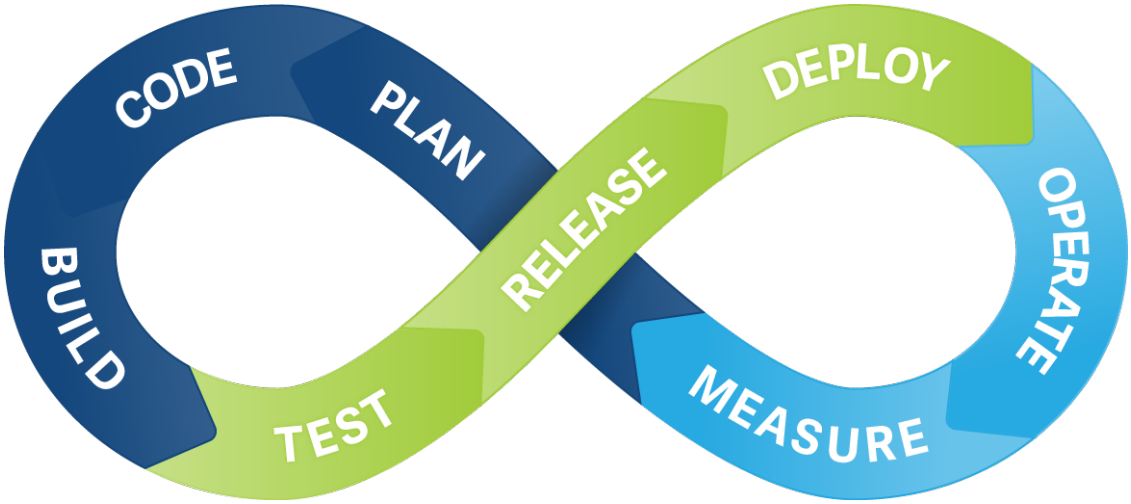
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Developer Confidence in Continuous Integration

Definition and Supporting Aspects

Master's thesis in Software Engineering and Technology

AYESHA ASLAM

ANNAPURNA ASHOK NAGANALLI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Developer Confidence in Continuous Integration

Definition and Supporting Aspects

AYESHA ASLAM
ANNAPURNA ASHOK NAGANALLI

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Developer Confidence in Continuous Integration
Definition and Supporting Aspects
AYESHA ASLAM
ANNAPURNA ASHOK NAGANALLI

Supervisor: Agneta Nilsson, Department of Computer Science and Engineering
Examiner: Eric Knauss, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Developer's Confidence in Continuous Integration
Definition and Supporting Aspects
AYESHA ASLAM
ANNAPURNA ASHOK NAGANALLI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context Organizations are facing a lot of challenges, both social and technical, in adopting continuous practices. In the midst of these challenges, little is known about how software developers are dealing with the usage of continuous integration as its immediate users.
Objective The objective of this study is to address the confidence related behaviour of developer in the context of continuous integration by proposing its definition and how it can be supported through features in CI tools and aspects in work environment.
Method A case study is conducted in a IT consultant company where developer's confidence is explored through interviews and workshop with developers having varied experience levels.
Result Developer's confidence depends on their knowledge and the quality of tests they add. Conditionally, it also depends on their confidence in CI tool and coworkers. Various features in CI tool and environmental aspects are identified that can potentially support developer's confidence in the context of continuous integration. These findings can help organizations in overcoming adoption challenges to continuous integration by supporting developer's confidence.
Conclusion To conclude, developer's confidence can be potentially supported to overcome adoption challenges in CI. Our findings for features in CI tools and environmental aspects can be implemented in organizations to validate their effectiveness and can serve as an inspiration to support developers for successful adoption of CI.

Keywords: Computer, science, computer science, engineering, project, thesis, continuous integration, CI tool, features, developer's confidence, supporting aspects.

Acknowledgements

Foremost, we would like to express our sincere gratitude to our supervisor Agneta Nilsson for her advice, guidance and immense knowledge. Besides her, we would also like to thank our company supervisors for helping us in conducting our research.

Ayesha Aslam & Annapurna A. Naganalli, Gothenburg, June, 2018

Contents

List of Figures

List of Tables

1
Introduction

'Integration is a "pay me now or pay me more later" kind of an activity' (Wells, 1991, p.1). If code is integrated in small bits during the project, there will be less struggle in the end to integrate the entire system built for months or even years. This could be achieved if developers integrate regularly into the mainline repository based on the decided integration frequency. Continuous integration (CI) is one such method which avoids fragmented development efforts especially when developers are working distantly from each other. All the developers have to be on the latest version, so that changes made to the code do not cause any integration headaches. Therefore, all the integration issues could be avoided mainly by adopting CI, which explains why this practice is needed in software development. CI life cycle begins when someone checks their revised code into the shared repository, then an automated tool or system picks up the change, checks out source code and runs the predefined tests or commands at different levels to verify that the change is good and the code is not broken. These CI tools are designed to be unbiased in checking the correctness of the changes, hence, *'preventing the "it works on my machine" syndrome'* (Meyer, 2014, p.14).

CI has a huge contribution in automated software engineering. This practice is widely adopted by many companies because of the need to deliver fast to customers. CI tools help in the automation of tasks such as the compilation, building, and testing of software. Incorporating additional functionality to these tools requires in-depth research in this area. Many tools and technologies, along with guidelines for adoption, have been developed to get the best out of continuous integration. However, according to Shahin et al. (2017), there are social and technical challenges in adopting continuous integration, delivery and deployment. They state that moving towards these continuous practices necessitates significant changes in the organization, such as changing team mindsets, organization's way of working and quality assurance activities among others. A general resistance to these changes is a hurdle in adopting continuous practices.

Another major challenge in fully adopting CI is a lack of expertise and skills in the developers which creates more pressure and high workload for them. This is further aggravated by a sense of responsibility in developers as they are now directly responsible for affecting their customers' experience (Shahin et al., 2017). Necessary organizational changes and developers' training require resources which in itself is a challenge for many organizations. Shahin et al. (2017) also states that lack of suitable tools and technologies is also a challenge in adopting CI and there is need

for improvement in these tools.

Laukkanen et al. (2017) mentions the behavioral characteristics of developers as a challenge to CI adoption such as experience, confidence and pressure. Since developers are the immediate CI users and practitioners, their actions directly affect the adoption of CI practices. To exemplify, when a developer commits a buggy code, it can lead to poor quality in the mainline and delays in feature delivery. These situations can be avoided by supporting developers while practicing CI. Among the various behavioural characteristics mentioned above, confidence is selected as the focus of our study based on the need of the case study company to explore developer's confidence in this context.

The literature provides a few statements but there is a lack of a universal definition and understanding about developer's confidence. The purpose of this study is to propose a definition of confidence in the context of CI, that will be derived from an in-depth analysis of the developers' perception of confidence combined with existing definitions from the literature. The main focus will be to explore developers' usage of CI tools and practices as well as how their confidence drives their behaviour when they commit code in a codebase due to the interest of the case study company to explore how developer's confidence can be supported both regarding aspects in the environment and features of CI tools. Consequently, this understanding will be used to propose, how confidence can be supported to achieve suitable behaviour of developers with respect to committing code in CI. Based on the identified research objectives, we have formulated the following research questions.

Research Question 1: *"How can we define 'confidence' of codecommit of developers in continuous integration?"*
Research Question 2: *"How can confidence be supported to achieve suitable developer behavior in continuous integration?"*

Pinto et al. (2017) state *'In spite of the increasing adoption, the large set of tools, and the well-known benefits, little is known about how software developers are dealing with the usage of continuous integration techniques'.* This relates to the behavioural characteristics of developers in the context of CI. Confidence is one of the important characteristics in driving their behaviour, but there is no accepted definition of confidence in this context. We aim to understand and analyze empirical data collected in our study along with existing definitions in the literature to propose a definition of this phenomenon. We assume that confidence of developers is influenced by their environment. Therefore, we choose a case study approach, which will allow us to explore both the environment as well as features of CI tools as potential aspects to be used to support developer's confidence.

The outcome of the study is a proposed definition of confidence emphasizing the factors that affect developer's confidence when practicing CI, and identified potential features in CI tools and environmental practices that can support developer's con-

fidence. This can help in improving CI processes by addressing confidence related problems among developers.

The structure of the report is as follows. Chapter 2 presents the literature review that contains background knowledge related to research questions. The design of the case study is then presented in the Research approach chapter. Following the research approach section, results are presented in Chapter 4 and discussed in Chapter 5. Chapter 6 concludes the report along with implications and explanation of validity threats.

2
Background

In order to keep pace with fast growing markets, companies need to deliver fast and release often to customers. Continuous delivery and deployment are automated ways of delivering value to the customers at a faster pace, whereas, continuous integration is about integrating work into the codebase multiple times a day (Shahin et al., 2017) by performing tests to avoid code breakages. Continuous deployment is only possible through continuous integration, hence, it puts continuous integration in a vital place for the entire chain to work efficiently. Some of the common practices in continuous integration are maintaining a mainline repository, automating the build, making it self-testing, committing to mainline frequently, keeping an up-to-date version of code on an integration machine, keeping the builds fast, fixing broken builds immediately and making builds and its results visible (Fowler, 2006). Carrying out these practices ensures predictable project progress, early detection of bugs and timely maintenance.

Despite the theoretical wealth of knowledge about the practices and advantages of CI, the implementation of these practices in industry is quite different and there are many adoption challenges faced by industry (Shahin et al., 2017) that refrains it from reaping the benefits of CI. Several studies (Shahin et al., 2017; Leppanen et al., 2015; Pinto et al., 2017; Mårtensson et al., 2017) have shed light on the social and technical challenges and reasons for failure in fully adopting CI in companies.

Among these challenges, developer's confidence in relation to continuous practices is mentioned as a reason for inefficient CI. One study mentions overconfidence as a reason for frequent build breakages (Pinto et al., 2017). Another study claims that immediate deployment to customers lowers developers' confidence and refrains them from fully adopting CI (Leppanen et al., 2015). There is also a false sense of confidence in software quality created by CI which causes the developers to blindly trust build tests (Pinto et al., 2017). Although the literature has various viewpoints on confidence of developers, there is no universally shared definition of the concept of confidence of developers in relation to CI.

We have taken a point of departure from definitions in the literature and used these as a basis for further exploration of the confidence of developers in the context of CI. According to Merriam-Webster dictionary, confidence is defined as *'the feeling of being certain that something will happen or that something is true'*. In the field of human behavior, Navajas et al. (2017, p.1) has defined confidence as *'the "feeling of knowing" that accompanies decision making'*. They further state that *'confidence*

is a function solely of the perceived probability of being correct'. It depicts the individual's faith in phenomenon.

Webster (1913) has defined confidence as a concept that comprises of trust, reliance and belief. Therefore, trust and confidence are strongly linked together. Both these concepts are used in decision making process, however, there are some differences that distinguish trust from confidence. In a Canadian governmental report aiming to distinguish trust from confidence, Adams (2005) describes trust as an issue only in decisions that involve risk and vulnerability. *'It is about taking a "leap of faith" from what is known to what is unknown'* (Adams, 2005, p.1). Whereas, confidence is involved *'in discrete reason-based judgments related to probability of specific event occurring that often occurs in situation without risks'* (Adams, 2005, p.3). Additionally, the scope and information used in decision making process for trust and confidence are significantly different. In confidence based judgments, only immediately available information is used to determine the probability of occurrence of an event, whereas, in trust-based judgments, the available information is extrapolated to consider both the past and present behaviour as well as other life experiences to make a judgment (Adams, 2005). In all these definitions, confidence is related to probability or certainty of being correct in decision-making. This *"feeling of knowing"* drives the human behaviour in taking different actions. The higher the confidence, the higher is the possibility of an individual taking an action.

The software engineering literature on continuous practices includes various pointers to the confidence of developers in different contexts (See Table 2.1). We were able to find eight different definitions of confidence that fall into the context of continuous integration, delivery and deployment. One of the definitions in the context of CI describes confidence of developers in being able to reach the integration frequency expected by their managers (e.g. once a day or every four hour). Whereas, two of the definitions in the same context describe confidence of developers in their code at the time of commit and resulting software state (broken or successful build) and quality (defect-free). Four of these definitions of confidence are focused on describing it as the confidence of teams or organizations in the quality of software at the time of its release. Finally, one of these definitions is focused on the confidence of project sponsor in software quality. These different definitions of confidence are put together in the table below:

Table 2.1: Definitions of confidence in the context of continuous practices taken from the literature on Software Engineering

No.	Reference	Subject	Context	Definition
1	Leppanen et al. (2015)	Developer	CI	**Summarized as:** Confidence of a developer in software quality and the defect-freeness of code.

2	Pinto et al. (2017)	Developer	CI	**Summarized as:** Confidence of CI users(developers) in code state and quality resulting from a commit.
3	Laukkanen et al. (2017)	Developer	CI	**Summarized as:** Confidence of a developer in being able to reach the integration frequency expected by managers.
4	Bugayenko (2009)	Project sponsor	CD/CDep	**Summarized as:** Confidence of project sponsor in software quality.
5	Schermann et al. (2016, p.2)	Team / organization	CD/CDep	**Quoted as:** *'amount of confidence gained on the three quality gates automated testing, manual testing, and code reviews'.*
6	Port and Wilf (2013)	Team / organization	CD/CDep	**Summarized as:** Confidence in software quality at the time of its release.
7	Ståhl and Bosch (2017, p.7)	Team / organization	CD/CDep	**Quoted as:** *'confidence that software is safe to release'.*
8	Chen (2017, p.2)	Team / organization	CD/CDep	**Quoted as:** *'Confidence in release reliability'.*

While exploring the state of the art in CI tools, we were able to find through practitioners' viewpoint and a thesis report (Kapelonis and Engineer, 2016; Polkhovskiy, 2016) some basic requirements a tool should have to support CI. Although, we could not find any scientific publications supporting these requirements, a wide amount of websites describing CI tools seem to support a set of basic requirements in the CI tools. These set of requirements were used to assess the state-of-the-art in the CI tools used within the case study company and are described as follows:

Connectivity Connectivity is very important for any team to use a particular CI tool. It describes how easy it is for end users to deliver code into the service. It's a very basic but quite important requirement in CI tools.

Initial Setup Setting up a sample project can be easy, but setting up one's own project with the build server can be very time consuming. In any case, it is good to have comprehensive guidelines for the setup process.

User Interface It can be difficult to define what could be stated as a good or bad interface. But, as a user, one should at least be able to view their build, browse through it and get some detailed information when required.

Build environment It includes the languages supported by the product, versions of compilers, build systems and the easy update or installation of the software. A build environment with extensive configuration properties would ease the job of users.

Feedback With a CI tool, anything could go wrong like unresponsive services, incorrect setup environment and failed unit tests. So, many teams would like to monitor the build in real time and logging extensively which would help them understand the errors.

Post build steps It's quite important to know what happens next after the builds are tested. If one needs to update the status, send an email, move it to the next stage or update a bug issue.

Documentation A good documentation is something which matters a lot for the users. A guide well written would help users to setup, understand and use the CI tool.

Support / Ease of Contact Support matters a lot when the team at hand has some issues and is stuck with the development work costing effort and time to the team. Especially, when the build leads to release which is quite important, support would be needed the most.

3
Research approach

3.1 Method

To conduct our research, we have selected case study (Runeson and Höst, 2008) as a methodology to study the case in its real-life context. Developers' confidence is a complex phenomena which requires a deep understanding and cannot be studied in isolation. Therefore, case study is a suitable research methodology in this case as Runeson and Höst (2008, p.2) state:

> "The case study methodology is well suited for many kinds of software engineering research, as the objects of study are contemporary phenomena, which are hard to study in isolation. Case studies ... they provide deeper understanding of the phenomena under study."

As a pre-study to our research, we started out by exploring the literature to find the existing definitions of confidence in the context of continuous practices. These definitions served to formulate initial definition of confidence which was used as a starting point during data collection. We also investigated the state-of-the-art in six commercial CI tools, which helped us to identify the basic requirements on a CI tool to meet the organizational needs. Additionally, we evaluated two CI tools used within our case study company to familiarize ourselves with their current CI workflow. The insights from this pre-study guided the design of our case study.

3.2 The case study company

The case study is conducted in a company called Combitech, which is an independent technical consulting company and is a part of Saab AB. It provides consultancy services to other companies in automotive, telecommunication and other business sectors. It has almost 1900 employees in 30 locations within the Nordic region, out of which around 300 are working at Goteborg which was our site of case study. The company is using extensive testing and integration procedures in its software development projects. It has a comprehensive CI machinery that involves both the automated and manual testing in different environments. It has an in-house CI tool for continuously integrating code into the pipeline and is currently in use by their developer teams. They are also developing a prototype of an open-source tool called *"CodeFarm"*. The unit of analysis in this case study is the developers in this company who are using these CI tools.

3.3 Data collection

In order to collect data, first degree technique of interviews (Runeson and Höst, 2008) was used to directly collect data from developers in real-time. These interviews were semi-structured with identified areas to cover in the data gathering. The arrangement of the interviews followed a pyramid approach (Runeson and Höst, 2008) where the initial questions were closed to focus on specific areas of interest, whereas the questions towards the end of interview were open questions to welcome further ideas from developers and gain a deeper insight into the subject of interest. Considering the research objectives of our study three areas of interests were identified which are confidence, features and work environment practices that support developer confidence.

Initially, some questions were asked to gain background information about developers. These questions focused on developer's experience and current role. Another group of questions was asked to help the developer reflect on the efficiency of current CI practices and tools. There was also a set of questions related to developer's experience with CI tools, their current workflow in tools and related trainings under the assumption that training in CI tools may support confidence of developers who use these tools.

The eight definitions found in literature were used to propose a definition of confidence which was then used to help developers reflect on their confidence in the context of CI. In order to further help the interviewee in the reflection process, they were asked to discuss the factors that affect their confidence and their commit criteria for code validation before codecommit. We asked questions related to the code validation activities and effects of build breakages, due to the assumptions that detailed code validation before commit could support developer's confidence and build breakages could lower it. Developers were then asked to reflect on their performance with the tools in order to gain insight into how they have improved/changed over time. We also asked questions related to the co-workers and upper management based on the assumption that support from such roles could improve developer's confidence. Towards the end of the interview, developers were asked to reflect openly about existing or proposed features in the tool and environmental practices that support their confidence.

Eyolfson et al. (2011, p.6) state that: *'the more experienced the developers are, the less likely that their commits are buggy'*. Therefore, the selection of subjects for interviews was made to capture the perspective of developers from different experience levels. We only provided the selection criteria and the case company selected the developers based on their years of experience in software development and availability. Since all the interviewees were developers, we have interchangeably used the terms *'developers'* and *'interviewees'* in subsequent sections of the report. Additionally, in order to conceal the gender of developers, we have used third person pronouns i.e. *'they'* or *'their'* throughout the report. The following table presents the division of 10 developers that we interviewed and the distribution in the different

groups of experience.

Table 3.1: Distribution of developers across different experience groups

Experience group (Years of experience)	**No. of developers**
Junior (< 2)	3
Medium (2 to 8)	4
High (> 8)	3

The interview sessions were recorded with the permission of developers and we assured confidentiality of the data. These recordings served as a backup and were repeatedly referred to during the analysis. Both the authors participated in the interviews and switched roles conducting the interview and taking notes. Each interview started with a brief introduction of the thesis and its objectives, had the developer not known it before. After that, interviewee was asked questions about their background, followed by the questions that captured data in the formulated areas of interest. Where needed, the answers were summarized by the interviewer to the interviewee in order to validate their answers and avoid misunderstandings. Interviewees were also guided with examples and ideas in order to help them to reflect on the questions. There were also some occasions where questions had to be rephrased or asked in a different way in order to prevent the interviewee from straying away from the topic under discussion. At the end of interview, the subjects were asked for any further comments or input they had on the topic. Additionally, they were also informed of where their data will be used and how the results would be presented.

As a further activity in our study, we arranged a workshop. This workshop was mainly planned to validate the tentative analysis and further discuss ideas on how to support developers' confidence. It lasted for two hours and we had 18 participants, which included 9 developers who were interviewed before and 9 others, including two managers, who were not interviewed before. Participants were distributed into four groups. Guidelines for the workshop contained instructions regarding an individual exercise, group discussions and the methods that were used in the group discussions. Participants were asked to read through the guidelines and complete the individual exercise prior to workshop, but they were also given ten minutes during the workshop to complete the individual exercise. The workshop started with a presentation of the tentative analysis from the study. The individual exercise was intended to validate the proposed definition of confidence and the data collected from interviews, covering existing and proposed, features and environmental practices supporting developers' confidence. Further, discussions within each group were lead by a preassigned discussion leader and ideas from each group were presented at the end of the workshop.

For group discussions we used the cheatstorming method (Faste et al., 2013) , where the participants were asked to ideate by using pre-existing ideas generated in the interviews. Each group internally discussed the data presented from the interviews to formulate new interesting ideas, provide suggestions to existing ideas and analyze

the limitations and potential these ideas have in supporting developer's confidence. In order to prioritize the ideas, we asked all the participants to vote for top three ideas within each group using post-it voting (Dam and Siang, 2017). All the ideas generated in group discussions were written on post-it notes, which were used in voting. It was important to understand which three ideas among the discussed ones weigh high. Voting was done in each group using the five dollars method (Schenk, n.d.), where every member in the group had five dollars which they assigned to the ideas they preferred the most. Based on which ideas had the highest dollars assigned, top three ideas were selected from each group.

3.4 Data analysis

Data was analyzed in parallel to the interviews in order to identify emerging patterns and themes which were further used to guide the reflection process of interviewees in later interviews. Data triangulation (Carter et al., 2014) was done, where data collected from interviews was analyzed by both the authors in parallel through thematic analysis (Alhojailan, 2012), where different quotes from interviews were assigned to suitable codes, and codes with repetitive patterns were grouped under an emerging theme. To exemplify, one interviewee stated *"Can I trust the tool?"*, this quote was identified as *"Trust in Tool"* code. While another interviewee stated *"Are the tests lacking coverage"*, which was identified as *"Test Coverage"* code. Both these codes had an emerging pattern which emphasized on developers confidence in the CI tool, thus the theme representing these codes was identified as *"Confidence in tool"*.

4

Results

In this chapter, we present our results on how the definition of developer's confidence evolved throughout the case study. We then describe our findings on factors affecting developer's confidence and different aspects for supporting it.

4.1 Confidence

This section presents our findings with respect to the evolution of definition of confidence and the factors affecting developer's confidence in terms of themes identified through thematic analysis.

4.1.1 Evolution of definition

All the definitions from the literature study summarized in Table 2.1 , are put together in the figure below based on their context related to continuous practices of integration, delivery and deployment.

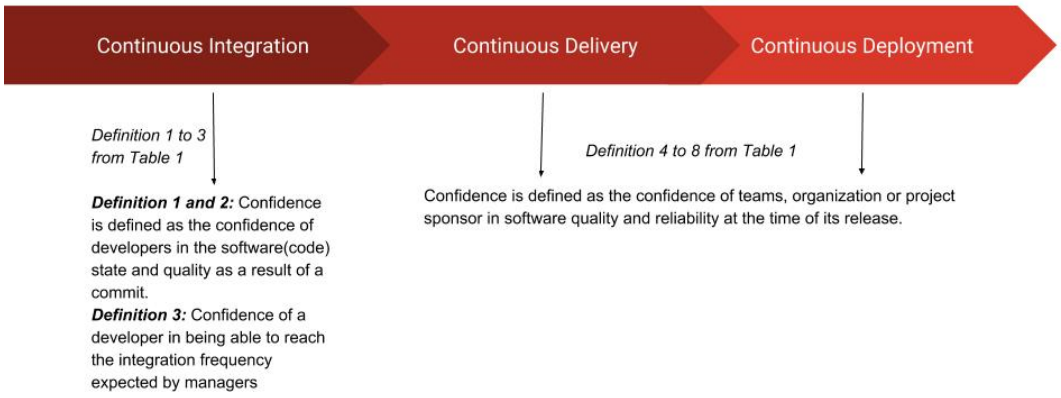


Figure 4.1: Distribution of confidence definitions in Table 2.1 based on the context in continuous practices

As a starting point, we have selected the definition 1 and 2 by Leppanen et al. (2015) and Pinto et al. (2017), respectively, in the context of CI from Table 2.1. While discussing developers' trust and confidence as obstacles to adopting continuous deployment, Leppanen et al. (2015, p.6) state that:

> 'Code going straight to production must be as defect-free as possible ... developers' reputations are on the line: deploying a broken build

> to customers could strain the relationship between parties and create an unwanted user experience ... Any lack of confidence in an application's quality is amplified by the knowledge that any and all changes are immediately deployed.'

Additionally, Pinto et al. (2017, p.1) state that *'CI usage increases the confidence that the code is in a known State'*. Based on our understanding of Leppanen et al. (2015) viewpoint, developers' concern for their reputation affects their confidence. Since they relate developers reputation to the broken build , we have selected build state as a factor in affecting developer's confidence. Considering the build can be evaluated to various states, we have added an ordinal scale from broken to successful build for elaboration. Pinto et al. (2017) discusses that having a knowledge about the code state improves developer's confidence. Whereas according to Leppanen et al. (2015), code quality which they also referred as application quality effects developer's confidence. They also describes code that is defect free as a qualifier for deployment, so we consider defect-free code as a high quality code. Based on the the correlation between developer's confidence and code quality, we have added code quality factor to our definition. To summarize, our literature study pertaining to defining developer's confidence led to the following definition of confidence as a first proposal.

First proposal: *"Confidence of a developer at the time of codecommit with their code changes can be defined as how they perceive the result of their commit based on the expected code quality and build state factor. In this context high code quality will be a defect free code and build state can be evaluated on a ordinal scale from broken to successful build"*

This definition was introduced to the first three interviewees. Discussions revealed that, the interviewees were not able to understand the definition from just reading it. They found the definition vague and needed additional explanation. According to them, build state and code quality were ambiguous factors. The interviewees could not understand the relation between high code quality and defect-free code and how a high quality code can support their confidence. According to them, code quality depends on the test coverage of the code which supports their confidence. Additionally, interviewees mentioned that their confidence depends more on their knowledge of current task and the tests they add for their code. These factors helped them in achieving high code quality. Therefore, code quality factor was replaced with knowledge and test coverage. Also, interviewees could not relate their confidence to the build state factor. We then explained that mainline is referred to as build and build state corresponds to the results of the build jobs running on mainline. After clarifications, they described that their confidence is affected by the stability of the mainline rather than the results of the build jobs. So, after these three interviews, the definition was re-formulated into the following:

> **Re-formulated version:** *"Confidence of a developer in their code at the time of committing it into the mainline can be defined as an outcome of their knowledge about current task, test coverage and their conjecture of stability of mainline after commit."*

This definition was used as an input for the subsequent interviews and interviewees were asked to comment on and rate the definition on a scale of 1 to 5 in terms of understandability and agreeability, with 1 being poor and 5 being excellent. Ratings and comments showed that interviewees found the definition understandable and agreeable. This definition was used for all of the remaining interviews and no further iteration was needed. In order to validate the definition further, it was presented in the workshop and participants were asked to comment on and rate the definition. Data from both the interviews and the workshop highlighted some interesting factors that also contribute to developers' confidence such as the experience with the system they're working with, code reviews from their co-workers, knowledge of scope and impact of the code change and the ability to trace their commit in the CI tool.

Data analysis highlighted three aspects of knowledge that support developer's confidence. These include knowledge about current task, already included in the previous definition, knowledge about the whole system and knowledge about scope or impact of code change. Therefore, in the final definition, task knowledge factor was replaced with overall knowledge that encompasses all the three aspects mentioned above.

Data collected from workshop provided some interesting insight into the factor of test coverage in affecting developers' self-confidence. Participants stated that test coverage is beneficial in a sense that it enforces developers to add sufficient tests for their code before commit but it only partially supports their self-confidence because 100% test coverage is meaningless if test quality is low. They added that in agile way of working, developers need to deliver fast. They add tests to meet the test coverage criteria before commit but due to the stress on fast delivery, no one has enough time to improve the existing tests to avoid duplication and overlapping tests. As a results, CI tool is burdened with large number of tests added to meet test coverage criteria, but the quality of tests decreases. Therefore, participants stated that they cannot rely on test coverage criteria alone to support their self-confidence. Additionally, they rely on their own efforts to add quality tests for their code before committing to meet the test coverage criteria. Adding quality tests for their code makes them more confident in their commit as there is a high chance that it will pass. However, when these tests are combined with other tests written by other developers, then it lowers their self-confidence because of not knowing the aspects such as sufficiency or degree of overlap of these tests. So, test coverage factor in the previous definition was replaced with test quality factor because test quality supports developers' self-confidence whereas test coverage only enforces adding sufficient tests to meet coverage criteria. But, the test coverage metric of tool improves developers' confidence in CI tool as it shows that this metric forces developers to

add sufficient tests for their commits, even though the quality of these test might be uncertain. To add to their confidence in CI tool, developer stated that tool stability and visualization of their commits and code base improves their confidence in the CI tool at the time of commit. If they have high confidence in CI tool at the time of commit, they find the results of their commit to be more reliable. But, developers stated that *"environment can never be stable"* while talking about tool stability. So, they cannot make the tool to be 100% stable. Hence developers cannot completely rely on the CI tool for their confidence.

Similarly, developers mentioned that their confidence also depends on their trust in co-workers and code reviews from them, but they do not rely completely on it. During the study, use of code reviews was found to be arguable in supporting developer's confidence. Junior developers considered code reviews from knowledgeable persons helpful in supporting their confidence, whereas, experienced developers preferred testing their code themselves over code reviews. They added that their trust in their co-workers depends on how competent they are for executing their assigned task. Since developers had varied views on their confidence in co-workers, it is a conditional supporter of their overall confidence at the time of commit. After analyzing the data from workshop, following definition was finalized:

> **Proposed definition:** *"Confidence of a developer in their code at the time of committing it into the mainline can be defined as an outcome of their knowledge(of task, system and scope or impact of code change) and test quality. Conditionally, it also depends on the confidence of developers in the CI tool and their co-workers."*

Following diagram shows the evolution of definition through interviews and workshop along with the ratings given by interviewees. It can be seen that the average rating of the definition improved gradually throughout the study.

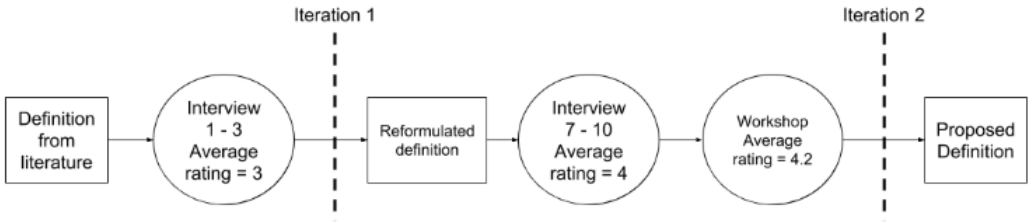


Figure 4.2: Evolution of definition of developer's confidence during case study

The thematic analysis generated three themes that affect developer's confidence, namely self-confidence, confidence in co-workers and confidence in the CI tool as discussed above. The diagram below shows the emergence of themes from quotations in the data through thematic analysis:

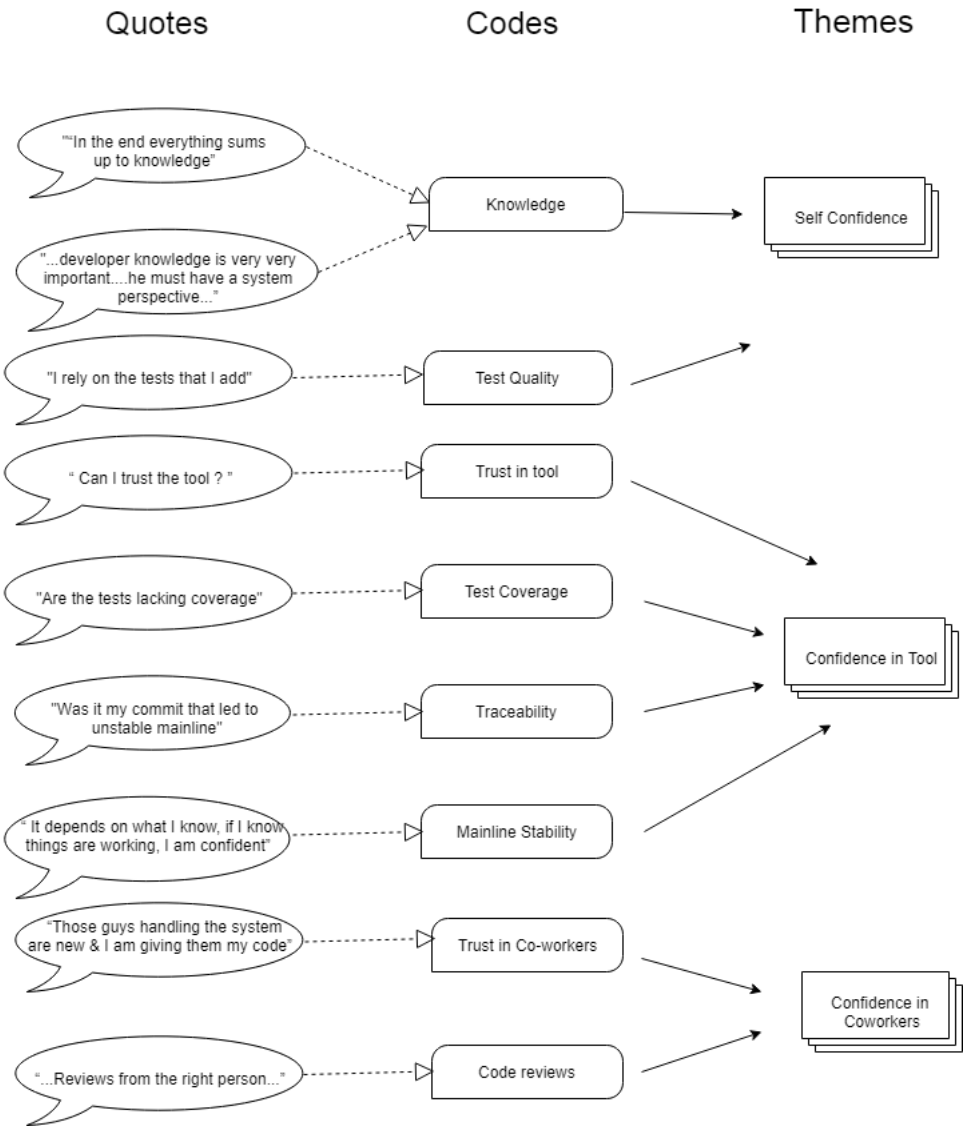


Figure 4.3: Examples of thematic analysis

4.1.2 Self confidence and confidence in co-workers

Interviewees stated that they aim for a high confidence level at the time of committing their code into the mainline. To achieve this confidence level, they write new test cases to cover their code and make sure that these test cases pass in addition to the existing ones. Additionally, they also mentioned that their knowledge about the current task also affects their confidence. So, they gather this knowledge from their co-workers, if they do not have it already, to gain higher confidence in their code. However, their knowledge about the system they are working with and about the scope and impact of the change they have made also adds to their confidence. They also related this knowledge to their experience with the system and the project they are working on. Therefore, the more experienced they are, the more confidence they have in their work.

Interviewees added that they also get code reviews from co-workers to gain a higher confidence level in their code. These reviews help them in increasing their knowledge and identifying potential problems in their code. They can also select reviewers which helps them to get expert opinion. Some interviewees mentioned that they prefer a face-to-face review in comparison to the reviews through the tool because it gives them an opportunity to explain their code and approach to problem solving. On the contrary, some interviewees also mentioned that they rely more on the tests they write rather than the code reviews.

Developers' confidence also depends on their perception of how good their co-workers are at their jobs, as one of the developers stated *"Those guys handling the system are new and I am giving them my code"*. Trust in co-workers was further highlighted while discussing code reviews where they mentioned that getting a review from a trustworthy person enhances their confidence.

4.1.3 Confidence in CI tool

Interviewees stated that their confidence is also affected by the trust in the CI tool that they are working with. The ability of a CI tool to catch defects in the code enhances their trust in it. Interviewees stated that if they have high trust in the CI tool that it will catch the bugs that were not detected during testing before commit, they have a high confidence. This trust is built over time through continuous committing and resulting failures, which indicates that the CI tool is doing its job. According to some developers, they find it suspicious if the mainline never breaks as it seems that the CI tool is not good at catching bugs and they cannot trust it. During the workshop, interviewees also stated that unstable environment also lowers their trust in the tools. For example, in case of a failure, they are unsure if it is the environment issue or a bug in the code that is causing a failure. Persistent instability of environment causes them to lose their trust in the tool.

Additionally, the traceability of developers' commit once it is inside the CI tool improves their confidence. After initiating the commit, they are able to follow its status and monitor it at different stages in the CI workflow. The transparency of the CI tool helps them to see what happens when their commit enters the tool and it improves their confidence.

Furthermore, interviewees stated that efficiency of CI tool is largely dependent on the test coverage that it provides. If the CI tool has a good test coverage of the code base, their confidence is high. However, some interviewees also stated that their confidence is also affected by the stability of test cases. Despite having a good test coverage if the test cases are unstable, they have a low confidence in the CI tool.

Interviewees also mentioned that the CI tool helps them in visualizing the status of the mainline. If they observe that the mainline is unstable at the time of committing their code, they have low confidence as they fear that their commit may aggravate the situation. On the other hand, if the mainline is stable at the time of

commit, their confidence is comparatively higher as they have already tested their code before commit and the chances are high that it will successfully merge into the stable mainline.

These themes reflect that confidence of a developer depends on the factors relating to their own self confidence, the confidence they have in the CI tool and the confidence they have in their co-workers, who are responsible for testing their code or reviewing it. However, it would be interesting to note that developers can put in the effort to improve their self confidence by testing their code well and gathering sufficient knowledge. On the contrary, improving their confidence in co-workers and tools is not in their power as people can not be controlled and CI tools are complex to handle.

4.2 Support

In this section, we present our findings regarding identified themes through thematic analysis for supporting developer's confidence.

4.2.1 Training in CI tool

Prior to the interviews, when the interview questions were designed, we assumed that some initial training with the CI tool would help the developers support their confidence. To investigate this assumption, we included some interview questions with trainings in focus. Most of the developers mentioned that they had attended presentation or received guidelines regarding the CI tool and some of them thought it was sufficient. While, few developers mentioned that they would prefer learning by doing, according to them they get a hold of the tool when they start working with the tool. Also, some developers suggested that having some traditional classroom trainings with hands on exercises with tools would help them support their confidence.

4.2.2 Commit criteria

It was assumed that all the developers have some criteria or checklist of their own, which they use to validate and gain confidence on their code before commit. With this assumption in mind, interview questions were framed to ask developers about their commit criteria. Some developers stated that they validate their code by running the legacy tests locally which are later run in the CI machinery. Further, developers do manual or automated testing on their end and some prefer to have sufficient test coverage before they commit their code. Code review was one of the popular criterion through which many developers get their code reviewed before they commit. It helps them to validate the code and get a second opinion. On the other hand, some developers mentioned that code reviews are not that important, as they are unsure on how experienced their reviewers are and how much knowledge they have regarding the task at hand. In either case, developers emphasized the

need to have sufficient knowledge of their tasks whether it's complemented by code reviews or not.

4.2.3 Features in CI tool

Interview data was thoroughly examined to identify existing and proposed features of CI tool stated by interviewees in supporting their confidence. These feature suggestions were complemented by the reflections from workshop where the participants discussed the potential, limitations and improvements of these features. Table 4.1 and 4.2 list the suggested features that already exist in their CI tool and the proposed ones, along with the data from the workshop.

Table 4.1: Existing features of the CI tool used by developers that support their confidence.

Existing Features	**Agree / Disagree**
Follow my commit: Helps the developers to track their commit in the CI machinery.	Agree = 17 Disagree = 1
Commit Gate: Performs the basic level block tests on the codecommit and does not allow the faulty code to pass beyond it in the CI machinery.	Agree = 15 Disagree = 3
Code reviews: Developers can get reviews on their code before they commit.	Agree = 14 Disagree = 2 No answer = 2
Functional Verification: Test the code at system level and verify if it meets the system specification or requirements.	Agree = 15 Disagree = 1 No answer = 2
Adding reviewers to your code: Developers can select and add a reviewer for their commit.	Agree = 16 Disagree = 1 No answer = 1
Test pass/fail indication: Developers can see the status of each test case as the particular test suite is executed.	Agree = 15 Disagree = 1 No answer = 2
Information on who developed a particular test: Developers can see who is working or has worked on a particular test case.	Agree = 11 Disagree = 5 No answer = 2
Bisect Script: At higher levels in CI machinery, multiple commits are batched together to run different test suites on them. If a test suite fails, it becomes difficult to find the faulty commit among the batched commits. In this situation, bisect script takes one commit at a time to run a test suite on it and find if it is at fault.	Agree = 15 Disagree = 1 No answer = 2
High node level testing: Node level testing enables developers to select a dedicated node that tests their commit before pushing it to the commit gate.	Agree = 16 Disagree = 1 No answer =1

Table 4.2: Proposed features for the CI tool used by developers that can support their confidence.

Proposed Features	**Agree / Disagree**
Information on who made modifications to code: A feature to see who frequently commits to the particular part of the code.	Agree = 11 Disagree = 7 No answer = 0
Bisect script: *"Bisect script"*(as above) exists only in specific CI flows but was proposed to be implemented in other CI flows as well.	Agree = 16 Disagree = 1 No answer = 1
Ability to run different tests in different environments and setups: Test code on different environment setups in commit gate.	Agree = 8 Disagree = 7 No answer =3
Brief logs: Having short useful descriptions in the logs.	Agree = 9 Disagree = 5 No answer = 4
Code Coverage: Indication of how much of the application code of their codecommit was actually tested by the CI tool.	Agree = 14 Disagree = 2 No answer = 2
Test Coverage: If tests are sufficient or some additional tests need to be added to validate the code beforehand.	Agree = 13 Disagree = 4 No answer = 1
History of stability of main branch: The ability to know the history of mainline stability describing how has it been for the past few days and its current status.	Agree = 15 Disagree = 2 No answer = 1
Appreciation for successful merges: Recognition or appreciation, for having a history of successfully committing to mainline or not introducing faults.	Agree = 7 Disagree = 7 No answer = 4
Finding related Commits: Ability to find the related commits that might have caused a commit to fail would help to analyze the problem faster.	Agree = 16 Disagree = 1 No answer =1
Adding flags to reviewed commits: Indicating if a particular commit has been reviewed or not, as unreviewed commits are more vulnerable to cause faults when the CI flow breaks.	Agree = 8 Disagree = 10
Guidelines for adding commit descriptions: Guidelines for adding commit description with relevant information might allow for automatic detection of a faulty commit in a batch of commits.	Agree = 14 Disagree = 4
Highlighting problematic code areas: An indication of problematic area of the code base, would help the developer to decide if they should commit or wait.	Agree = 13 Disagree = 5
History of developer's commits: Developers would like to know which commits they have made in the past alongwith the details.	Agree = 11 Disagree = 6 No answer =1

Update support information: Keep support documentation up-to-date to find the required information and contact the right person.	Agree = 13 Disagree = 3 No answer =2

Some of the existing features were found to be particularly popular among developers in supporting their confidence. *"Follow my commit"* is one of these as evident from the data in Table 4.1. It was mentioned as quite useful in providing traceability of developer's commit in the CI tool and prevented the tool from behaving like a black box. Developers also added that it eased the visualization of enormous amount of information present in the tool which would otherwise be difficult to extract from it. Features related to code reviews and various testing levels were also found to be helpful to developers as these features enabled them to achieve a higher confidence level on their code.

Discussions on proposed features revealed both the positive and negative impacts of implementing some of these features from Table 4.2, whereas some proposed features were found to be really helpful in supporting developer's confidence. Highlighting problematic areas in the code base was considered useful as it would allow developers to have a bigger picture of code base. As a result, developers will be able to analyze the relevancy of their commit to these areas and will take necessary steps to avoid aggravating the problems. Another feature that was discussed a lot both in terms of its potential and limitations was the *"Bisect script"*. Potentially, it was considered a good feature to complement current practices for finding faulty commits in a batch of commits. By providing valuable information, it can help dedicated CI monitor team in quicker troubleshooting. However, this script was also referred as a "dream" as it had some serious limitations such as resources and unstable environment. The feature displaying information on who has made modifications to the code received mixed responses from developers. Some of them thought that the ability to identify persons who are actively committing in a particular code area will help them to get a quick help, especially if the person is working in another team. On the contrary, some developers thought that gathering statistics on developers' commits will be a social disaster. It will create an environment of blame and shame and a bonus point market based on the number of commits. Similarly, feature for appreciation on successfully merging the commits with mainline was discouraged by developers as it would create an environment of competition among developers leading to negative impacts on their work. Another proposed feature was to have brief logs so that the developers could easily find the reason for errors. On the contrary, some developers mentioned that they would prefer to have verbose logs with proper log levels.

4.2.4 Environment

The study also investigated the existing work environment practices or possible changes in work environment practices which have a potential to support developer's confidence. Knowledge sharing is one common suggestion from interviewees which would support their confidence. Some developers suggested that an informal discussion with the team about their tasks would provide helpful feedback. One

existing practice of having dedicated team to monitor the CI machinery which monitors codecommit failures and informs the responsible developer to fix the issue, helps support some developers' confidence in the CI machinery. But, a few developers mentioned that their confidence is affected negatively when the team points out their commit failures. However, when this practice was discussed in workshop, developers confirmed that they would all prefer to have such a team to monitor the CI machinery, as it plays a vital part in maintaining the stability of CI system which supports their confidence in the CI tool.

One practice that was suggested from the interviews was to have one member from each team who should work closely with the existing dedicated team which monitors the CI machinery. This suggestion was intended to improve the communication between the monitor team and developers. According to current practice, the team monitoring the system sends a formal mail to the developer that their commit has failed and they need to work on it, after which they have no further help or communication with the team. So, it was suggested to have a member from each team who is part of the CI monitor team, which would help them in understanding their commit failures. This suggestion was further discussed by developers in the workshop and they mentioned that any support the developer needs is provided by their own team members. Each team works internally on fixing the commit issues which is a good practice, concluding that the suggested practice is not needed.

Finally, one practice that was mostly discussed during workshop was how to maintain a stable CI tool or environment. To elaborate, developers discussed that prior to the agile mindset, there were separate test development teams who would continuously improve test quality. However, since the adoption of agile methodology, test and development have been merged. Developers tend to pay more attention to features since these are delivered to customers but very little attention is paid to the improvement of test cases. To exemplify, developers frequently add new test cases for their code, but existing test cases are not reviewed and maintained properly, which might also lead to redundancy. As a result, test cases become expensive both in terms of time and resource leading to poor performance and long feedback time which affects developers' work. Summarizing this concern, one developer stated that *"Fast-track is making them slow "*. Some developers mentioned that they have backlogs which contain improvements for the tool but they are so focused on the features that they have no time to look at the tool improvement. One developer added that *"Always, feature wins over the tool "*. Finally, some developers suggested on having an allocated team to maintain tool and environmental stability, while others highlighted that a few teams have a *"Product care"* concept which refers to maintaining tool improvement backlog and work on it.

Aspects that were perceived to support confidence varied across the groups of interviewees. As developers with low experience emphasized that code reviews and formal knowledge sharing sessions are important supporting aspects for their confidence. On the other hand, developers with high experience level said that they do not rely on code reviews and prefer testing the code on their own. Since they have a

better overall knowledge of the system, they don't emphasize on formal knowledge sharing sessions. Therefore, this distribution of developers over different levels of experience helped us to capture different perspectives.

Some other findings that are indirectly related to developer's confidence are build breakage frequency, code commit frequency and developer's behavior close to deadline. These factors are debatable as developer's confidence is not affected by these but there are some behavioral changes that could be implied as the result of these factors. To exemplify, developers mentioned that close to deadline they change their commit criteria as they have a tight schedule when they are working in the agile way. When asked *"if code commit frequency is proportional to developer's confidence"*, some developers mentioned that it depends on the individual. Some developers suggested that those who have divided their task into smaller deliverables, they make small commits frequently and are more confident as they are not affecting major part of the code. On the other hand, some emphasized that some developers are confident by nature and their commit frequency has got nothing to do with their confidence. Lastly, when asked about the effects of frequent build breakages, some developers mentioned that they are not affected by frequent build breakages as the main purpose of the CI tool is to restrict the faulty commits and this actually helps them to be more confident in the tool. On the contrary, some junior developers are low in confidence when they break the build.

5

Discussion

In this study, we investigated how developer's confidence can be defined in the context of CI and how it can be supported through environment or features in CI tools. In this chapter, we discuss the results from our study and connect with relevant literature. We begin by discussing the overall challenges in adopting CI practices and focus on how developer's confidence can be defined and what aspects can support confidence.

5.1 Confidence

The existing literature in Software Engineering provides rich information on challenges in adopting continuous practices. Shahin et al. (2017) have mentioned various challenges in adopting continuous practices such as lack of awareness and transparency, insufficient tools and technologies, coordination and collaboration challenges, merging conflicts, testing, lack of experience and skills, organizational structure and processes, and dependencies with hardware and other (legacy) applications. In the light of these challenges, the role of a software developer as a CI user is a grey area. As stated by Pinto et al. (2017, p.1), *'In spite of the increasing adoption, the large set of tools, and the well-known benefits, little is known about how software developers are dealing with the usage of continuous integration techniques'*. This section discusses how developer's confidence can be defined in terms of various factors that can potentially address some of these challenges. The fishbone diagram below demonstrates these factors .

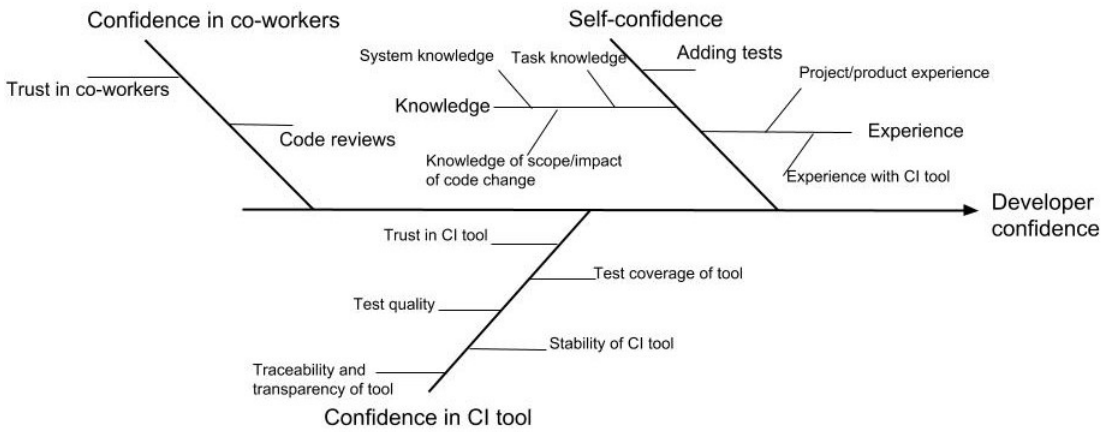


Figure 5.1: Factors affecting developer confidence.

Savor et al. (2016, p.7) state that *'developers need to be generalists with the ability to understand many aspects of the system'* to achieve efficient continuous deployment. During the study, developers recommended that their knowledge about the whole system improves their confidence. This system knowledge relates to the generalism stated by Savor et al. (2016) and helps developers to gain a better understanding of how their commit will play out when merged with the rest of the system. However, our study has highlighted that, in addition to system knowledge, developers' knowledge about their current tasks and about the scope or impact of the code change adds to their confidence and promotes efficient continuous integration.

The knowledge of developers is related to their experience and skill level, as Rahman and Devanbu (2011, p.2) state that *'Knowledge gained from experience matters: it can lead to better ability to answer questions about previous work, and better quality work'*. Eyolfson et al. (2011, p.6) also state that *'the more experienced the developers are, the less likely that their commits are buggy'*. Likewise, our study has also highlighted experience of developers as one of the factors in supporting their confidence in the context of continuous integration. To elaborate, developers stated that with more experience, they are more confident that their commit will be defect-free and merge successfully. Furthermore, our results have shown that developers consider both their experience with the product or project they are working with and their experience with the CI tools and practices as contributing factors to their confidence. The more experienced they are, the more knowledge they possess to efficiently work on their tasks in a continuously integrated codebase.

According to Fowler (2006), one of the core practices of continuous integration is to make the build self-testing. He states that one needs to add automated test suites that check codebase for bugs. Yuksel et al. (2009) emphasize that in order to achieve efficient continuous integration, developers need to design, code and run all the necessary unit and integration tests before codecommit. Although interviewees have confirmed that this practice of adding sufficient tests before codecommit enhances their confidence, but some issues have been raised as well. It was mentioned that in today's agile world, developers focus more on delivering features on fast-track. Everyone adds tests for their code to meet the test coverage criteria for their codecommit but only a few invest in cleaning the existing tests in the CI tool. As a result, the tool is burdened with overlapping tests leading to lower quality of tests in the tool which consequently lowers developers' confidence in the tool. Developers added that to address the test quality issue, they invest individual efforts in adding tests that are sufficient to meet the coverage criteria and are of good quality, however, there is a need to recognize this as a standard practice to maintain test quality in tool. It implies that developers favour quality as well as quantity when it comes to adding tests for their codecommit. Therefore, in addition to the test coverage criteria which indirectly emphasizes on quantity of tests, organizations need to introduce practices that put focus on test quality as it is equally important in supporting developer's confidence.

Existing literature presents mixed views on the efficiency of code reviews in find-

ing bugs. Bavota and Russo (2015, p.1) state that *'unreviewed commits have a much higher chance (over two times) of inducing bug fixes with respect to reviewed commits'*. Whereas, Czerwonka et al. (2015) state that code reviews do not find functionality issues, must be performed by person with specific skill set and have social aspect that can not be ignored. Our results have also highlighted mixed opinions on code reviews. Some developers think that code reviews from a knowledgeable person in a particular area are really helpful in getting a second opinion on their solution and enhance their confidence. It is due to this reason that developers favored the feature of selecting a reviewer for their commit as it enabled them to request an experienced person for review. On the contrary, some developers said that they do not rely much on code reviews as they sometimes have low trust in reviewers. Additionally, some developers stated that the trust they have in the manual testers who test their code before delivery also affects their confidence. Developers are unsure of how their code will be handled by the testers and this affects their confidence negatively. The aspect of trust within agile teams is discussed by McHugh et al. (2012, p.1), who state that *'Trust requires team members to believe that their colleagues possess the knowledge, competence, and integrity to complete their assigned tasks'*. Based on our findings, we suggest that trust similarly influences interactions among people working in different teams and across the parts of CI pipeline and affects their confidence in each other.

The third and the final viewpoint on developer confidence is the confidence of developers in the CI tool they're using. During study, developers mentioned that having a high level of trust in the CI tool improves their confidence as they believe that any uncaught bugs in the codecommit will be caught by the tool. Whereas, some developers said that they do not trust the tool much and rely on adding sufficient tests for their codecommit with test quality in focus, because the tool is only as good as the tests they add. This is in agreement with the stance of Pinto et al. (2017, p.4) who state that blindly trusting the tests in the tool creates a false sense of confidence as *'The continuous integration pass is only as meaningful as the test coverage'*. Therefore, the higher the test coverage a tool has, the higher the trust developers have in the CI tool.

Developers' trust in the CI tool also depends on the stability of CI tool and environment. The need for stable environment for adopting continuous integration is emphasized by Shahin et al. (2017, p.4), who state that *'CI is a foundation for CDE, in which implementing reliable and stable CI practice and environment should be the first and highest priority for a given organization to successfully adopt CDE practice'*. During the study, developers have repeatedly mentioned stable environment as a challenge and that there is a need to focus on getting the environment stable as it is an enabler of different features that can support their confidence. Persistent instability of environment causes them to lose their trust in the tool because they are unsure if the failures are due to defects or unstable environment. It is reported that intermittent faults in the continuous integration pipeline occur frequently and cause confusion and delays in feedback. Some developers mentioned solutions such as product care backlogs and dedicated tool team to cope with this challenge but

these would be costly in terms of time and resources. Having product care backlogs would require developers to spare some time from developing features which can possibly slow down feature delivery. So, there is a trade-off between achieving stable environment and delivering features fast. Depending on the nature of project and product, organizations can prioritize one thing over another and allocate necessary resources to it. However, as mentioned previously, developers quoted that *"Fast-track is making them slow"* which warrants a careful consideration of this issue.

5.2 Support

This section focuses on how confidence of developer, as discussed in previous section, can be supported to overcome some of the CI adoption challenges mentioned previously. It also sheds light on the potential and limitations of these supporting aspects to address these challenges in a better way. The fishbone diagram below demonstrates the features from CI tools and environmental aspects that can potentially support developer's confidence.

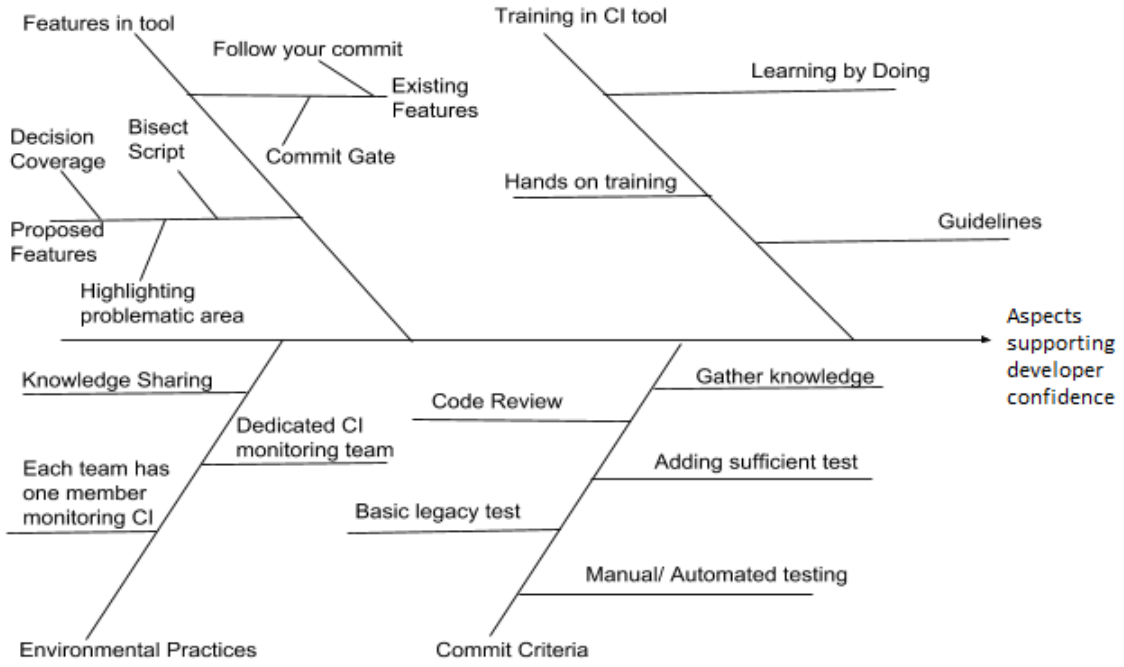


Figure 5.2: Supporting aspects for developer's confidence.

Both interviews and workshop discussions highlighted the need of knowledge sharing sessions which could either be formal or informal. Gervigny and Nagowah (2017, p.1) describe the goal of knowledge sharing as to allow *'the right people to gain access to the correct content at the right time'*. They also discuss the importance of this practice in helping developers to get relevant information and expertise. Moreover, during our case study, developers emphasized that having information on whom to ask for help, what each team member is working on or has worked on in the past

would aid them in decision making and problem solving skills. The need for knowledge sharing is also reflected in the proposal of a feature called *"information on who made modifications to code"* which helps in knowing who commits frequently to a particular code area in order to seek help from them.

One of the existing practices within the case company is having a dedicated team to monitor the CI machinery. This practice was found to be the most important and an added value to the work environment as it is crucial for maintaining the CI machinery. This is related to the viewpoint of Shahin et al. (2017) who assert the implementation of stable CI practices and environment as a top priority to overcome the challenges concerning CI adoption. Nevertheless, during the case study some developers stated that communication between the monitoring team and developers is limited, which is a barrier in understanding the reason for their commit failures. So, they suggested a two way communication with the monitor team in diagnosing the problem. On the contrary, some developers did not see the need of a two-way communication and suggested that the team members collaborate internally to find the issue. This practice is good for team coordination enabling transparency in the team and transparency was stated as one of the problem when adopting CI and CD by Laukkanen et al. (2017).

Savor et al. (2016) describe that education is vital because the software process with continuous practices is different from what was traditionally preached and practiced. They also highlight the investment that is required with respect to educating and training developers to encourage adoption of CI practices. This is in line with one of the assumptions in our case study that trainings in CI tools or practices can support developer's confidence. Although some developers favoured this assumption, others preferred *"learning by doing"* over traditional classroom trainings. While describing why employees prefer *"learning by doing"*, Schank (1995, p.2) stated *'If you do something often enough, you get better at it'*. He further adds that when people care about what they are doing, they will try to learn better ways of doing it; it's human nature.

Code coverage helps to analyze what percentage of the source code is tested. It was also discussed during the case study, where developers highlighted that code coverage is an important metric for supporting their confidence in the CI tool and having a threshold of 80% on it enforces developers to write sufficient tests before the codecommit. On the other hand, some developers suggested that there should be a 100% code coverage criteria with exclusion marks indicating the code which cannot be tested. According to them, having 80% is as bad as having 60 or 70% and the level falls down to whatever level you decide sooner or later. On the contrary, Marick (1999, p.7) states that *'Designing your initial test suite to achieve 100% coverage is an even worse idea'* and further describes it as a way of creating a test suite which is weak in finding important faults of omission. Hemmati (2015) discusses the effectiveness of code coverage criteria and mentions that even a test suite with 100% coverage fails to detect all faults. Therefore, in the light of existing literature and suggestions from developers, threshold for code coverage metric

is specific to developers and organizations. Finally, one additional suggestion from workshop discussions was to have a decision coverage criteria. Antinyan et al. (2018, p.1) describes decision coverage as *'the percentage of decision blocks in a file that have been exercised during a test run'*.

Throughout the case study, traceability of commit was mainly highlighted as an additional factor that supported developer's confidence. **?** also mentioned traceability as a key challenge in CI. During our study, developers discussed about an existing feature called *"Follow my commit"*, which enables developers to trace their commit in the CI machinery and helps them to achieve better traceability in the tool. This feature could be related to, *'Follow your commit, feature from Ericsson's Eiffel framework, which provides developers with real time view and timeline of their recent commits'* (**?**, p.4). Additionally, during the case study developers confirmed that such a feature enables better transparency in the tool, further building their confidence in the CI tool, which indirectly supports developer's confidence during codecommit.

"Highlighting problematic code areas", was a feature suggested in our study, helping developers to have a clear idea of the codebase. This feature relates to having transparency in the tool and helps the developers as they are well aware of the risk they are taking while making commits in few problematic code areas. *Identify hotspots in your code* (n.d.) describes a similar approach which uses the commit information to identify hotspots and risky commits. Hotspot can be a file which contains sensitive code that is risky to change or code which is not tested properly. These are identified based on commit information and a file is flagged as hotspot if it was recently changed while fixing a defect. Additionally, *'a risk indication for commits that include changes to hotspot files and for features associated with such commits'* (*Identify hotspots in your code*, n.d., p.4) is also mentioned. Therefore, a feature related with this idea would help the developer to be more confident when they know that the code area they are committing to is not risky.

During the case study, some developers suggested to have a feature to appreciate developers who have successful commits or don't make faulty commits, as such a feature would encourage developers to do good work. *'Make sure that the teams are well rewarded and morale is high'*, Chopra (2014, p.3) stated with respect to motivating the team with agile framework. On the other hand, some developers mentioned that such a feature would increase competition in the team and negatively impact the team spirit. To elaborate, *Individual Recognition and Team Performance; The Good, the Bad and Our Solution.* (2017, p.2) states that *'If you are recognizing individual efforts, your team in turn will most likely continue to operate as individuals rather than collectively as a team'*. It also discusses how individual recognition demotivates team members to partake in team activities.

Ziftci and Reardon (2017, p.2) state *'If code development continues in the presence of failing tests, other developers might introduce more regressions along the way before the current regression is resolved'*. This situation emphasizes the need for

quick feedback to resolve these problems as soon as possible. During the study, developers repeatedly emphasized on the importance of quick feedback from merge results in the higher levels of CI pipeline. They added that this feedback helps them to identify the issues and remove these issues from the pipeline as soon as possible. Fast removal of bugs from pipeline is important to them because the longer the bugs stay in the pipeline, the more it aggravates the problem. Quick feedback helps to deliver fast without causing delays in feature development. It is in agreement with *'Identifying changes that introduce regressions to a codebase is critical to keep the momentum on software development, especially in very large scale software repositories with rapid development cycles'* (Ziftci and Reardon, 2017, p.10). In order to solve this problem, developers proposed a feature *"bisect script"* that takes one commit at a time and run tests on it to find if it is the culprit for failure. Developers added that it will be costly in terms of time and hardware resources. Some other variants of this feature were also proposed by developers such as using binary search on commits or running selective tests for each commit to reduce time and resource consumption. Nevertheless, this feature is still seen as a challenge to implement within the company. Ziftci and Reardon (2017) have done some work to find the buggy commit in the codebase. They have suggested an n-ary search instead of binary search to find the buggy commit. Additionally, they have presented an algorithm that uses heuristics to filter and rank the change logs in order to help developers at Google to find the culprit. They also discuss running time of the algorithm and how to deal with unstable tests. As a future work to their study, they propose to include logs analysis into heuristics to improve its performance. However, we suggest that this algorithm might have different outcomes when implemented in different organizations and there is a need to explore it further to make it more generalizable.

6
Conclusion

In this chapter, we present the implications of our study for organizations and practitioners. We then conclude our study along with a discussion on validity threats to our study.

6.1 Implications and Conclusion

Our case study has proposed a definition of developer's confidence in the context of CI and how it can be supported. These proposals have some implications for practitioners and researchers in supporting developers' confidence and addressing the challenges faced in CI adoption.

Key factors supporting developers' confidence are knowledge and test quality. Based on these findings, we imply that knowledgeable developers are more confident and suggest that organizations should promote a knowledge sharing culture to facilitate developers in seeking knowledge required to support their confidence. Additionally, test quality in the tool is as important as test quantity in supporting developers' confidence, which implies that organizations should promote the practice of cleaning redundant tests and maintaining test quality in the CI tools. Developers' confidence also depends conditionally on their confidence in the tool and co-workers. Stability of CI tools enhance developers' confidence in the tool. But, achieving this stability consumes resources and tools might never be fully stable. Given the need to deliver fast in the agile era, practitioners and organizations should balance the resources allocated to tool stability and fast delivery based on the nature and requirements of the projects. Confidence of developers is also strengthened by the trust they have in manual testers who will test their codes in later parts of CI pipeline. Building trust among the developers and testers can enhance developers' confidence, therefore, organizations should promote communication among people working in different parts of the CI pipeline.

Proposed features such as *"Highlighting problematic code area"* and *"Information on who made changes to code"* depict that developers need higher transparency of information available in the CI tools. Focusing on improving the transparency of CI tools can provide valuable information to developers for strengthening their confidence. Experienced developers see code reviews as a complementary practice to support their confidence, whereas, junior developers consider code reviews important to get expert opinions on their code. Similarly, junior developers prefer formal

knowledge sharing sessions as opposed to senior developers who believe that knowledge is shared informally within the team all the time. These findings highlight the experience based differences among developers regarding their confidence and its supporting aspects, which can steer the organizations' mindset in catering the needs of developers with varying experience levels.

To conclude, we believe that as immediate users and practitioners of CI tools and principles, developers' perspective and viewpoint must be studied in more detail in order to overcome adoption challenges of continuous practices. Organizations need to adapt the CI tools to developers' way of working to promote efficient usage of continuous integration in order to fully reap its benefits. Confidence of developers is of prime importance while practicing CI as they are now directly responsible for their commits. It warrants a need to add features to CI tools and improve work environment of developers to support their confidence. Suggestions for these features and work environment have been made through this case study, which can be validated through implementation and evaluation in different contexts and can serve as inspiration for future work in successful adoption of CI. Also, as a future work, our study could be conducted with different context and with different subject selection criteria such as cultural difference or gender, to further explore developer's confidence.

6.2 Validity Threats

This section describes the threats to the validity of our case study.

Internal Validity: The selection of developers for interviews was based on their experience level. It could be based on other factors such as gender, cultural differences and different domains in software development. However, our study has highlighted that developers' confidence varies with their experience. As a future work, further research can be made to explore developers' confidence based on above mentioned selection criteria.

External Validity: Given the nature of case studies, there are known limitations regarding generalizability. Our study has provided rich understanding of developers' confidence in the environment under study. We have generalized our results through abstractions and connected our results to existing theory in CI and generated suggestions for CI practitioners and organizations. Hence, our results can potentially be applicable to other contexts and organizations. However, further studies that cover a diversity of context and other aspects, may yield further understanding of developers' confidence in CI.

Construct Validity: Confidence is a diverse concept and has various interpretations under its umbrella. Therefore, different interpretations of confidence among developers were a threat to our study, which we have addressed by establishing a common understanding of confidence with the developers. During the interviews,

developers were presented with the formulated definition of confidence based on existing literature. This definition was used to reach a common understanding of our intended research and guided the interviewees in their reflections.

Bibliography

Adams, B. D. (2005), Trust vs. confidence, Technical report, HumanSystems Inc Guelph (Ontario).

Alhojailan, M. I. (2012), 'Thematic analysis: A critical review of its process and evaluation', *West east journal of Social Sciences* **1**(1), 39–47.

Antinyan, V., Derehag, J., Sandberg, A. and Staron, M. (2018), 'Mythical unit test coverage', *IEEE Software* **35**(3), 73–79.

Bavota, G. and Russo, B. (2015), Four eyes are better than two: On the impact of code reviews on software quality, *in* 'Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on', IEEE, pp. 81–90.

Bugayenko, Y. (2009), Quality of code can be planned and automatically controlled, *in* '2009 First International Conference on Advances in System Testing and Validation Lifecycle', pp. 92–97.

Carter, N., Bryant-Lukosius, D., Dicenso, A., Blythe, J. and J Neville, A. (2014), 'The use of triangulation in qualitative research', **41**, 545–547.

Chen, L. (2017), 'Continuous delivery: Overcoming adoption challenges', *Journal of Systems and Software* **128**, 72 – 86.

Chopra, S. (2014), Implementing agile in old technology projects, *in* 'Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization', pp. 1–4.

Czerwonka, J., Greiler, M. and Tilford, J. (2015), Code reviews do not find bugs: how the current code review best practice slows us down, *in* 'Proceedings of the 37th International Conference on Software Engineering-Volume 2', IEEE Press, pp. 27–28.

Dam, R. and Siang, T. (2017), 'Stage 3 in the design thinking process: Ideate', `https://www.interaction-design.org/literature/article/stage-3-in-the-design-thinking-process-ideate`. Accessed:2018-05-26.

Eyolfson, J., Tan, L. and Lam, P. (2011), Do time of day and developer experience affect commit bugginess?, *in* 'Proceedings of the 8th Working Conference on Mining Software Repositories', MSR '11, ACM, New York, NY, USA, pp. 153–162.

Faste, H., Rachmel, N., Essary, R. and Sheehan, E. (2013), Brainstorm, chainstorm, cheatstorm, tweetstorm: New ideation strategies for distributed hci design, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '13, ACM, New York, NY, USA, pp. 1343–1352.

Fowler, M. (2006), 'Continuous integration', `http://www.martinfowler.com/articles/ContinuousIntegration.html`. Accessed:2018-01-13.

Gervigny, M. L. I. and Nagowah, S. D. (2017), Knowledge sharing for agile distributed teams: A case study of mauritius, *in* 'Infocom Technologies and Unmanned Systems (Trends and Future Directions)(ICTUS), 2017 International Conference on', IEEE, pp. 413–419.

Hemmati, H. (2015), How effective are code coverage criteria?, *in* '2015 IEEE International Conference on Software Quality, Reliability and Security', pp. 151–156.

Identify hotspots in your code (n.d.), `https://admhelp.microfocus.com/octane/en/latest/Online/Content/UserGuide/hotspot-files-in-code.htm`. Accessed:2018-05-05.

Individual Recognition and Team Performance; The Good, the Bad and Our Solution. (2017), `https://www.critical-metrics.com/individual-recognition-team-performance/`. Accessed:2018-05-05.

Kapelonis, K. and Engineer, K. (2016), '9 features you need to demand from a hosted continuous integration service', `https://zeroturnaround.com/rebellabs/9-features-you-need-to-demand-from-a-hosted-continuous-integration-service/`. Accessed:2018-05-13.

Laukkanen, E., Itkonen, J. and Lassenius, C. (2017), 'Problems, causes and solutions when adopting continuous delivery—a systematic literature review', *Information and Software Technology* **82**, 55 – 79.

Leppanen, M., Makinen, S., Pagels, M., Eloranta, V., Itkonen, J., Mantyla, M. V. and Mannisto, T. (2015), 'The highways and country roads to continuous deployment', *IEEE Software* **32**(2), 64–72.
URL: *doi.ieeecomputersociety.org/10.1109/MS.2015.50*

Marick, B. (1999), How to misuse code coverage, *in* 'Proceedings of the 16th International Conference on Testing Computer Software', pp. 16–18.

McHugh, O., Conboy, K. and Lang, M. (2012), 'Agile practices: The impact on trust in software project teams', *Ieee Software* **29**(3), 71–76.

Meyer, M. (2014), 'Continuous integration and its tools', *IEEE Software* **31**(3), 14–16.

Mårtensson, T., Ståhl, D. and Bosch, J. (2017), Continuous integration impediments in large-scale industry projects, *in* '2017 IEEE International Conference on Software Architecture (ICSA)', pp. 169–178.

Navajas, J., Hindocha, C., Foda, H., Keramati, M., Latham, P. E. and Bahrami, B. (2017), The idiosyncratic nature of confidence, *in* 'Nature Human Behaviour'.

Pinto, G., Reboucas, M. and Castor, F. (2017), Inadequate testing, time pressure, and (over) confidence: A tale of continuous integration users, *in* '2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)', pp. 74–77.

Polkhovskiy, D. (2016), Comparison between continuous integration tools, Master thesis, Dept. of Pervasive Computing. Information Technology, Tampere University of Technology.

Port, D. and Wilf, J. (2013), The value of certifying software release readiness: An exploratory study of certification for a critical system at jpl, *in* '2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement', pp. 373–382.

Rahman, F. and Devanbu, P. (2011), Ownership, experience and defects: a fine-grained study of authorship, *in* 'Proceedings of the 33rd International Conference on Software Engineering', ACM, pp. 491–500.

Runeson, P. and Höst, M. (2008), 'Guidelines for conducting and reporting case study research in software engineering', *Empirical Software Engineering* **14**(2), 131.

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K. and Stumm, M. (2016), Continuous deployment at facebook and oanda, *in* 'Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on', IEEE, pp. 21–30.

Schank, R. C. (1995), What we learn when we learn by doing, Technical report, Northwestern University.

Schenk, R. (n.d.), 'Dollar voting', `http://ingrimayne.com/econ/AllocatingRationing/DollarVoting.html`. Accessed:2018-05-26.

Schermann, G., Cito, J., Leitner, P. and Gall, H. C. (2016), Towards quality gates in continuous delivery and deployment, *in* '2016 IEEE 24th International Conference on Program Comprehension (ICPC)', pp. 1–4.

Shahin, M., Babar, M. A. and Zhu, L. (2017), 'Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices', *IEEE Access* **5**, 3909–3943.

Ståhl, D. and Bosch, J. (2017), 'Cinders: The continuous integration and delivery architecture framework', *Information and Software Technology* **83**, 76 – 93.

Webster (1913), `www.websters1913.com`. Accessed:2018-03-13.

Wells, D. (1991), 'Integrate often', `http://www.extremeprogramming.org/rules/integrateoften.html`. Accessed:2018-01-14.

Yuksel, H. M., Tuzun, E., Gelirli, E., Biyikli, E. and Baykal, B. (2009), Using continuous integration and automated test techniques for a robust c4isr system, *in* 'Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on', IEEE, pp. 743–748.

Ziftci, C. and Reardon, J. (2017), Who broke the build? automatically identifying changes that induce test failures in continuous integration at google scale, *in* '2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)', pp. 113–122.