Large scale continuous integration and delivery
Stahl, Daniel

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2017

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Chapter 1. Introduction

1.1 The Rise and Prominence of Continuous Integration and Delivery

Since its inception as an engineering discipline in its own right in the 1960’s, software engineering has come to play an ever more crucial role in our society. Fifty years ago, electronic and mechanical products were largely designed to serve a single function – phones were designed to make phone calls, television sets were designed to view television broadcasts, cars were designed to be driven and fridges were designed to preserve food – and their functionality was predominantly determined by their hardware: their physical circuits, their wiring and their mechanical systems.

Today, however, we live in an age where that functionality is increasingly implemented in software. It is not the case that hardware is unimportant – after all, the software requires hardware to execute – but the hardware is becoming ever more generic. In other words, it is not so much the physical hardware that sets two competing product models apart, as it is the software. Indeed, if one were to open up one’s phone, television set, fridge or car, one would find that much of the hardware is the same. Instead, the differentiating features have moved from the realm of hardware to the realm of software. Consequently, those products have also become more generic: we now use our phones to read the news and play games, we hold video conferences using our television sets, cars drive themselves and our fridges write shopping lists.

This is all made possible by software, and our ability to create ever more complex software algorithms at an ever higher level of abstraction. We find that the size of software roughly follows Moore’s law [Moore 1965], doubling every two years [Rooijmans 1996, Genuchten 2007]. At the same time, the safety criticality of software is also increasing: software controls major parts of the infrastructure in modern society, and its robustness to withstand accidental failures as well as malicious attacks is a key concern to not only engineers, but to all human beings.

This increasing size and complexity of software coupled with its growing importance and ubiquity has led the software engineering community to pursue more efficient, effective and reliable methods of producing software. One paradigm in this struggle is that of continuous software engineering practices, driven by the idea that “If something hurts, do it more often”.

One such pain point in software engineering has long been the challenge of integrating multiple, simultaneous changes to software. Large software systems are not the products of single master-mind software engineers: instead, they are the results of sustained collective efforts by tens, hundreds or thousands of engineers working together in parallel. Integrating the changes made by all these engineers has long been one of the major sources of delay, cost, risk and faults in software engineering, and methods to accomplish this integration in a reliable yet efficient manner have been sought for decades.

The scene thus set, continuous integration emerged in the late 1990’s, proposing that the best way to solve this problem is not to do fewer and larger integrations, but instead more frequent smaller ones. Similarly, other continuous practices have been proposed and adopted by the industry, following the same logic: if it is better to integrate as frequently as possible, then why not also test, deliver, release and deploy as frequently?
Today, continuous practices in software engineering have long since ceased to be a curious novelty, but have become firmly established in the industry mainstream. The precise moment of their birth is up for debate: depending on whether one counts the introduction of continuous integration as a term by [Grady 1994], the definition of Extreme Programming in 1997 or its subsequent popularization by [Beck 2000], these practices are now in their early twenties or late teens. During that time they have not only greatly influenced industry practice, but also stirred substantial interest in the research community, not least with regards to their effect on related aspects of software engineering, such as configuration management, traceability, communication, testing and planning.

Today we find ourselves at a point where continuous integration and delivery – along with the related practice of continuous deployment – serve a mission critical role in large parts of the software industry: the fully automated continuous integration and delivery pipeline is the method of transforming source code changes into tested and documented product revisions with known content, quality and functionality.

At the same time, these pipelines are highly complicated and complex systems in their own right, particularly in large scale contexts, requiring significant effort and resources to design, implement and maintain. Yet they are quickly becoming not so much a competitive edge as a simple hygiene requirement: consumers have grown used to, and indeed come to expect, continuous deployment of new software to their electronic devices. Today, everything from desktop operating systems and mobile phone apps to cars are continuously updated with new and – hopefully improved – versions without user interaction. At the same time these expectations are seeping into business-to-business segments of the software industry, where e.g. network and telecommunications operators increasingly seek continuous deployment of software from their equipment vendors.

This means that not only is the efficacy of these continuous integration and production pipelines of critical importance – that is, how quickly and cheaply they can churn out new versions and improve R&D efficiency by e.g. providing feedback to developers – but their quality as well: a single bad revision let slip by these systems has the potential to wreck a computer, crash a car or shut down a power plant. We live in an increasingly software-intensive and software-dependent world, and the systems we use to build, test and deliver that software is a crucial concern in modern society.

Apart from its importance, what makes the field of continuous integration and delivery a fascinating topic of both research and practice is its multidisciplinary nature. Continuous practices affect the entire software production chain: from requirement handling, issue tracking and traceability, through programming, compilation, linking and integration to testing, analysis, packaging, documentation and deployment. Along the way it touches upon a long list of related fields, many of them not strictly software engineering fields: project planning, information visualization, organizational theory, psychology, team behaviors, communication, resource optimization, customer relations and sales, to mention a few. Consequently, it requires a truly holistic view of the software engineering process and constantly offers new unexpected perspectives and insights – particularly at the boundary where technology and human behavior meet.

This thesis represents one small step in the attempt to better understand, support and evolve the practices of continuous integration and delivery. Because of the multidisciplinary nature of the field it actively and deliberately moves between multiple points of view, seeking multiple angles of attack to address the larger goal of making great software better and faster.
1.2 Problem Statement

Based on the reasoning in the previous section, the starting point of this thesis is the following problem statement:

The software engineering practices of continuous integration and delivery have since their inception evolved from radical avant-garde ideas to become firmly established in the industry mainstream, to the point where at least ostensible adoption is often taken for granted. Their ubiquity in multiple industry segments, including safety-critical systems, places great demands on not only their efficacy, but also their quality in the sense of preventing faulty and potential dangerous software from being released into the wild. At the same time, practitioners witness and report stark differences in interpretations, implementations and experiences of continuous integration and delivery implementations. This is disconcerting as it pulls into question the ability to reliably adopt these practices with the required level of efficacy and quality. Consequently, an improved understanding of their implementation and their impact on software engineering, along with improved methods and techniques for the adoption, design and evolution of continuous integration and delivery pipelines, is urgently called for.

This problem statement in turn has given rise to the fundamental question of How can continuous integration and delivery practices be effectively, efficiently and reliably adopted, and what is the impact of such adoption? This fundamental question can in turn be broken down, and a number of sub-questions can be derived from it (colored in purple), as shown in Figure 1.

Other questions relate specifically to the two respective practices, and can only be formulated by first defining those practices in detail. This is surprisingly problematic, as there are multiple definitions and interpretations of various continuous practices in use. To exemplify, [Humble 2010] discusses the confusion regarding continuous integration, delivery and release, while [Rodriguez 2016] concludes that while some authors separate continuous delivery and deployment, others “use the terms continuous deployment and continuous delivery interchangeably”.

This lack of consensus regarding terminology renders the community great harm and impedes progress for practitioners and researchers alike: unless one is clear on what one describes, studies, presents recommendations for or otherwise reports on, it is very difficult for the community to benefit from and build upon those findings. Consequently, to gain increased clarity short-term and consensus long-term, the definitions of continuous integration and delivery used here are developers frequently integrating their changes with a common development branch, which is frequently and rapidly built and tested to evaluate those changes and treating each of those changes as a release candidate, to be frequently and rapidly evaluated through one's continuous delivery pipeline, and that one is always able to deploy and/or release the latest working version, but may decide not to for business reasons, respectively (colored blue in Figure 1).

In the case of continuous integration, it is relevant to ask how each of the three parts of the practice can be achieved: frequent integration of changes, frequent and rapid builds, and frequent and rapid testing. Similarly, in the case of continuous delivery, how can the pipelines be reliably designed, how can release candidates be frequently and rapidly evaluated, and how can they be made to meet any applicable legal and regulatory expectations, so that they are actually deployable and/or releasable?
This latter question is highly significant and arguably often overlooked. Some types of software, such as various online services, can be deployed under little or no regulatory constraints. In other cases, on the other hand, such as the safety-critical software of self-driving cars, telecommunications or airplanes, what has been changed, why it has been changed and how it has been verified must be rigorously documented and accounted for.

This is but one example of the multidisciplinary nature of continuous integration and delivery. Its adoption is not only a software engineering challenge, an organizational challenge, a legal and regulatory challenge or a communication and human behavior challenge: it is all of the above.

1.3 Research Objectives

Based on the problem statement and its breakdown in the previous section, the following research objectives have been phrased:

- Identify and investigate the impact of continuous integration and delivery adoption practices in an industry context.
• Identify and investigate any impediments to continuous integration and delivery adoption, particularly related to frequent integration, frequent and rapid builds or frequent and rapid testing.

• Present industry-validated systematic and reliable methods for the design and description of continuous integration and delivery implementations, allowing stakeholders of different backgrounds to align on a single comprehensive view of their production pipeline, enabling design and analysis of hypothetical implementations and supporting case-to-case comparison of de facto practice to foster dissemination of good practice.

• Investigate solutions to the problem of rapidly and frequently documenting and analyzing release candidates to meet legal and regulatory demands.

• Investigate solutions to the problem of frequent and rapid evaluation of release candidates in the continuous delivery pipeline.

1.4 Research Methodology

This section describes the employed research methodology – "the overall approach to the entire process of the research study" [Collis 2009]. There are many methods for collecting data as well as the subsequent analysis of and drawing of conclusions from that data. Hence the methodology to apply in any particular study must be chosen carefully, and shall be considered in light of the type of research results to be achieved by the study.

The next section describes the types of research results achieved in this thesis, followed by the research process and the research methods. The methodology of each individual chapter of the thesis is is presented in detail in Section 1.8.

1.4.1 Types of Research Results

There are many types of research results manifest in software engineering literature. [Shaw 2002] lists the following eight types:

• **Procedure or technique:** A new or better way to do some task, such as design, implementation, measurement or evaluation

• **Qualitative or descriptive model:** Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis

• **Empirical model:** Empirical predictive model based on observed data

• **Analytic model:** Structural model precise enough to support formal analysis or automatic manipulation

• **Notation or tool:** Formal language to support technique or model

• **Specific solution:** Solution to application problem that shows use of software engineering principles; careful analysis of a system or its development

• **Answer or judgment:** Result of a specific analysis, evaluation, or comparison

• **Report:** Interesting observations, rules of thumb
Of these eight, this thesis presents the following five: Procedure or technique; Qualitative or descriptive model; Notation or tool; Answer or judgment; Report. Section 1.8 presents the type of research results per individual chapter.

1.4.2 Research Process

[ Bailey 2012] defines research as "a systematic process based on the scientific method that facilitates the identification of relationships and determination of differences in order to answer a question". This is an iterative process involving multiple steps.

There are two perspectives from which to regard the research process of this thesis. One is the overarching process of moving from problem identification to solutions over the studies included in the thesis. In such broad terms, each part of the thesis represents its own respective step in that process:

- **Part I: Continuous Integration and Delivery Practice Impediments and Divergence** focuses on problem identification. The practices of continuous integration in delivery in industry and their descriptions in literature are studied and described, and theories as to the underlying reasons for observed problems and differences are formulated.

- **Part II: Continuous Integration and Delivery Modeling and Architecture** identifies, following on the work presented in Part I, the lack of systematic methods for the description and design of continuous integration and delivery systems as a problem to be solved. Through several studies such methods are iteratively designed, applied, analyzed and refined, leading up to the proposal of the Cinders architecture framework.

- **Part III: Continuous Integration and Delivery Traceability** takes a more forward-looking stance by identifying the problem of traceability in software engineering in general, and in continuous integration and delivery – with their emphasis on high speed and frequency – in particular, as a serious software engineering concern. It then moves on to study the availability of solutions that meet the continuous delivery requirements on rapid traceability analysis, and analyzes the open source framework Eiffel in this regard. It then looks even further beyond that, identifying more efficient testing practices in general and dynamic test case selection in particular as a problem to be studied further, with possible solutions to be found given the traceability capabilities afforded by Eiffel.

This iterative process is also shown in Figure 2, where Chapters 2-6 (colored red) of Part I address the fundamental questions of identifying consequences and challenges, with Chapter 6 serving as a bridge to set the scene for the next level. Chapter 7-9 (colored yellow) of Part II subsequently seek to answer how the practice can be improved, while Chapter 10-11 (colored green) of Part III look ahead, asking how these practices may enable further improvements to software engineering practice.
The second perspective is that of the process followed in each individual chapter. This is discussed in Section 1.8, and also addressed in the individual chapters themselves.

### 1.4.3 Research Methods

[Easterbrook 2008] identifies the five research methods most relevant to software engineering:

- **Controlled Experiments (including Quasi-Experiments)** strive to test one or more hypotheses by manipulating one or more independent variables and measuring any effects on dependent variables through experimentation.

- **Case Study** is "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" [Yin 2002]. Such case studies may be either exploratory, used as initial investigations to build theories, or confirmatory, for the purpose of evaluating, comparing and possibly refuting theories.

- **Survey Research** investigates a broad yet well-defined population to identify characteristics. While the use of questionnaires is common, other data collection techniques such as the parsing of logs can be used.
• **Ethnographies** focus on culture and sociological aspects of interactions within a community, emphasizing the importance of field observation while consciously striving to avoid imposing any preconceptions on behalf of the researcher. In software engineering this method can be used to analyze cultures within communities of practitioners.

• **Action Research** strives to intervene in the studied situation and solving the observed problems, while at the same time analyzing and documenting that experience.

In addition to these five, a powerful research method is that of literature reviews:

• **Literature Reviews** are "a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest" [Kitchenham 2004]. The studies included in the review are called *primary* studies, while a literature review is a *secondary* study. Consequently, a literature review by definition relies on other work being done in the area of interest – typically employing one or more of the five methods listed by [Easterbrook 2008]. Literature reviews can be used in several contexts, such as formulating theories, assessing state of the art or evaluating hypotheses.

Of the above methods, the chapters in this thesis use the following four: Case Studies; Survey Research; Action Research; Literature Reviews. Section 1.8 presents the methods used per individual chapter.

### 1.5 Research Questions

This section presents the research questions driving the research presented in this thesis. As described in more detail in Section 1.6, the thesis is structured into three distinct parts, each addressing continuous integration and delivery from its own respective point of view. Consequently, the research questions are presented for each individual part in Table 1.

Note that several of the chapters do not pose explicit research questions. In these cases, the questions presented in Table 1 are phrased specifically for the purpose of providing the reader with an overview of questions addressed throughout the thesis. To exemplify, Chapter 2 presents four hypotheses which are here rephrased and presented as research questions.

As shown in Table 1 and discussed further in Section 1.6, the research questions in Part I very much focus on understanding the nature of continuous integration and differences in its interpretation and implementation: what are the benefits, what are the impediments, which are the points of divergence in the practice, and what are the causes of that divergence?
Table 1: Thesis research questions.

Part II goes on to search for ways of documenting, describing and analyzing implementations of continuous integration and delivery. Through several iterations of design, industry application and evaluation of these techniques, Chapter 9 arrives at the Cinders architectural framework.
Part III, on the other hand, does not build upon Part II. Instead it asks related questions stemming from industry experience: given these large scale implementations of continuous integration and delivery, how are we to meet the ever present demands on traceability? And going one step further, given that level of traceability infrastructure and capability, what can we do to further improve our software engineering practices?

From the perspective of the problem domain breakdown (see Figure 1), each of these research questions can be mapped to one or more of the leaf nodes of that graph, as discussed in Section 12.1.

1.6 Thesis Outline and Article Overview

The main body of this thesis consists of articles published in major journals and conferences, as well as a contributed book chapter. Several included articles are in review or in press. Each article, its state, background and place in the thesis is discussed below. All included articles have largely been included as is, with the exception of minor layout and formatting adjustments.

In this thesis the articles are not included chronologically, but rather structured thematically into three parts. The reason for this is that each part deals with the field of continuous integration and delivery from a distinct and partly independent point of view.

1.6.1 Part I: Continuous Integration and Delivery Practice Impediments and Divergence

Part I investigates continuous integration and delivery practices in industry, establishing that there is a high degree of divergence in how they are implemented, in what their experienced benefits are and in the degree to which continuity is actually achieved. One conclusion of these findings is that better techniques for modeling and communicating continuous integration systems is required, and ends by presenting an initial proposal for such a modeling technique.

- **Chapter 5** Ståhl, D., Mårtensson, T., & Bosch, J. (2017). The continuity of continuous integration: correlations and consequences. In press; accepted by *Journal of Systems and Software*.
1.6.2 Part II: Continuous Integration and Delivery Modeling and Architecture

Part II proceeds to study the problem of modeling continuous integration and delivery systems, evolving the model proposed in Part I and combining it with methodology proposed in related work. This work and the experiences gained from application in multiple industry cases results in the proposal of Cinders, a unified architecture framework for continuous integration and delivery systems.


1.6.3 Part III: Continuous Integration and Delivery Traceability

Part III recognizes that traceability in software engineering is a crucial problem lacking adequate attention and support from tooling and infrastructure in the industry. It is explained how continuous integration and delivery adoption threatens to exacerbate the consequences of this lack, as its focus on speed and high frequency of release candidate production precludes manual solutions to the problem. In this context Eiffel, an open source framework for continuous integration and delivery, and its ability to satisfy real time traceability needs is investigated. Finally, the implications of Eiffel usage for dynamic runtime selection of test scope are discussed.


1.7 Applicability of the Research Results

The software engineering practices of continuous integration and delivery are widely used throughout the software industry. Consequently the research presented in this thesis is, on the whole, generally applicable to software engineering at large. That being said, several chapters present studies which are conducted in the context of a specific domain, or focused on the specific challenges of certain types of software engineering – particularly that of large scale projects involving large numbers of software engineering professionals of multiple roles. Table 2 presents the focus and context of each chapter.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Focus</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>2: Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study</td>
<td>Large scale</td>
<td>Industry; large scale</td>
</tr>
<tr>
<td>3: The Impact of Agile Principles and Practices on Large-Scale Software Development Projects</td>
<td>Large scale</td>
<td>Industry; large scale</td>
</tr>
<tr>
<td>4: Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences</td>
<td>Embedded systems</td>
<td>Industry; large scale; embedded systems</td>
</tr>
<tr>
<td>5: The Continuity of Continuous Integration: Correlations and Consequences</td>
<td>Large scale</td>
<td>Industry; large scale; embedded systems</td>
</tr>
<tr>
<td>6: Modeling Continuous Integration Practice Differences in Industry Software Development</td>
<td>General</td>
<td>General</td>
</tr>
<tr>
<td>7: Automated Software Integration Flows in Industry: A Multiple-Case Study</td>
<td>General</td>
<td>Industry</td>
</tr>
<tr>
<td>8: Industry Application of Continuous Integration Modeling: A Multiple-Case Study</td>
<td>General</td>
<td>Industry; large scale</td>
</tr>
<tr>
<td>9: Cinders: The Continuous Integration and Delivery Architecture Framework</td>
<td>General</td>
<td>Industry</td>
</tr>
<tr>
<td>10: Achieving Traceability in Large Scale Continuous Integration and Delivery</td>
<td>Large scale</td>
<td>Industry; large scale</td>
</tr>
<tr>
<td>11: Dynamic Test Case Selection in Continuous Integration: Test Result Analysis using the Eiffel Framework</td>
<td>General</td>
<td>General</td>
</tr>
</tbody>
</table>

Table 2: Focus and context of research studies.

### 1.8 Detailed Overview of Research Processes and Methods

This section presents the research process and methods of each individual chapter.
1.8.1 Chapter 2: Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study

This chapter presents a literature review and a multiple-case study of industrial experiences of continuous integration among software professionals working in large scale development projects.

- **Type of research result:** Answer or judgment; the chapter answers the question of how software engineers perceive the benefits of their respective continuous integration implementations, and how those perceptions differ, particularly examining four continuous integration benefits proposed in related work.

- **Research process:** The presented research begins by reviewing related work, and thereby phrasing hypotheses as to the potential benefits of continuous integration. These hypotheses are then investigated through the study of four industry cases and interviews with the engineers employed in those cases.

- **Research methods:** Literature Review; related work is studied in search of explicit claims to the benefits of continuous integration, informing the subsequent interviews. Case Study; four independent product development projects, considered representative for the studied company, are investigated, primarily by interviewing a total of 22 engineers of multiple working in those projects.

1.8.2 Chapter 3: The Impact of Agile Principles and Practices on Large-Scale Software Development Projects

Chapter three presents an explorative study of the impact of agile practices, including continuous integration, in two separate cases.

- **Type of research result:** Answer or judgment; the study presents a list of found effects of agile practices.

- **Research process:** The study starts by identifying the problem of lacking empirical studies on the effects of agile software development in a large-scale, industrial setting. A conceptual framework for effects of agile software development was constructed based on the results of a literature review, and evidence for these effects in the two studied cases is gathered through a web-based survey.

- **Research methods:** Literature Review; published literature was searched for effects of agile software development, the results of which formed the basis of a conceptual framework. Survey Research; the main method of data collection was an online survey distributed to 240 project members across the two studied cases.

1.8.3 Chapter 4: Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences

This chapter summarizes experiences from applying continuous integration in software-intensive embedded systems contexts.

- **Type of research result:** Report; a list of seven topics representing potential impediments for the application of continuous integration to embedded systems development is presented, and mapped to the seven corner stones of the practice listed by [Duvall 2007].
• **Research process:** The starting point of the study is the experience that successful implementation of continuous integration is in many cases far from trivial, and the intuition that the nature of embedded systems development poses specific challenges. To investigate this, experiences from two independent cases are reported, compared and analyzed in the context of Duvall’s seven corner stones.

• **Research methods:** Case Study; the study reports and discusses experiences and difficulties from two separate cases, and then compares those experiences to draw conclusions.

### 1.8.4 Chapter 5: The Continuity of Continuous Integration: Correlations and Consequences

This chapter searches for correlations between size and the continuity of various continuous integration implementations.

• **Type of research result:** Procedure or technique; Qualitative or descriptive model; Notation or tool; Answer or judgment; Report.

• **Research process:** This chapter notes that not all ostensibly continuously integrating projects are as continuous, and hypothesizes that this is related to size. Various methods of measuring size and continuity, respectively, are then discussed and a set of relevant metrics decided upon. Data is then collected and analyzed, both exploratively and by examining correlations between collected metrics of size and continuity, respectively.

• **Research methods:** Survey Research; this study mines software repository logs to analyze the behavior of 1,049 developers from six cases over a two month period. Case Study; the quantitative information gathered from software repository logs was complemented by qualitative information on process and practice in each of the studied cases.

### 1.8.5 Chapter 6: Modeling Continuous Integration Practice Differences in Industry Software Development

Influenced by the study presented in Chapter 2, this chapter performs a literature review to investigate divergence in continuous integration practice as reported in published literature.

• **Type of research result:** Answer or judgment; it is found that there is a high degree of divergence with regards to interpretation and/or implementation of continuous integration, specifically in 16 identified areas. Qualitative or descriptive model; based on the results of the literature review, a descriptive model addressing the 16 identified points of divergence is proposed.

• **Research process:** Based on the findings presented in Chapter 2, this chapter begins by hypothesizing that the observed differences in experience may be – at least in part – due to differences in actual practice. To investigate this, a systematic literature review is conducted, confirming a high degree of divergence. Identifying the problem that a clear, unambiguous method for expressing such implementations is lacking, a model addressing the 16 identified areas of divergence is proposed. The applicability of the model is subsequently demonstrated in an illustrative case study.

• **Research methods:** Literature Review; a systematic literature review of descriptions of continuous integration practice is conducted.
1.8.6 Chapter 7: Automated Software Integration Flows in Industry: A Multiple-Case Study

This chapter studies continuous integration practices in five independent cases through application of the model proposed in Chapter 6 and interviews with software engineers in those cases.

• **Type of research result:** Answer or judgment; the studied modeling technique is confirmed to be applicable to a wide range of industry contexts. Report; six guidelines for design and implementation of continuous integration systems are presented.

• **Research process:** The research presented in this chapter starts with identification – based on previous work as well as practical experience – of the problem that continuous integration practice is rife with divergence and conflicting interpretations. To improve understanding of this problem, the methods of Chapters 2 and 6 are applied in tandem: interviewing software engineers regarding their experiences of continuous integration, and modeling the continuous integration systems, respectively. The resulting models are then compared in search of possible explanations to the differences in experienced effects, and guidelines resulting from this analysis for continuous integration systems are presented.

• **Research methods:** Case Study; this research investigates five independent cases and documents their respective practices and experienced effects, respectively.

1.8.7 Chapter 8: Industry Application of Continuous Integration Modeling: A Multiple-Case Study

In this chapter two continuous integration modeling techniques – Automated Software Integration Flows (ASIF) and Continuous Integration Visualization Technique (CIViT) – are applied to four industry cases.

• **Type of research result:** Qualitative or descriptive model; while the chapter does not introduce any new models per se, it proposes improvements to the investigated modeling techniques. Answer or judgment; it is concluded that the modeling techniques ASIF and CIViT can be favorably combined. Report; guidelines for and experiences from continuous integration modeling are presented.

• **Research process:** In a sense picking up where Chapter 7 ends, two problems are identified. First, there is a lack of research on how continuous integration modeling can be applied to benefit industry practitioners in their day-to-day work. Second, two continuous integration modeling techniques have been proposed in literature, and it is unclear how they relate to one another conceptually as well as in practice. Consequently the two models are applied in tandem to four live industry cases to assess their effectiveness.

• **Research methods:** Action Research; the studied continuous integration modeling techniques are applied to industry cases and their ability to address their needs and challenges is evaluated. Literature Review; an overview of the current state of continuous integration research is presented.
1.8.8 Chapter 9: Cinders: The Continuous Integration and Delivery Architecture Framework

This chapter formulates two suppositions: that an architecture framework can be defined which unifies existing continuous integration and delivery modeling techniques, and that such an architecture framework can be shown to address requirements derived from previous application of those techniques, respectively. These suppositions are then investigated, resulting in the design of a continuous integration and delivery architecture framework.

- **Type of research result:** Qualitative or descriptive model; the main research result of the chapter is the architecture framework Cinders.
- **Research process:** The chapter presents research driven by the two suppositions listed above. By reviewing literature, documented experiences from continuous integration and modeling are collected. Through thematic analysis these experiences are transformed into requirements on an architecture framework. A solution to the identified problems is then designed in the form of Cinders. This solution is subsequently evaluated through interviews with previous users of continuous integration and delivery modeling techniques, through analysis of requirements compliance, through application to live industry cases and through a group interview with continuous integration specialists.
- **Research methods:** Literature Review; literature is searched for experiences of continuous integration and delivery modeling, forming input to the architecture framework design. Action Research; the Cinders architecture framework is applied to industry cases and its ability to address their needs and challenges is evaluated.

1.8.9 Chapter 10: Achieving Traceability in Large Scale Continuous Integration and Delivery

This chapter consists of three separate parts: an investigation into the industry developed continuous integration and delivery framework Eiffel, identification of prominent traceability needs in the industry and validation of the Eiffel framework.

- **Type of research result:** Notation or tool; the chapter presents the Eiffel framework as a tool addressing the challenges of traceability in continuous integration and delivery. Procedure or technique; apart from the tool itself, the procedure of real-time *in situ* generation of trace links – as opposed to *ex post facto* analysis of documents – realized by the Eiffel framework is discussed.
- **Research process:** The chapter begins by investigating the Eiffel framework through perusal of documentation, informal discussions with continuous integration architects, demonstrations and participant observation. To identify traceability needs in large scale industry projects, 15 software engineering professionals in three cases are then interviewed and their responses thematically analyzed. Based on the identified needs, the Eiffel framework is subsequently validated using five methods: interviews with users of the framework, interviews with engineers not using the framework, comparison of traceability data gathering process with and without Eiffel, searching literature for solutions comparable to the Eiffel framework and observer participation.
**Research methods:** Case Study; processes and the behavior and experiences of software engineers in three independent cases are studied in order to investigate the validity of the Eiffel framework. Literature Review; related work is reviewed in order to answer the question "Which solutions to the traceability problem in a continuous integration and/or delivery context have been proposed in literature?".

### 1.8.10 Chapter 11: Dynamic Test Case Selection in Continuous Integration: Test Result Analysis using the Eiffel Framework

This chapter does not present any research study, but discusses traceability as prerequisite for dynamic run-time selection of test cases based on findings in previous work, along with experiences gained both as a researcher and a practitioner of continuous integration and delivery.

- **Type of research result:** Report; the inherent conflict between the emphasis on speed and frequency in continuous integration and delivery, on the one hand, and the need for thorough and extensive testing to achieve a high degree of confidence in release candidates, on the other, is discussed along with its implications for traceability.

- **Research process:** The chapter focuses on problem identification. By analyzing previous findings and experiences, it points to an area in need of further investigation.

- **Research methods:** The chapter does not present any study of its own, but discusses studies performed in previous work. Consequently, no specific research method was employed.

### 1.9 Related Publications

Two related publications are not included in this thesis.


### 1.10 Contributions to the Articles

This thesis contains articles resulting from research and work by multiple authors. This section describes my personal contributions to each of the articles.
1.10.1 Chapter 2


My contribution to this article was ideation, research design, data collection, analysis and the writing of the paper. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.2 Chapter 3


This article is based on the MSc thesis of Lagerberg and Skude. My contribution to the thesis and the subsequent article was ideation, supervision, coaching and reviewing. Lina Lagerberg and Tor Skude collected the data, analyzed it and wrote the article. Per Emanuelsson and Kristian Sandahl acted as reviewers and mentors to Lagerberg and Skude.

1.10.3 Chapter 4


Research design, data gathering and analysis for this article was a joint effort between myself and Torvald Mårtensson, with Mårtensson performing the larger share of the work and writing the resulting paper. I reviewed the paper, while Jan Bosch acted as mentor, sounding board and reviewer.

1.10.4 Chapter 5


The research design of this article was jointly developed between myself and Torvald Mårtensson. I collected the primary case data and parts of the validation data, analyzed it and wrote the paper. Mårtensson collected the majority of the validation data and acted as a reviewer. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.5 Chapter 6


My contribution to this article was ideation, research design, data collection, analysis and the writing of the paper. Jan Bosch acted as mentor, sounding board and reviewer.
1.10.6 Chapter 7

My contribution to this article was ideation, research design, data collection, analysis and the writing of the paper. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.7 Chapter 8

My contribution to this article was ideation, research design, data collection, analysis and the writing of the paper. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.8 Chapter 9

My contribution to this article was ideation, research design, data collection, analysis and the writing of the paper. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.9 Chapter 10

My contribution to this article was ideation, research design and the writing of the paper. Data collection and analysis was performed jointly by myself and Kristofer Hallén, who also acted as a reviewer. Jan Bosch acted as mentor, sounding board and reviewer.

1.10.10 Chapter 11

This book chapter was the result of Jan Bosch being invited to contribute to Analytic Methods in Systems and Software Testing and subsequently inviting me to co-author it. My contribution was to propose the topic and write the chapter. Jan Bosch acted as mentor, sounding board and reviewer.
Part I:
Continuous Integration and Delivery
Practice Impediments and Divergence