 2016 IEEE 24th International Conference on Program Comprehension (ICPC). May 2016, pp. 1–4. DOI: 10.1109/ICPC.2016.7503737.

[16] Md Tajmilur Rahman et al. “Feature toggles: practitioner practices and a case study”. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR’16. ACM Press, 2016, pp. 201–211. ISBN: 978-1-4503-4186-8. DOI: 10.1145/2901739.2901745.

[17] Timothy Fitz. *Continuous Deployment*. Feb. 8, 2009. URL: http://timothyfitz.com/2009/02/08/continuous-deployment/ (visited on 09/08/2014).

[18] Martin Fowler. *Continuous Integration*. May 1, 2006. URL: http://martinfowler.com/articles/continuousIntegration.html (visited on 09/08/2014).

[19] *Jenkins CI*. URL: http://jenkins-ci.org/ (visited on 06/13/2014).

[20] John Downs, John Hosking, and Beryl Plimmer. “Status Communication in Agile Software Teams: A Case Study”. In: *Proceedings of the 2010 Fifth International Conference on Software Engineering Advances*. ICSEA ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 82–87. ISBN: 978-0-7695-4144-0. DOI: 10.1109/ICSEA.2010.20.

[21] *Docker containerization platform*. URL: https://www.docker.com/ (visited on 01/12/2017).

[22] *Ansible IT automation*. URL: https://www.ansible.com/ (visited on 01/12/2017).

[23] *Robot Framework for generic acceptance testing*. URL: http://robotframework.org/ (visited on 01/12/2017).

[24] Pilar Rodríguez et al. “Continuous deployment of software intensive products and services: A systematic mapping study”. In: *Journal of Systems and Software* 123 (Jan. 2017), pp. 263–291. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.12.015.

[25] Marko Leppänen et al. “The Highways and Country Roads to Continuous Deployment”. In: *IEEE Software* 32.2 (2015), pp. 64–72.

[26] Simo Mäkinen et al. “Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises”. In: *Information and Software Technology* 80 (Dec. 2016), pp. 175–194. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2016.09.001.

[27] Mika V. Mäntylä et al. “On rapid releases and software testing: a case study and a semi-systematic literature review”. In: *Empirical Software Engineering* 20.5 (2015), pp. 1384–1425. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-014-9338-4.

[28] Daniel Ståhl and Jan Bosch. “Experienced benefits of continuous integration in industry software product development: A case study”. In: *IASTED Multiconferences - Proceedings of the IASTED International Conference on Software Engineering, SE 2013*. 2013, pp. 736–743.

[29] Bogdan Vasilescu et al. “Quality and Productivity Outcomes Relating to Continuous Integration in GitHub”. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 805–816. ISBN: 978-1-4503-3675-8. DOI: 10.1145/2786805.2786850.

[30] Michael Hilton et al. “Usage, Costs, and Benefits of Continuous Integration in Open-source Projects”. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. New York, NY, USA: ACM, 2016, pp. 426–437. ISBN: 978-1-4503-3845-5. DOI: 10.1145/2970276.2970358.

[31] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. “Climbing the "Stairway to Heaven" – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development Towards Continuous Deployment of Software”. In: *Proceedings of the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. Washington, DC, USA, 2012, pp. 392–399. ISBN: 978-0-7695-4790-9. DOI: 10.1109/SEAA.2012.54.

[32] Alexander Eck, Falk Uebernickel, and Walter Brenner. “Fit for Continuous Integration: How Organizations Assimilate an Agile Practice”. In: *Twentieth Americas Conference on Information Systems*. Savannah, Georgia, USA, 2014.

[33] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. “On the journey to continuous deployment: Technical and social challenges along the way”. In: *Information and Software Technology* 57 (2015), pp. 21–31. ISSN: 0950-5849.

[34] Michael J. Gallivan. “Organizational Adoption and Assimilation of Complex Technological Innovations: Development and Applica-

tion of a New Framework". In: *SIGMIS Database* 32.3 (July 2001), pp. 51–85. ISSN: 0095-0033. DOI: 10.1145/506724.506729.

[35] Claes Wohlin and Aybüke Aurum. "Towards a decision-making structure for selecting a research design in empirical software engineering". In: *Empirical Software Engineering* (2014), pp. 1–29.

[36] B.A. Kitchenham. *Guidelines for performing systematic literature reviews in software engineering*. Keele University Technical Report, 2007.

[37] D. S. Cruzes and T. Dybå. "Recommended Steps for Thematic Synthesis in Software Engineering". In: *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. IEEE, Sept. 2011, pp. 275–284. ISBN: 978-1-4577-2203-5 978-0-7695-4604-9. DOI: 10.1109/ESEM.2011.36.

[38] Robert K Yin. *Case study research: Design and methods*. 2nd. Sage publications, 1994.

[39] Michael Q. Patton. *Qualitative Research & Evaluation Methods*. 3rd. Published: Hardcover. SAGE Publications, Jan. 2002. ISBN: 0-7619-1971-6.

[40] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164. ISSN: 1382-3256. DOI: 10.1007/s10664-008-9102-8.

[41] Shane McIntosh et al. "A Large-Scale Empirical Study of the Relationship between Build Technology and Build Maintenance". In: *Empirical Software Engineering* (Aug. 1, 2014). ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-014-9324-x.

[42] MB Buff Lawson. "In praise of slack: Time is of the essence". In: *The Academy of Management Executive* 15.3 (2001), pp. 125–135.

[43] Dorothy Leonard-Barton. "Implementing Structured Software Methodologies: A Case of Innovation in Process Technology". In: *Interfaces* 17.3 (June 5, 1987), pp. 6–17. ISSN: 00922102.

[44] Md Tajmilur Rahman and Peter C. Rigby. "Release Stabilization on Linux and Chrome". In: *IEEE Software* 2 (2015), pp. 81–88.

[45] S. Bellomo et al. “Toward Design Decisions to Enable Deployability: Empirical Study of Three Projects Reaching for the Continuous Delivery Holy Grail”. In: *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. June 2014, pp. 702–707. DOI: 10.1109/DSN.2014.104.

[46] L. Chen. “Towards Architecting for Continuous Delivery”. In: *2015 12th Working IEEE/IFIP Conference on Software Architecture*. 2015 12th Working IEEE/IFIP Conference on Software Architecture. May 2015, pp. 131–134. DOI: 10.1109/WICSA.2015.23.

[47] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. “The Intersection of Continuous Deployment and Architecting Process: Practitioners’ Perspectives”. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM ’16. New York, NY, USA: ACM, 2016, 44:1–44:10. ISBN: 978-1-4503-4427-2. DOI: 10.1145/2961111.2962587.

[48] Gerald Schermann et al. *An empirical study on principles and practices of continuous delivery and deployment*. PeerJ Preprints, 2016.

[49] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. “Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering”. In: *Information and Software Technology* 54.10 (Oct. 2012), pp. 1107–1124. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2012.04.006.

[50] Gary Gruver, Mike Young, and Pat Fulghum. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. 1st. Addison-Wesley Professional, 2012. ISBN: 0-321-82172-6 978-0-321-82172-0.

[51] A. Balalaie, A. Heydarnoori, and P. Jamshidi. “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture”. In: *IEEE Software* 33.3 (May 2016), pp. 42–52. ISSN: 0740-7459. DOI: 10.1109/MS.2016.64.

[52] Darja Šmite et al. “Empirical evidence in global software engineering: a systematic review”. In: *Empirical Software Engineering* 15.1 (Feb. 2010), pp. 91–118. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-009-9123-y.

Release engineering means the process of bringing the individual changes made to a software system to the end users of the software with high quality. Traditional release engineering processes have focused on maximizing the quality of a release, which has decreased the speed of the release. The modern release engineering practices emphasize using build, test and deployment automation and facilitating collaboration across functional boundaries, so that it is possible to achieve both speed and quality in the release engineering process. While some companies have been successful in adopting modern release engineering practices, other companies have found the adoption to be problematic. In this dissertation, we aim to understand what prevents organizations from adopting modern release engineering practices.

Aalto-DD 220/2017

ISBN 978-952-60-7713-0 (printed)
ISBN 978-952-60-7714-7 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS