Laukkanen, Eero; Lehtinen, Timo; Itkonen, Juha; Paasivaara, Maria; Lassenius, Casper

Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization

Published in:
Proceedings of the ACM/IEEE 10th International Symposium on Empirical Software Engineering and Measurement

DOI:
10.1145/2961111.2962608

Published: 01/01/2016

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

Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization

Eero Laukkanen
Department of Computer Science
PO Box 15400
FI-00076 AALTO, Finland
eero.laukkanen@aalto.fi

Timo O.A. Lehtinen
Department of Computer Science
PO Box 15400
FI-00076 AALTO, Finland
timo.o.lehtinen@aalto.fi

Juha Itkonen
Department of Computer Science
PO Box 15400
FI-00076 AALTO, Finland
juha.itkonen@aalto.fi

Maria Paasivaara
Department of Computer Science
PO Box 15400
FI-00076 AALTO, Finland
maria.paasivaara@aalto.fi

Casper Lassenius
Department of Computer Science
PO Box 15400
FI-00076 AALTO, Finland
casper.lassenius@aalto.fi

ABSTRACT

Context: Continuous delivery (CD) is a development practice for decreasing the time-to-market by keeping software releasable all the time. Adopting CD within a stage-gate managed development process might be useful, although scientific evidence of such adoption is not available. In a stage-gate process, new releases pass through stages and gates protect low-quality output from progressing. Large organizations with stage-gate processes are often hierarchical and the adoption can be either top-down, driven by the management, or bottom-up, driven by the development unit.

Goal: We investigate the perceived problems of bottom-up CD adoption in a large global software development unit at Nokia Networks. Our goal is to understand how the stage-gate development process used by the unit affects the adoption.

Method: The overall research approach is a qualitative single case study on one of the several geographical sites of the development unit. We organized two 2-hour workshops with altogether 15 participants to discover how the stage-gate process affected the adoption.

Results: The stage-gate development process caused tight schedules for development and process overhead because of the gate requirements. Moreover, the process required using multiple version control branches for different stages in the process, which increased development complexity and caused additional branch overhead. Together, tight schedule, process overhead and branch overhead caused the lack of time to adopt CD. In addition, the use of multiple branches limited the available hardware resources and caused delayed integration.

Conclusions: Adopting CD in a development organization that needs to conform to a stage-gate development process is challenging. Practitioners should either gain support from the management to relax the required process or reduce their expectations on what can be achieved while conforming to the process. To simplify the development process, the use of multiple version control branches could be replaced with feature toggles.

ESEM '16, September 08-09, 2016, Ciudad Real, Spain
ACM 978-1-4503-4427-2/16/09.
http://dx.doi.org/10.1145/2961111.2962608

CCS Concepts

•**Software and its engineering** → **Agile software development; Waterfall model; Software testing and debugging;** *Software configuration management and version control systems;*

Keywords

continuous delivery; stage-gate process; case study

1. INTRODUCTION

High-performing software companies have adopted the development practice of continuous delivery (CD) [14] to be able to execute rapid releases [22]. Traditionally, rapid releases have not been possible due to bottlenecks in the delivery process, such as the need for a long code freeze period before a release, extensive manual testing and error-prone manual releases. The CD practice is achieved with discipline and by automating the delivery, including building, testing and deploying the software.

While successful adoptions of CD exist [24], some companies have found it challenging to adopt [5, 3, 16]. Previous studies have found many adoption problems, but they have not investigated the causal mechanisms behind the problems. Better understanding of the causes of the problems would make it easier to adopt CD or allow evaluating to what extent CD adoption is suitable for the adopter.

In this study, we investigate the emergent problems of a bottom-up CD adoption within a stage-gate [4] managed development process. We use the term bottom-up to indicate that the adoption was driven by a development unit

instead of its management. Traditional processes have been shown to cause problems when adopting CD [2], but the exact mechanisms have not been previously studied. We utilize root cause analysis [19] to systematically examine the mechanisms between the process and CD adoption problems.

We study a large distributed software development unit at Nokia Networks developing a complex software product in the telecommunications market. The unit had experienced problems when adopting CD [13] and the stage-gate process used by the unit was claimed to significantly hinder the adoption. We conducted two workshops to collect qualitative data in order to understand the relationship between the stage-gate process and CD adoption problems.

This paper is structured as follows: we introduce the terminology used in this study and review previous studies in Section 2. The case organization and our research methods are described in Section 3. The results of the study are represented in Section 4. Finally, the results are discussed in Section 5 and the study is concluded in Section 6.

2. RELATED WORK

In this section, we first explain the concept of continuous delivery (CD) and how it relates to other similar concepts. After that, we review previous research on the CD adoption paths and problems.

We do not know of any previous studies that have studied the CD adoption within a stage-gate development process. However, there are several previous studies that have investigated the usage of agile methods together with traditional processes. We summarize those studies in the last part of this section.

2.1 Continuous Delivery

A succinct definition for CD is given by Martin Fowler:

> *Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.*
>
> — Martin Fowler [10]

The term was coined by Humble and Farley [14] who introduced an ideal of software delivery practices and tools to achieve the goal—*the software can be released to production at any time*. The definition says *can be released* instead of *is released*; thus the discipline considers only the capability of the software development organization, while other constraints might not actually allow continuous releases. An extreme version of CD is *continuous deployment* [8], where all quality-checked changes are automatically released. Some previous authors have used the term continuous deployment when actually referring to continuous delivery (e.g. [3, 21]).

One simple reason why companies want to adopt CD is to shorten the software delivery cycle time. CD takes the practice of continuous integration (CI) [9] to the extreme (see Figure 1); each change is integrated all the way until it has been confirmed to be releasable. Thus, integration problems are discovered early and fixing them is cheaper. Also exploratory testing is simple, because the newest changes can be released to a user testing environment at any time. Finally, the release process becomes less risky when automated.

In practice, CD is achieved by developing and maintaining a *deployment pipeline* [14]. An example pipeline (Figure 1) consists of a version control system, commit stage, acceptance test stage and finally other stages or production

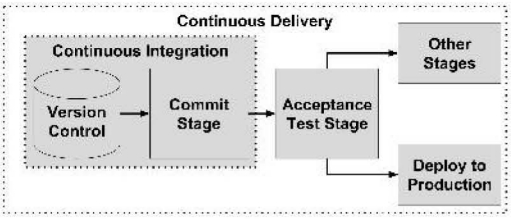

Figure 1: Conceptual definitions used in this study.

deployment. Anything that can change in the production environment is version controlled, including the source code, the application configuration, the database schema and the environment configuration. Whenever something needs to be changed, the change is applied to the version control and then the change creates an instance of the pipeline to be executed. During the execution, the change is evaluated whether it is releasable. Any discovered problem will halt the pipeline and depending on the issue, it is either fixed or the change is reverted to keep the software releasable.

2.2 Continuous Delivery Adoption Paths

To our knowledge, there are two previous studies that discuss adoption paths of CD directly.

Holmström Olsson et al. [12] propose an adoption path towards R&D as an experiment system, where "the entire R&D system responds and acts based on instant customer feedback". The steps in the path are traditional development, agile R&D organization, continuous integration, continuous deployment and finally R&D as an experiment system. Moreover, they suggest that continuous integration and further steps require changes outside the R&D organization: in product management, system validation and customers.

Eck et al. [6] propose that CI adoption consists of three stages: acceptance, routinization and infusion. However, they do not discuss the role of different organizational functions or processes in the adoption.

2.3 Continuous Delivery Adoption Problems

We found seven previous studies on CD adoption problems, shown in Table 1. Some studies focused on CI, but since it is a prerequisite for CD, they can be considered to address CD adoption problems, too. One distinct difference between studies of CI and CD is that only CD studies consider release problems.

Debbiche et al. [5] studied CI adoption challenges at a Swedish telecommunications company. They found many challenges synthesized under topics of mindset, tools and infrastructure, testing, domain applicability, understanding, code dependencies and software requirements.

Claps et al. [3] studied CD adoption challenges at Atlassian Software Systems, an Australian software company. They found 11 social and 9 technical challenges related to the adoption. The challenges were synthesized to categories of the need to be Lean, management driven adoption, changing responsibilities, the risks of adopting CD and the transition from CI to CD. Thus, their study was not focused on problems faced, but on the actions needed to succeed with the adoption.

Leppänen et al. [21] studied CD in the contexts of 15 information and communications technology companies in

Table 1: Summary of previous studies of CD adoption. The lack of studies about process problems is highlighted.

	[5]	[3]	[21]	[2]	[11]	[16]	[13]
Identified Problem Themes							
Build design							X
Product design		X	X		X	X	X
Integration	X	X				X	X
Testing	X		X		X	X	X
Release		X	X				
Human and Organizational	X	X	X	X	X	X	X
Process				**X**			
Resources		X	X	X	X	X	X
Data Collection Method							
Self-report				X	X		
Interviews	X	X	X			X	
Observation							X

Finland. Only one of the companies was reported to have adopted CD. However, CD was not even the goal for every company. Again, companies had faced multiple obstacles when transitioning towards CD: resistance to change, customer preferences, domain constraints, developer trust and confidence, legacy code considerations, duration, size, structure, different development and production environments and manual and nonfunctional testing.

Chen [2] reported how Paddy Power PLC, Irish bookmaking company, had adopted CD. He reported that the organization faced organizational, process and technical challenges. Of these challenges, organizational challenges were reported to be the biggest. This study is the only one to mention process challenges. Still, the connection between process and CD adoption is mentioned only briefly but not analyzed in depth.

Gmeiner et al. [11] reported how automated testing is used in the CD pipeline of an Austrian online business company. They reported lessons learned when establishing and operating the pipeline: ensure management commitment, take collective responsibility, establish test environment management, improve testability, manage test data provisioning, define the ownership of acceptance tests, acceptance tests should not compensate for missing unit tests, invest in the maintenance of automated tests and combine automated and manual testing.

Laukkanen et al. [16] studied CI adoption at an R&D program at Ericsson, a global telecommunications company. They found challenges of lack of time, difficult components, unstable tests, slow tests, insufficient testing environments, agreement on tools and global distribution of organization. The largest challenges were the lack of time, architecture of the product and global distribution of the organization.

Hukkanen [13] studied CI adoption at a software development unit at Nokia Networks, a global telecommunications company. He found challenges of CI practicalities, communication, testing, infrastructure and dependency management. In addition, he found out that the challenges were causally related to each other.

As a summary, previous studies show that similar problem themes are found in different cases. However, the causal mechanisms behind the problem themes have not been studied previously, except in the work by Hukkanen [13]. If the underlying causal relationships are not known when designing solutions for the problems, the solutions might turn out to be ineffective, solving only symptoms of the real causes. In this study, we attempt to reveal the underlying causes preventing the adoption of CD in a single case study.

In this case study, we study the same case organization as Hukkanen studied, but with a different research scope. Hukkanen focused on build failures and their causes. After Hukkanen had completed his study, we continued studying the same case by focusing on all CD adoption problems and by using a different kind of data collection method; Hukkanen observed build failure standup meetings, while we arranged root cause analysis workshops. Continuing studying the problems was logical, because the results by Hukkanen showed that the situation in the case did not improve despite multiple months of time passing during the study by Hukkanen.

2.4 Adopting Agile Methods within Traditional Development Processes

Adopting CD has not been studied specifically within traditional development processes. Nevertheless, there has been previous studies about combining agile methods with traditional processes. Here, we summarize those studies.

In a *stage-gate process* [4], new releases pass through stages and in between there are gates that protect low quality input entering the next stage. Karlström and Runeson [15] studied how agile software development could be integrated into stage-gate managed product development. The method under study was extreme programming (XP). They found it possible to integrate XP to the stage-gate process, given that certain success factors are present.

Van Waardenburg and van Vliet [26] studied what challenges there are when agile methods and plan-driven processes coexist. They conclude that they can coexist, but insist that good communication should be present between different parts of the organization using different processes.

Theocharis et al. [25] studied whether agile methods replace traditional processes altogether when adopted or do they coexist afterwards. They conclude that agile and traditional processes are often combined as hybrid approaches.

As a summary, previous studies have concluded that traditional processes and agile methods can coexist and adopting agile methods does not usually mean that traditional processes are removed. Communication between different people using different processes has been considered important. In this paper, we investigate whether CD can be adopted within a stage-gate process, when Scrum agile method is being used already.

3. METHOD

In this section, we first introduce the case organization. Second, we define our research goal and questions. Finally, we describe our data collection and analysis methods, followed by the validation of the results.

3.1 Case Organization

This study is based on a single case study [28] of a Nokia Networks software development unit developing a telecommunications software product. The product is produced by the unit and sold to the customers who take care of operating the product. In addition, there is a product management

unit who is managing the development unit. When we use the term *case organization* in this study, we refer to the development unit only.

The development unit is large and distributed across several sites and countries. A more in-depth description of the case can be found in [13]. Due to availability, we collected data from only one site of the unit. However, the data collected applies to the whole unit, because the same process and CD system was used at every site.

The development unit had begun to improve their build, deploy and test practices in early 2014. The main goals of the unit were to enable faster time-to-market in a competitive business environment and to reduce the code verification feedback time to developers. However, in an earlier investigation during 2014 by Hukkanen [13], the unit faced serious problems during the improvement, e.g. frequently failing builds and flaky tests. Despite the efforts put into the improvement activities, major problems still remained in early 2015 when this study was initiated.

It was believed at the unit that adopting CD would not require changes in the processes external to the unit, and thus the unit itself could drive the adoption (*bottom-up* adoption). The development unit had a vision of being able to adopt CD *internally*, meaning that while the other organizational functions, such as product management, sales and marketing, were still working with the old processes, the unit could continuously deploy the product in a releasable condition to a clone of a production environment. The participants used the term "continuous deployment" when referring to the vision.

> *Internally, the target would be continuous deployment, deploying to target hardware continuously [...] For continuous deployment, the management is not needed or changes in the processes.* — Workshop participant

To align with the vision of the unit, we define CD as adopted in the unit if *the software is built in such a way that it can be given to the downstream units at any time.* Thus, we exclude the automated deployment to production from this study, and consider only the capability to deliver verified software artifacts to the next phase in the stage-gate process.

3.2 Research Goal and Questions

The research goal of this study is *to understand how the stage-gate process affected CD adoption in the case organization.* Despite the efforts put into the adoption, there were direct signs of a dysfunctional CD practice, as shown by Hukkanen [13]. The research goal is achieved by answering the following research questions:

- RQ1. What direct signs of a dysfunctional CD practice did the case organization show?
- RQ2. What caused the direct signs of a dysfunctional CD practice in the case organization?
- RQ3. How did the stage-gate process used by the case organization explain the direct signs of a dysfunctional CD practice?

We identified three types of concepts that we use for describing the phenomenon in the case organization (see Figure 4):

- **Direct signs**: direct signs of a dysfunctional CD practice. They are the leaves of a cause-effect diagram.

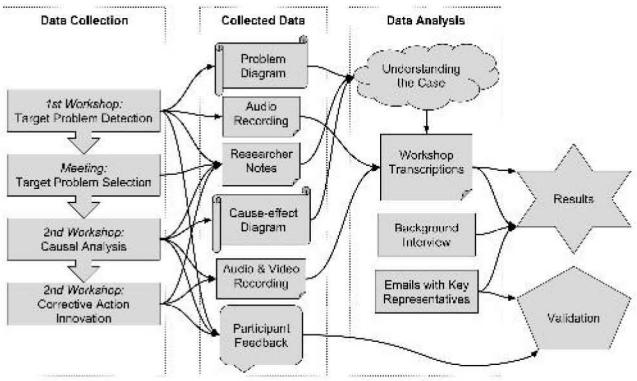

Figure 2: The research process of this study.

- **Mechanisms**: mechanisms that explain the direct signs. They are neither leaves nor roots of a cause-effect diagram.
- **Root causes**: root causes are factors specific to the case organization that are considered to be the last relevant mechanisms explaining the direct signs. They are the roots of a cause-effect diagram.

The overall research process is depicted in Figure 2. Next, we describe the process steps.

3.3 Case Selection Rationale

While initially the case selection was done for convenience, it turned out to be *revelatory* considering CD adoption within a stage-gate process. The rationale for a revelatory case study is described by Yin [28] as "when an investigator has an opportunity to observe and analyze a phenomenon previously inaccessible to scientific investigation."

3.4 Data Collection

We utilized a root cause analysis method named ARCA [19] as the data collection method of the study. The ARCA method is a retrospective method and it was designed to help developing corrective actions for a target problem, require low effort, be easy to use and be adaptable for different kinds of target problems. The method has been successfully applied for analyzing software project failures [19, 20].

In this study, we used the ARCA method to systematically collect mechanisms between the direct signs and root causes perceived by the case organization members. Next, we introduce the details about how ARCA was utilized. Data was collected between March and June 2015. The data collection consisted of four steps: target problem detection, target problem selection, causal analysis and corrective action innovation (see Figure 2). First, we arranged the target problem detection workshop. Then, the target problem selection step was performed with discussions with a key representative from the unit. Finally, the second workshop included the causal analysis and corrective action innovation steps.

3.4.1 Target Problem Detection

Ten people from various roles at the studied site, such as managers, developers and testers, took part in the first workshop. The participants were invited to first hear results from a previous study [13] and then continue with the workshop. Participation was not mandatory; participants could

decide whether they wanted to take part in the workshop or not. The workshop lasted for one hour and 45 minutes and it was audio recorded. Two researchers also took notes during the workshop.

During the workshop, results from an unpublished systematic literature review (SLR) [17] about CD adoption problems were first presented to the participants. The results were presented organized into the categories of design, integration, testing, release, people, organization, resources and tools problems when adopting CD. Next, the participants were asked what current CD adoption problems they were facing.

The participants were first given five minutes to individually write down experienced CD adoption problems. Then, every participant in turn described their problems, and each problem was placed by the first author on a problem diagram with the categories found in the SLR. The diagram was projected to the wall of the workshop room during the whole workshop.

The participants came up with many problems in all other categories except release and organization. However, organizational problems were discussed under an emergent "milestone thinking" category, which proved to be a critical problem in the case. Workshop participants used the term *milestone* as a synonym for time-based releases.

In retrospective, based on the audio recording of the workshop, some problems were categorized incorrectly into the SLR categories. In further replications, the categorization of the elicited problems could be done purely inductively, to avoid similar problems. However, this does not have any effect on the results of this study, because the results are based on a qualitative analysis done after the workshops.

After everyone had had a chance to discuss about the problems, the participants voted on which problems should be addressed in causal analysis workshop. Everyone had three votes. We instructed the participants to vote for problems that had a large impact on the adoption and could be prevented by the case organization.

3.4.2 *Target Problem Selection*

The results from the target problem detection were used in two further discussions to select focus points for the later parts of the investigation. The votes from the previous workshop did not determine the target problems directly, because the votes were uniformly divided between multiple problems.

The first three authors and a key representative from the case organization participated in the target problem selection. In addition, other key stakeholders from the case were consulted to gain advice, but the decision on how to proceed was done together by researchers and the key representative. The selected target problems for causal analysis were *support for multiple branches* and *milestone release model.*

3.4.3 *Causal Analysis*

In the causal analysis workshop, there were ten people present who were mainly software developers and people responsible for the CD system used by the unit. Again, the participants could decide themselves whether to take part in the workshop or not. The causal analysis step lasted for one hour and 30 minutes and it was audio and video recorded. In addition, the first author made reflective notes after the workshop.

Instead of asking for causes of the target problems, we investigated their effects. The effects proved to be more interesting to study than the causes. The decisions to do multiple branches and have milestones were given as granted to the developers, thus the causes of the decisions were not fully known for the participants.

During the workshop, the cause-effect diagram was constructed. The participants were first given five minutes to write down effects of the target problems individually. After that, each participant had a turn to describe their additions to the cause-effect diagram. The target problems were investigated individually, first milestone release model and after that support for multiple branches.

3.4.4 *Corrective Action Innovation*

Corrective action innovation was done during the same workshop as the causal analysis, except after a short break. It lasted for one hour and 11 minutes and was also audio and video recorded. However, only four people remained to participate the corrective action innovation workshop, so the results from this part are not that comprehensive. First, participants developed corrective actions in pairs based on the cause-effect diagrams. Next, the actions were described to other participants. Finally, the participants voted about the feasibility and effectiveness of the actions.

3.4.5 *Collected Data*

As a summary, we collected multiple kinds of data:

1. Problem and cause-effect diagrams from the workshops that represented a common understanding of the situation.
2. Audio recordings from both workshops and a video recording from the causal analysis workshop.
3. Notes from the target problem selection discussions.
4. Feedback forms that were given at the end of the workshops to the participants, see Section 3.6.

3.5 Data Analysis

The audio and video recordings were used for transcribing the workshops as text. The transcription was performed by the first author. The transcriptions were coded with the ATLAS.ti [1] qualitative data analysis tool. The codes were grouped into code groups and the groups were finally used to construct the narrative results of this study.

We constructed six code groups for the adoption problems: *architecture*, *branches*, *distribution*, *limited hardware*, *process* and *time pressure*. In addition, we found quotations that described why the process did not work in practice and grouped those codes under the code group *why the process does not work*. Those quotations described the tight schedule mechanism (see Section 4.4). After coding, we connected the identified mechanisms to the direct signs of dysfunctional CD practice (see Figure 4).

In addition to the data gained with the ARCA method, we had previously interviewed a member of the unit about the stage-gate process they used. Moreover, we triangulated the results of the study by discussing about them by email with key representatives.

3.6 Validation

The workshop recordings allowed rigorous analysis of the results. In addition, each workshop was accompanied by at least two researchers, mitigating single researcher bias.

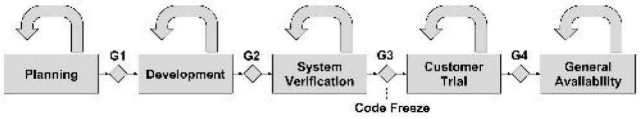

Figure 3: The stage-gate development process of the case organization.

The workshop participants were allowed to give anonymous responses by email, in case they did not feel safe to say them publicly in the group. There were only a few email replies, and it turned out that the participants were bold enough to say their opinions in the group.

Feedback was gathered from the participants after each workshop to evaluate the results from their perspective. The feedback was positive and the participants agreed that the workshop results truthfully described their situation, although the feedback gathered from the causal analysis and corrective action innovation steps was limited due to some participants leaving between those steps. Two key representatives from the case organization have read this paper and agree that the results are correct.

The selection of invited participants was done by a key representative in the unit in order to get knowledgeable participants. To guide the selection, we instructed that the participants' daily work should be related to the CD practice. Based on the participant feedback, the workshops had been relevant for the participants.

4. RESULTS

In this section, we present the results of the study. First, we describe the stage-gate development process used in the case organization. Second, we describe the direct signs of a dysfunctional CD practice the case organization was suffering from. Third, we describe why were the direct signs present by describing the root causes and mechanisms that explain the direct signs. Finally, we focus on how the stage-gate process used by the case organization explains the direct signs.

4.1 Stage-gate Development Process

During the workshops it became evident that the product development process used by the case organization hindered the CD adoption significantly. Thus, to understand the results of this paper, the external stage-gate process is described here. The product development was organized as a stage-gate process, as depicted in Figure 3. The process consisted of multiple stages which had gates between them. To proceed to a next stage, certain gate requirements needed to be met depending on the gate in question. The planning and development stages actually consisted of multiple stages, but separating them is not relevant for this paper. The round arrows visualize the iterative nature of different stages.

During the planning stages, specifications for new features were created and the features were grouped into content packs. Resources were then allocated for the content packs and the packs were scheduled for specific releases.

After the planning stage, the actual development started. This stage was performed by the case organization. During the development, the content packs were implemented and continuously tested. The case organization used Scrum to organize their development work during the development stage.

After the development was done, the content packs were tested as a whole during the system verification stage. This stage was performed by a unit other than the case organization. During the system verification, the product was used as the customer would install and use it. The system verification was executed on a production-like environment at the premises of Nokia Networks.

After the system verification stage, it was decided whether the content pack was ready for customer trials or not. This decision point had a specific date which was determined during the planning stage. If a content pack was ready, its source code was in principle frozen, meaning that only critical bug fixes could be applied to it afterwards, although in practice the code freeze was not strictly followed. The content packs were then tested in production-like environments of specific trial customers. If any bugs were found during this stage, they could be either fixed immediately or deferred to a later release.

Finally, if the customer trials were considered to be successful, the content packs could be made generally available for other customers as well.

Table 2: Summary of the direct signs of a dysfunctional CD practice in the case organization.

Problem	Description
Failing builds	CD pipeline builds were often failing and not fixed immediately afterwards.
Flaky tests	Some tests were flaky, meaning that they might fail randomly even when there was no issue in a code change.
Low test coverage	Test coverage was not considered high enough for having confidence to release the product after running the tests.
Slow feedback	Feedback about the changes made to the product came slow to the developers.

4.2 RQ1. Direct Signs

The case organization suffered from four direct signs of a dysfunctional CD practice, listed in Table 2: *failing builds, flaky tests, low test coverage* and *slow feedback.*

Failing Builds. The case organization suffered from failing builds. The existence of build failures is not necessarily a problem for the CD practice, but they become problematic if they are common and not fixed immediately. It was reported that builds were failing for long time periods and sometimes people had to fix builds that others had broken. It was suggested that an automatic reverting system could solve the problem; if a change broke the build, then that change could be automatically reverted to keep the build unbroken.

Flaky Tests. Some of the tests used in the CD pipeline were flaky and could fail randomly even if a software change did not have any problems in it. This makes it difficult to trust the build results and increases the build maintenance effort. The participants were aware that flaky tests should be fixed, but they did not have the resources required to make the tests more robust.

Low Test Coverage. Many participants felt that the test coverage was not high enough for being confident about that a change did not break anything in the product. This is

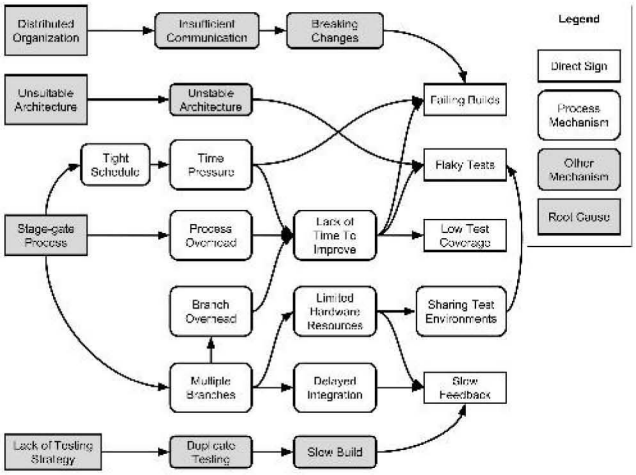

Figure 4: Overview of the direct signs, root causes and mechanisms in between.

problematic because when practicing CD, one should be confident to release the product after running the tests. Raising the test coverage higher was restricted by the push to develop the product features.

Slow Feedback. The developers received some of the feedback for their changes slowly. For example, the whole product was not integrated all the time; only the changes that would go into a release were integrated together. In addition, limited hardware resources limited the possibility to do manual testing, which delayed the feedback. Thus, the main benefit of CD, fast feedback, was not achieved in all cases.

4.3 RQ2. Mechanisms and Root Causes

The existence of the direct signs was explained with four root causes, as shown in Figure 4: *distributed organization, unsuitable architecture, stage-gate process* and *lack of testing strategy*. We describe the mechanisms of the stage-gate process in the next section and the other mechanisms in this section.

Distributed Organization. The case organization was distributed to several sites and countries. The participants thought that the communication was not working between the sites, especially when there were cultural differences. The communication problems were shown during the CD adoption by surprising breaking changes. These included API changes, version changes of libraries or changes in installation scripts. The breaking changes caused failing builds.

In order to fix the failing builds, developers needed to understand the changes made by other people. This was told to be time-consuming work. It is not known why the people who made the breaking changes did not fix the failing builds themselves. This could be again explained by the insufficient communication caused by the distribution.

Unsuitable Architecture. The participants reported that the product architecture did not support the adoption of CD. The architecture was said to be unstable and cause flaky tests. The instability came from the use of third-party software that was not robust enough.

Other than that, the architecture required the whole product to be released as a whole. It was mentioned that if pieces of the architecture could be released independently, the stage-gate process could be adjusted so that different sites could work more independently and perhaps cause less conflicts.

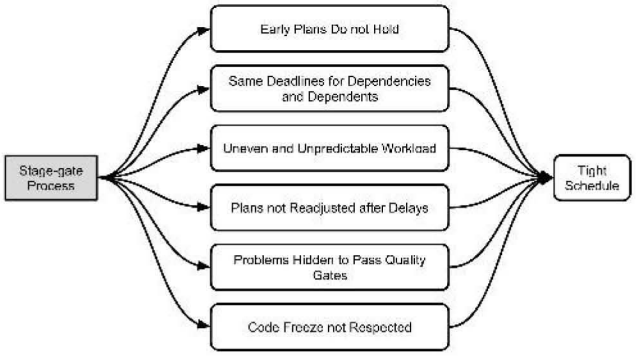

Figure 5: Different mechanisms why the stage-gate process caused the tight schedule in practice.

Finally, there were dependencies between subsystems in the architecture, and the subsystems were developed on different sites. The dependencies caused trouble, because the API's of the dependencies could change surprisingly.

Lack of Testing Strategy. Participants identified that there was no common view of how different types of tests should be used. They had noticed that some tests were redundant and some things were tested on a wrong level. The term unit, integration and acceptance tests were used to differentiate different kinds of tests. However, different sites used the terms differently. The lack of a testing strategy had caused duplicate testing in the CD pipeline and the pipeline was considered slow because of that.

4.4 RQ3. Stage-gate Process Mechanisms

The stage-gate process explains the direct signs through three primary mechanisms: *multiple branches, process overhead* and *tight schedule*, as illustrated in Figure 4. In particular the tight schedule mechanism proved to be caused by other minor mechanisms, shown in Figure 5.

Multiple Branches. The process required that after the code freeze, the product should be developed in a new branch. The branch needed its own testing environment and CD pipeline. This took some of the limited hardware resources away from other uses. In addition, working with multiple branches was described to cause complexity during development. Different people were working with different branches, which made communication difficult. Bug fixes and even new features sometimes had to be applied to multiple branches, taking additional effort.

In addition to the code freeze and development branches, the process required that a third branch was created for an upcoming release, even before development had started for that release. Also this branch was required to have its own CD pipeline and testing environment. This limited the available hardware resources even more.

Process Overhead. The process demanded that certain kind of tasks were performed during different stages and gates. For example, it was reported that large amounts of manual tests were performed before each release. Some participants thought that the manual tests were not always necessary considering the changes that had been made, but

they were nevertheless executed because it was mandated by the process. Other process overhead was caused by reporting and documentation work that was not considered necessary by the participants. For example, new documentation had to be created for some parts that had not changed during the development of a release.

Tight Schedule. The process caused a tight schedule for development, leaving no organizational slack [18] for improving the CD adoption. The tight schedule was itself caused by other minor mechanisms, shown in see Figure 5 and discussed next.

Early Plans Do not Hold. The content planned early in the stage-gate process was not fixed; instead, it kept changing during development. Thus, the estimates used during planning turned to be too optimistic, because more work was performed during the actual development. This caused time pressure, especially near the deadlines.

Same Deadlines for Dependencies and Dependents. Different teams were developing different subsystems of the product architecture. Participants described that another team developing a subsystem that others depend on might finish their work very near to a deadline. Thus, there would be little time for solving integration problems in the dependent subsystems. One suggestion was that the dependent subsystems would use a version of the subsystem they depend on that was developed one release earlier, so there would be more time to solve the integration problems.

Uneven and Unpredictable Workload. Because the plans kept changing during development, the workload near the deadlines was high, with people having to work weekends and feeling mental stress due to the time pressure. If a deadline could not be met, there was a possibility that the deadline was delayed for the whole product and developers would receive extra time to finish development. However, it was not known beforehand whether a deadline would be delayed or not and thus the developers could not predict how much time they would have for finishing the features. The conclusion was that sometimes developers were rushing their work and sometimes they had to wait for the possibility to release completed work.

> *Finally, we figure it out [that] we have to delay [the release] for two weeks and then we have lost time and this two weeks we could have used for some effective development. If we would have known that early enough.*
> — Workshop participant

Plans not Readjusted after Delays. Sometimes deadlines could not be met and thus they were delayed. However, this did not have an effect on the content in the future releases or deadlines. Thus, delays in early releases would cause more time pressure during the future releases. This was said to be a vicious cycle.

> *It causes problems if a fixed deadline is not reached. The next fixed deadline will suffer. [...] If one deadline is delayed, the next one is not. So we have less time there.*
> — Workshop participant

Problems Hidden to Pass Quality Gates. Sometimes product problems were hidden in order to meet the gate requirements in time. Some participants even reported that product managers would get bonuses for meeting the deadlines and therefore were distorting the real situation in reports. After the gate had been passed, the problems were fixed during later stages. However, this took time from the next release development and increased the time pressure.

> *There is a lot of this kind of work not done before release. There might come another release where fixes are delivered.*
> — Workshop participant

Code Freeze not Respected. In principle, after the customer trials had started, there should not have been any new feature development. However, in practice this was not the case. The hidden problems that had silently passed the gate requirements were being fixed during the customer trials. Also, it was reported that urgent customer requests were often done after the code freeze, because it was the fastest way to get them done and released. If they had to go through the normal planning process, it would take too long for the customers.

> *We don't support fast release cycles, so new features required by customers can only be done [after the code freeze]. [...] And it causes a vicious cycle that we are even more late with a newer release.*
> — Workshop participant

5. DISCUSSION

In this section we answer the research questions and discuss the limitations of our conclusions.

RQ1. What direct signs of a dysfunctional CD practice did the case organization have? The identified direct signs of a dysfunctional CD practice were *failing builds, flaky tests, low test coverage* and *slow feedback.* All of these are known in the literature and our results support their existence. Similar problems were found by Hukkanen [13], who studied the same case. The signs of low test coverage and slow feedback were not identified by Hukkanen, mainly because the focus of his study was on CI instead of CD, and branching issues were not discussed in his study.

RQ2. What caused the direct signs of a dysfunctional CD practice in the case organization? The identified root causes for the direct signs were the *distributed organization, unsuitable architecture, lack of testing strategy* and the *stage-gate process.* In our previous study at Ericsson [16], we identified and studied the root causes of distribution, unsuitable architecture and lack of testing strategy. Our case study supports the existence of the previously found root causes. The investigation of stage-gate process is a major contribution of this study, because it has not been studied elsewhere.

Distribution, unsuitable architecture and the lack of testing strategy played a significant role hindering the CD adoption in the case organization, too. It is difficult to quantify the proportional effects of different root causes, because they usually have relationships with each other in real-life organizations. For example, it seems that combining distribution of the organization with an unsuitable product architecture causes problems if there are architectural dependencies between the distributed sites.

RQ3. How does the stage-gate process used by the case organization explain the direct signs of a dysfunctional CD practice? The stage-gate process of the case organization explains the direct signs with the mechanisms of *tight schedule, process overhead* and *use of multiple branches.* These mechanisms caused lack of time to improve the CD practice, limited hardware resources and delayed integration. Especially the mechanisms leading to lack of time to improve the CD practice were most emphasized by the participants. All of the mechanisms have been previously known to hinder CD adoption, e.g. in our Ericsson study [16] and SLR [17], and this study supports their existence.

An additional contribution of this study is the explicit links from the stage-gate process to the direct signs.

Previous research has identified that integrating agile methods with traditional processes is possible and common [15, 26, 25]. This is verified also by our case study, because the case was using Scrum to organize their development work. However, generalizing this to CD does not seem to be valid based on our case study. One possible explanation is given by van Waardenburg and van Vliet [26] who emphasize the need for good communication between the traditional and agile processes. This was not achieved in the case because of the global distribution. Vuori and Huy [27] also had discovered communication problems at Nokia even when distribution was not an issue.

Our results show that adopting CD within a stage-gate process was not working as intended. This could be explained with the Stairway to Heaven model [12]. The model describes that while adopting agile methods require changes only in the R&D organization, further adoptions of CI and CD require changes in the other organizational functions, too. Thus, our contribution provides evidence for the Stairway to Heaven model as a hypothesis that adopting CD requires changes in the traditional processes external to the R&D organization.

Tight scheduling seemed to be a particular problem in the case organization because the process design did not properly take real-life constraints into account. Perhaps a more agile process with enough organizational slack [18] would have suited this product better, because the product was developed for an emerging market and the needed features were not known clearly enough beforehand. The product management who was managing the case organization was not included in this study and it could be fruitful to investigate the interface between product management and development in future studies.

It was briefly suggested that feature toggles could allow removing the multiple branches altogether. Literature supports this suggestion [7, 23]. However, implementing a feature toggle architecture is a challenge on its own. It might be difficult to adopt it considering the other problems the organization was facing.

Finally, the process demanded that branches were created at specific times, even if the developers did not think it necessary. Thus, stakeholders in the case organization who had knowledge of the consequences of the branches could not make the decision not to branch. Branching decisions were made outside the development unit. Perhaps giving more decision-making power to the developers would have prevented some of the adoption problems.

5.1 Limitations

Workshop participants were from a single site of the development unit only. Thus, the results represent only a partial view of the whole unit. Some of the results are directly related to the relationship between the development unit and product management. To gain a more holistic view of the situation, the whole organization should participate in the root cause analysis. However, in an organization this size, it can prove to be impossible to get specific people to meet on a certain time, especially when there might not be other incentive than participating in a research project.

Unfortunately the time reserved for corrective action innovation was shorter than expected and there were not many participants in the session. This inherently limited the output of the corrective action innovation. However, this does not reduce the validity of the conclusions, because most of the data was gathered during other steps in the research process. In addition, because feedback was collected only after corrective action innovation, there were participants in the causal analysis workshop from who we did not receive feedback.

The conclusions of the study can be analytically generalized to similar organizations. We do not attempt to generalize the results statistically. Instead, we claim that these mechanisms existed in this case study and thus can exist in other similar cases, too.

6. CONCLUSIONS

Adopting agile practices within a stage-gate process has been considered successful in the previous studies. However, CD adoption has not been studied previously in the context of a stage-gate process. Our case study results give indications that the stage-gate process can significantly hinder CD adoption and the adoption might not be possible without changes to the process.

The direct signs of a dysfunctional CD practice in the case organization were failing builds, flaky tests, low test coverage and slow feedback. These signs were caused by four major root causes hindering CD adoption: distributed organization, unsuitable architecture, stage-gate process and lack of testing strategy. Previous research supports the existence of other root causes except stage-gate process.

Stage-gate process hindered CD adoption by three mechanisms that limited the time available for the adoption: tight schedule, process overhead and the use of multiple branches for different process stages. Tight scheduling was partially caused by the process-unsuitability to the nature of the product. In addition, the process limited the available hardware resources and caused delayed integration.

We contribute to the previous research by the following ways. We identified previously undocumented mechanisms from stage-gate process to direct signs of a dysfunctional CD practice. We also constructed a hypothesis that it might not be possible to adopt CD in the context of a stage-gate process if the process is not changed to allow more organizational slack and decision-making power to the developers.

We provide the following implications for practice. Branching causes additional complexity for CD adoption and it should be avoided. Similar stage-gate processes might not be suitable for similar products which are made for emerging markets. Giving more organizational slack and decision-making power to developers might make the adoption possible.

We propose following future work. Stage-gate processes could be investigated in other organizations to see whether they have similar effects. Moreover, studying the interface between product management and development could be fruitful for understanding CD adoption.

7. ACKNOWLEDGMENTS

The authors would like to thank Nokia Networks and in particular the participants of the workshops in this study. This work was supported by TEKES as part of the Need for Speed research program of DIGILE (Finnish Strategic Center for Science, Technology and Innovation in the field

of ICT and digital business).

References

[1] ATLAS.ti GmbH. ATLAS.ti, 2016. URL http://atlasti.com/.

[2] L. Chen. Continuous Delivery: Huge Benefits, But Challenges Too. *Software, IEEE*, 32(2):50–54, 2015.

[3] G. G. Claps, R. B. Svensson, and A. Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57(0):21 – 31, 2015.

[4] R. G. Cooper. Stage-gate systems: a new tool for managing new products. *Business horizons*, 33(3):44–54, 1990.

[5] A. Debbiche, M. Dienér, and R. Berntsson Svensson. Challenges When Adopting Continuous Integration: A Case Study. In *Product-Focused Software Process Improvement*, volume 8892 of *Lecture Notes in Computer Science*, pages 17–32. Springer International Publishing, 2014.

[6] A. Eck, F. Uebernickel, and W. Brenner. Fit for Continuous Integration: How Organizations Assimilate an Agile Practice. In *Twentieth Americas Conference on Information Systems*, Savannah, Georgia, USA, 2014.

[7] D. Feitelson, E. Frachtenberg, and K. Beck. Development and Deployment at Facebook. *IEEE Internet Computing*, 2013.

[8] T. Fitz. Continuous Deployment, 2009. URL http://timothyfitz.com/2009/02/08/continuous-deployment/.

[9] M. Fowler. Continuous Integration, 2006. URL http://martinfowler.com/articles/continuousIntegration.html.

[10] M. Fowler. ContinuousDelivery, 2013. URL http://martinfowler.com/bliki/ContinuousDelivery.html.

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

[12] H. Holmström Olsson, H. Alahyari, and J. 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*, pages 392–399, Washington, DC, USA, 2012.

[13] L. Hukkanen. *Adopting Continuous Integration – A Case Study*. M.Sc. thesis, Aalto University, 2015.

[14] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, Upper Saddle River, NJ, 1 edition edition, 2010.

[15] D. Karlström and P. Runeson. Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering*, 11(2):203–225, 2006.

[16] E. Laukkanen, M. Paasivaara, and T. Arvonen. Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study. In *2015 Agile Conference*, pages 11–20, Washington, DC, USA, 2015.

[17] E. Laukkanen, J. Itkonen, and C. Lassenius. Problems, Causes and Solutions When Adopting Continuous Delivery - A Systematic Literature Review. *Submitted to Information and Software Technology*, 2016.

[18] M. B. Lawson. In praise of slack: Time is of the essence. *The Academy of Management Executive*, 15(3):125–135, 2001.

[19] T. O. A. Lehtinen, M. V. Mäntylä, and J. Vanhanen. Development and evaluation of a lightweight root cause analysis method (ARCA method)–field studies at four software companies. *Information and Software Technology*, 53(10):1045–1061, 2011.

[20] T. O. A. Lehtinen, M. V. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius. Perceived Causes of Software Project Failures–An Analysis of their Relationships. *Information and Software Technology*, 56(6):623–643, 2014.

[21] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö. The Highways and Country Roads to Continuous Deployment. *Software, IEEE*, 32(2):64–72, 2015.

[22] M. V. Mäntylä, B. Adams, F. Khomh, E. Engström, and K. Petersen. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5):1384–1425, 2015.

[23] S. Neely and S. Stolt. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *Proceedings of the 2013 Agile Conference*, AGILE '13, pages 121–128, Washington, DC, USA, 2013. IEEE Computer Society.

[24] A. A. U. Rahman, E. Helms, L. Williams, and C. Parnin. Synthesizing Continuous Deployment Practices Used in Software Development. In *Agile Conference (AGILE), 2015*, pages 1–10, 2015.

[25] G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold. Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. In *Product-Focused Software Process Improvement*, volume 9459, pages 149–166. Springer International Publishing, Cham, 2015.

[26] G. van Waardenburg and H. van Vliet. When agile meets the enterprise. *Information and Software Technology*, 55(12):2154–2171, 2013.

[27] T. O. Vuori and Q. N. Huy. Distributed Attention and Shared Emotions in the Innovation Process: How Nokia Lost the Smartphone Battle. *Administrative Science Quarterly*, pages 1–43, 2015.

[28] R. K. Yin. *Case study research: Design and methods*, volume 5. Sage publications, second edition, 1994.