Tampere University

Yulia Bardinova

IMPROVEMENT OF CONTINUOUS INTEGRATION WORKFLOW WITH USER RESEARCH

Master of Science Thesis
Faculty of Information Technology
and Communication Sciences
Examiners: Harri Siirtola
Roope Raisamo
Supervisors: Harri Siirtola
Kari Seppälä
October 2022

ABSTRACT

Yulia Bardinova: Improvement of Continuous Integration Workflow with User Research
Master of Science Thesis
Tampere University
Human-Technology Interaction
October 2022

In a fast-paced, continuously changing IT industry, it is important for organizations to deliver their products both quickly and with high quality. Continuous practices help achieve that. Specifically, Continuous Integration (CI) aims at submitting code frequently and consistently, checking every new piece of code with automated tests. Therefore, software malfunctions are caught early. At the same time, improper implementation of CI can hinder the development process. Thus, a proper CI assessment is required. The work utilizes a survey to gather employee feedback regarding CI, and then analyzes it with a thematic matrix. Even though employees' general attitude towards CI is positive, they highlight its instability, slowness, lack of documentation, and unclear error messages, among other findings. Additionally, the study revealed deep-rooted issues like shortage of employees and no communication channel between developers and the CI team. In the end, we produced a CI improvement plan with a list of tasks to implement to achieve user goals of the employees.

Keywords: continuous practices, quality assurance, human-technology interaction

PREFACE

The thesis was developed and written at a company, and it neatly integrates with the tasks I perform there. The company where I work wished not to disclose its name.

I would like to thank my colleagues, friends and family for supporting me during the thesis writing. Particularly, I want to express my gratitude to my supervisors Harri Siirtola for guiding me through the thesis writing process and Kari Seppälä for helping with pitfalls of a corporate thesis.

I would also like to thank my colleagues for dedicating their time to participate in the survey and providing valuable feedback, as well as my friends who gave me writing tips and reminded me to keep working on the thesis.

Tampere, 6th October 2022

Yulia Bardinova

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

BQ	Background question
CD	Continuous Deployment
CDE	Continuous DElivery
CI	Continuous Integration
DevOps	Software development (Dev) and IT operations (Ops)
HCD	Human-Centered Design
HCI	Human-Computer Interaction
HTI	Human-Technology Interaction
IP	Semiconductor Intellectual Property Core
OEQ	Open-ended question
SBQ	Scale-based question
SCM	Source Code Management
UCD	User-Centered Design
VCS	Version Control System
XP	eXtreme Programming

1. INTRODUCTION

1.1 Motivation

In a highly competitive software development market, companies strive to produce software both faster and with higher quality than their competitors. To achieve this, continuous practices are taken to use [1]. Continuous practices refer to Continuous Integration (CI), Continuous Deployment (CD) and Continuous DElivery (CDE). CI advocates for submitting frequent changes, and CD with CDE aim at releasing and delivering the product to the customer quickly and reliably [2].

The practice of CI is gaining more popularity in the industry over the recent years [3]. One evidence is the increasing search frequency of the topic "CI/CD" in Google search engine (see Fig. 1.1) [4]. The reason for it is CI's benefits. It accelerates the development process without compromising software's quality [2]. Research shows that implementing CI increases developers' productivity [5]. Moreover, CI has an improvement over the communication both within the teams and between them, makes troubleshooting more effective, and improves project predictability [6].

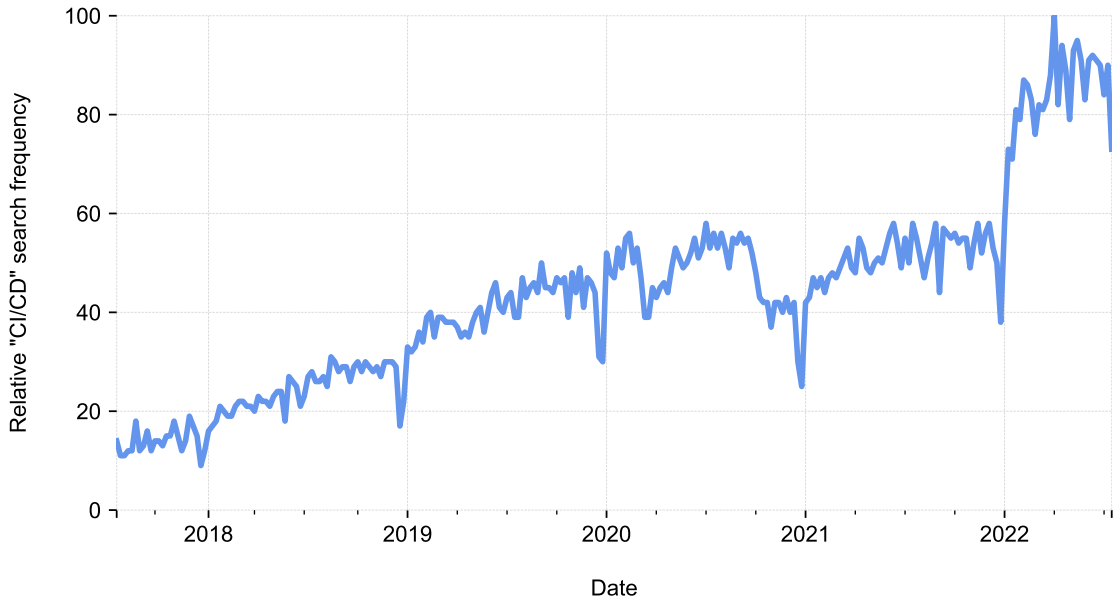

Figure 1.1. *The interest in the topic of "CI/CD" has been steadily increasing over the last 5 years, according to Google Trends [4].*

At the same time, one must note that all these advantages of CI are relevant and reachable only if the company implements it correctly [7]. Sometimes, the project stakeholders keep in mind the user needs of the final customers only, neglecting the user needs of the developers involved in the project. So, the continuous practices and their tools should be utilized according to the user – i.e., software engineer – and project needs. Otherwise, improper application can result in a slower, ineffective, hard-to-maintain pipeline [7].

As the company or a project expands, the CI workflow expands as well and becomes more complicated. Any inefficient CI practices can pile up, leading to developers' frustration [7]. And so, engineers' productivity can decrease. Not only the software developed by engineers needs to satisfy user needs, but also the processes and tools involved in software development require user (in this case, engineer) satisfaction.

And thus, the usability assessment of the CI environment is required. **This thesis aims at improving the company's CI workflow by gathering feedback from the employees, analyzing it and suggesting an improvement plan.**

1.2 Objective and scope

The company utilizes CI to release high-quality products in a competitive market. However, over the years of its use, implementation errors occurred. Thus, continuous practices are not applied to their full potential.

The purpose of this thesis is to improve the usability of a current CI workflow within the company's department. The thesis has three research objectives:

Research Objective 1: To identify experienced issues and functionality ideas of CI users by conducting user research.
Research Objective 2: To analyze the user study data by appropriate means.
Research Objective 3: To develop a roadmap with tasks aimed at improving CI's usability based on the obtained results.

The author conducts user study in a form of an online survey to discover the needs of the engineers for the workflow. Based on the user research, the author suggests an improvement roadmap for the company.

The research is carried out within one department (System-on-Chip Software) of the company among its employees. The company where the author prepared the thesis wished not to disclose its name.

1.3 Thesis structure

The outline of the thesis work is as follows. Chapter 2 discusses the background of the topic, explaining the key concepts behind CI and the terms that will be used further in the document. Chapter 3 focuses on the utilized methodology. Chapter 4 features the analysis of the problem via the use of a survey distributed among the employees to gather data. Next, chapter 5 discusses the results of the analysis. Finally, chapter 6 summarizes the thesis.

For the reader's convenience, a summary section is provided at the end of each following chapter. Only the last chapter doesn't have a summary, as it concludes the thesis.

2. BACKGROUND

This chapter discusses the key concepts related to the thesis work. Section 2.1 shows the importance of human-centered approach to design and the role of usability. Section 2.2 explains the meaning behind continuous practices and their benefits. Next, Section 2.3 shows what a deployment pipeline looks like. Finally, Section 2.4 lists the tools typically used in continuous practices and a deployment pipeline. Chapter summary is available at the end in Section 2.5.

2.1 Human-Technology Interaction

Human-Computer Interaction (HCI) is a research area that concentrates on interaction between a computer interface and a user [8]. It researches novel ways a person can interact with a computer and what makes the interaction effective. Human-Technology Interaction (HTI) is an extension of HCI used in a broader context, i.e., HTI researches interaction between a user and any technology.

Both of these fields of study assist designers and users in improving the system or a product by specifying user needs and satisfying them. Human-Centered Design (HCD), often used interchangeably with User-Centered Design (UCD), is an approach in the design process that aims at problem-solving by relying on a human's (user's) perspective [9]. It does not mean that a product should be designed for everyone, but instead that HCD accommodates for people for whose needs and capabilities it is intended.

Utilizing HCD approach is beneficial to product design [10]. It helps gain a deeper understanding of the user, therefore resulting in the following advantages. The productivity increase is possible due to effective operating of the system by the user, achieved via following of the usability principles, which will be discussed further. Often errors occur due to poorly designed interface, so avoiding ambiguous design will help reduce the errors. Additionally, these principles accelerate product operation learning, resulting in reduced training time. Finally, the combination of the above gains a reputation boost to the product, attracting more customers [10].

One of the key concepts of HTI and HCD is usability. Usability is a degree to which users can utilize a service with stated goals in a stated context of use with regard to

effectiveness, efficiency and satisfaction [11]. Usability is recognized to be vital to gain success for an interactive system or a product [12, 13, 14]. Utilizing HCD approach and usability principles will help in the process of this thesis work and, if applied correctly, will result in a usable and effective workflow.

2.2 Continuous practices

Continuous software engineering is a research area and an industry practice. It has been taken into use since 1990s, when an eXtreme Programming (XP) approach emerged [15], which adopted the core concept behind continuous practices [16, 17].

Continuous practices aim at accelerating the development and deployment workflow in the software industry without compromising the product quality [18]. Continuous practices include Continuous Integration (CI), Continuous DElivery (CDE) and Continuous Deployment (CD).

In CI practice, the emphasis lies in frequent, consistent changes committed throughout the software development cycle. At the same time, Continuous DElivery focuses on the product being in a reliable, ready-to-release state at any given time, ensured by automated tests. Finally, Continuous Deployment practice aims at automated and continuous deployment of the product to the customer. To better understand the relationship between these practices, see Fig. 2.1. The testing levels illustrated in the figure are described further in the next section.

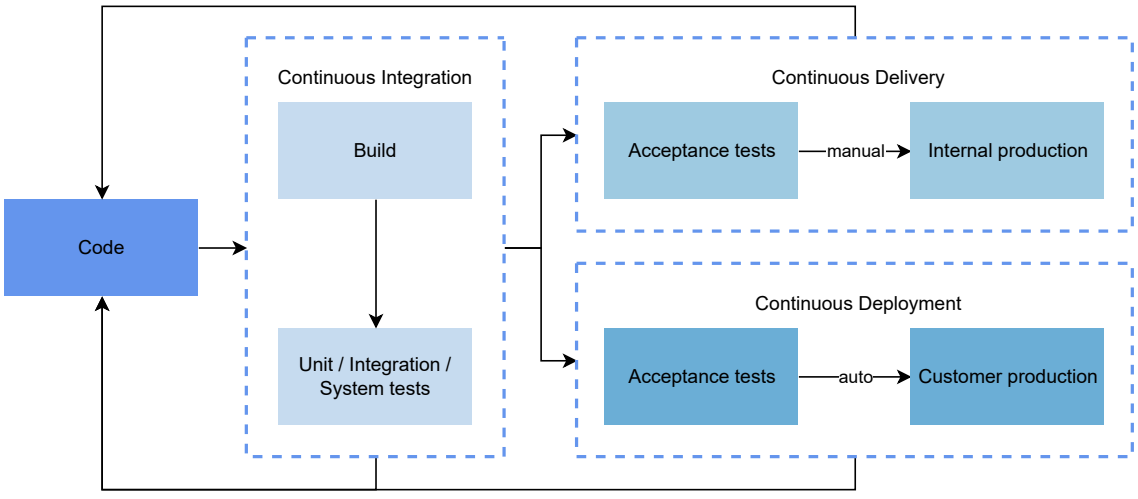

Figure 2.1. *An interpretation of a relationship between CI, CD and CDE [18].*

What distinguishes Continuous DElivery from Continuous Deployment is the production environment, where Continuous Deployment is intended for actual customers. Another difference is the approach to the release method: Continuous DElivery utilizes manual releases, while Continuous Deployment relies on automated ones [19]. Researches call

Continuous Deployment a continuation of Continuous DElivery and suggest that Continuous DElivery is more universal compared to Continuous Deployment practice [19].

Continuous practices bring several benefits to the software development process, such as: (i) frequent and reliable product releases; (ii) regular feedback from users, customers and contributing developers; (iii) automation of manual tasks; (iv) better communication between developers and operations team [18].

2.3 Deployment pipeline

The deployment pipeline is important to describe as it illustrates the components involved in CI. The term *deployment pipeline* refers to the automated process of getting the software from the development phase to the release phase. It can differ between organizations, but generally, the pipeline's skeleton includes building the software, testing and releasing it. Historically, the process was performed manually, but a modern deployment pipeline automates the tasks [2].

The pipeline typically starts with a new change submitted to a Version Control System (VCS) (see Fig. 2.2). Every change triggers a software build that has to pass through a variety of automated tests. The goal is these tests is to minimize the chance of errors occurring in the software. The tests can still have slight differences in their purpose, because they check the software from different perspectives.

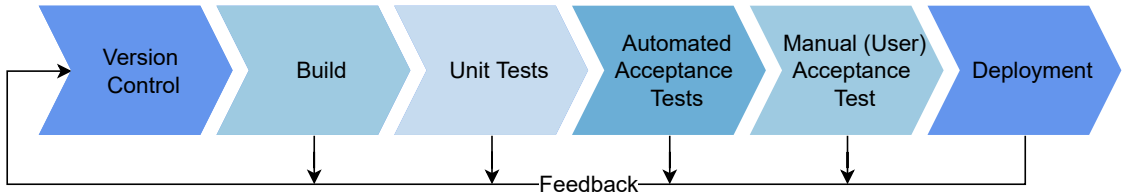

Figure 2.2. *An example of a basic deployment pipeline [2].*

Testing is usually divided into 4 levels: (i) unit testing; (ii) integration testing; (iii) system testing; and (iv) acceptance testing (see Fig. 2.3) [20]. They start from the smallest possible scope, a unit of code, and move on to a bigger one where the interaction between different parts of software is tested. At the same time, there are different practices of implementation of these levels; testing can have additional levels (e.g., subsystem testing) or exclude some of them.

At the stage of the unit testing, software bugs are cheaper to fix in comparison to the other levels, as they are caught early and focus on the smallest piece of code [20]. Integration tests center around the interaction between the components or units, while system testing checks software as a whole.

Then, acceptance tests ensure that the software acceptance criteria, like user needs and

customer specifications, are met. Acceptance tests, unlike the other levels, are often both automated and manual. Automated acceptance testing centers around measurable specifications, while manual acceptance testing (often called user acceptance testing) asserts the usability of the software [2].

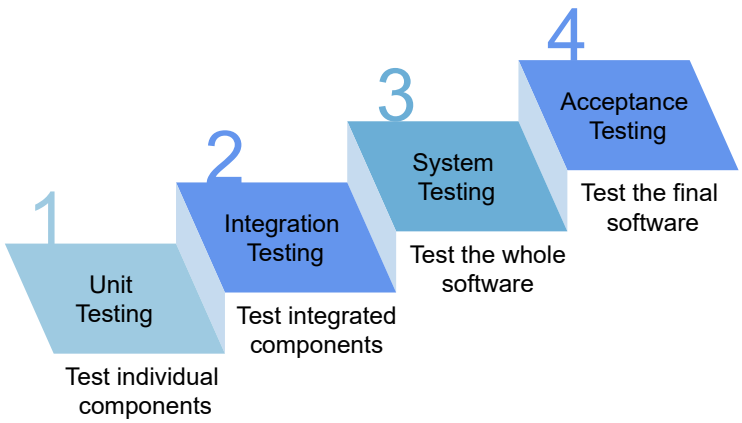

Figure 2.3. *Levels of software testing [20].*

Additionally, all these levels of testing can have different types of tests: functional and non-functional [20]. They focus on characteristics of software. Functional characteristics include software-specific functionalities as well as security, accuracy, interoperability and others; non-functional characteristics can be reliability, usability, efficiency, portability, etc. [20].

If the software fails at any stage, the pipeline should start from the beginning. The developer has to address failed tests and the feedback, and submit new code to fix the errors. A new build is created, which needs to run through tests again until all are passed. This cycle prevents unfit software from reaching the release stage.

After passing all the tests, the software is ready for release. It can be released as a packaged software or deployed to a production environment. The tools that make up the pipeline are discussed in more detail in the next section, 2.4.

2.4 Tools of continuous practices

There are a variety of tools that facilitate the deployment pipeline. Shanin et al. [18] suggest splitting the tools into 7 categories, or stages: (i) version control systems; (ii) code management and analysis tools; (iii) building tools; (v) continuous integration servers; (vi) testing tools; (vii) configuration and provisioning; and lastly (viii) continuous delivery or deployment servers. It should be noted that there isn't a universal standard for a pipeline, and not every stage is compulsory [1]. At the same time, these categories provide a good idea of a typical pipeline.

The first stage involves developers submitting code to a repository that is integrated with a Version Control System (VCS). Version control keeps the history of every submitted change with the timestamp and the author's name. Modern VCS allow the developers to work on the same files in parallel with the use of branches, keeping the changes isolated from each other and merging them when required [21]. Version control is important to track down regression bugs and retrieve an older version of a repository.

Next, code management and analysis tools scan the code for possible software bugs and vulnerabilities. Additionally, they perform quality checks to meet code quality objectives. While performing the analysis, the tools also collect metrics such as test code coverage [22].

The third pipeline stage is build systems. The process itself differs between the programming languages used to write the code and the operating systems, but the concept of software building is the same. Building tools or systems retrieve the source code from the repository and convert it into software artifacts, e.g. executable programs or application binaries [23].

CI servers check the repository for code changes and trigger build processes and tests for them. These servers can additionally be configured to run code checks like code coverage, quality and style [18]. Furthermore, some CI servers also function as CDE/CD servers and thus are able to deploy software to a production environment [18].

The next step is to test the build with the testing tools. As stated earlier, this stage is often performed by a CI server. Software testing is a process of assessment of the software quality to reduce the risk of non-intending behavior [24]. It is usually done by comparing the software against its original requirements. Testing tools automate this process. They play a big role in quality assurance, since manual testing is more time-consuming and is prone to human error [25].

Configuration and provisioning tools are used to manage and automate the configuration of servers involved in the deployment pipeline. Provisioning refers to the process of setting up the infrastructure of the servers, while configuration is the establishment and maintenance of the structure to a desired state, and it comes after provisioning.

Lastly, CDE/CD servers automate the process of software artifact deployment to a production environment. This functionality can also be combined with continuous integration processes, as done, for example, in a Jenkins automation server [26]. However, having a separate CDE/CD server can help distinguish between internal and external releases, as implemented in [27].

2.5 Chapter summary

Human-Technology Interaction (HTI), an extension of Human-Computer Interaction, researches interaction between a user and technology. Human-Centered Design (HCD), which is an HTI design approach, involves human's perspective to design problem-solving. Its use is beneficial to product design, because HCD follows usability principles. Usability is a degree to which users can utilize the service with regard to effectiveness, efficiency and satisfaction. It is vital for an interactive product's success.

The thesis strives to improve usability of Continuous Integration (CI). CI is part of continuous practices. They aim at accelerating the product's development and deployment workflow without compromising its quality. In CI, people submit code changes frequently and consistently. Continuous practices help make reliable releases, get regular user feedback, automate tasks and improve team communication.

Deployment pipeline is a vital part of CI. It automates the process of getting the software from the development phase to the release phase. Engineers submit code to the Version Control System (VCS), which triggers different levels of tests: unit, integration, system and acceptance. If any level fails, the developers submits new code, and the tests are run again from the beginning. If all tests are passed, the software is ready for release.

Deployment pipeline is made up of a variety of tools. They are VCS, code management, building tools, CI servers, testing tools, configuration and provisioning, delivery and deployment tools. Not every tool is compulsory, and the final pipeline depends on the organization and project.

3. RESEARCH METHODOLOGY

This chapter discusses the methodology used in the thesis. First, Section 3.1 focuses on the activities involved in a Human-Centered approach to design. Next, Section 3.2 lists a variety of user research methods. Then, Section 3.3 explains the reason behind choosing survey as a user research method. Afterwards, Section 3.4 describes the methods of qualitative data analysis. Finally, Section 3.5 summarizes the methodology steps. The last Section 3.6 provides a chapter summary.

3.1 Human-centered approach

Before describing the methodology for the thesis, it is important to discuss the phases of the project, so that the methodology can be defined accordingly.

The development cycle of this project is using HCD approach. This approach uses four main activities: (i) understanding and specification of context of use; (ii) specification of user requirements; (iii) ideation of design solution that meet user requirements; and finally (iv) evaluation of the design against user requirements [28].

Based on these activities, a two-phase methodology can be utilized to achieve the desired results. First, user research needs to be conducted to specify context of use and user requirements. Second, evaluation involving the users will shed the light whether the established requirements have been met.

One should note the scope of the thesis. The described HCD approach can be done in iterations, but there's only one iteration for this project. Additionally, the evaluation also falls beyond the scope of the thesis, because the final roadmap's implementation time frame is outside the thesis schedule. However, the possibility of evaluation is addressed later in the final chapter of the thesis in Section 6.2.

3.2 User research methods

The validity of user research can be undermined by the bias expressed by users, both conscious and unconscious. For example, participants of the study may give socially acceptable answers to the researcher's question. The answers can also vary based on a person's internal factors like values, habits and personal norms [29]. Based on these

factors, two different approaches can be utilized in user research: (i) investigating user insights that they are conscious about; (ii) investigating user insights that they are **not** conscious about [28].

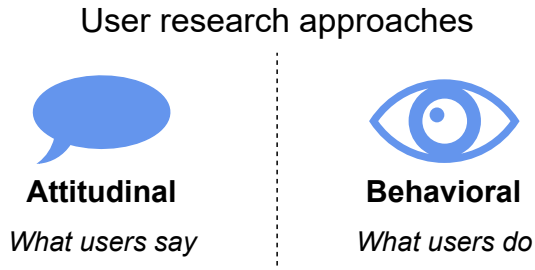

Figure 3.1. *Two approaches to user research [30].*

Taking into account the above-mentioned approaches, user research methods can be split in two categories: attitudinal and behavioral (Fig. 3.1). In attitudinal methods, the research emphasis lies in user-reported data, i.e., these methods gather participant's opinions by communicating with them. Attitudinal methods are discussed in Subsection 3.2.1. In behavioral methods, on the contrary, researchers investigate what people do by observing them to negate the unconscious bias [30]. They are described in Subsection 3.2.2.

3.2.1 Attitudinal user research

Attitudinal research methods rely on participant-reported data. They collect data like person's attitudes, beliefs, opinions, motivations. Such research usually aim at understanding user's stated beliefs [30].

Even though attitudinal methods produce valuable results, a researcher should interpret them with caution. The respondents may give socially-acceptable answers, or not fully understand their needs or perceptions [30]. Because of it, the user's expectations may not align with the final product. Still, these methods can uncover a participant's mental model, i.e., the thought process representing how the person views the relationships between different concepts.

The most common attitudinal methods are summarized in Table 3.1 [28] [30]. The list does not cover all possible methods, and instead focuses on the ones frequently described in research literature.

3.2.2 Behavioral user research

Contrary to attitudinal research, behavioral methods focus on observing the user. They collect data on how the participant interacts with the system [30].

Table 3.1. *Common attitudinal user research methods.*

Method	Description
Interview	Interviewing a participant provides rich information about their own opinion on the subject. There are different types of interviews, each suitable for a particular goal. E.g., individual interviews provide insight on a sensitive topic, and unstructured interviews are suitable when uncovering less obvious subjects. It is important to note that interviews provide qualitative data that can be difficult to analyze [31].
Survey	Surveys and questionnaires are suitable when a large sample size of data is required. Scale-based questions are easy to visualize, and thus provide representative data quickly. However, free-form questions via surveys can yield limited answers [31].
Focus group	Focus groups interviews usually require less time per person compared to regular interviews. The discussion in a focus group can invoke synergy among the participants that can in result provide data that wouldn't be otherwise available. On the other hand, a focus group might become dominated by a leading participant who "hijacks" the discussion [10].
Verbal protocol	In a verbal protocol, the participants verbalize their thoughts, ensuring that the thinking process is recorded. This method, however, is usually coupled with other techniques, especially observational ones [32].
Conjoint technique	In a conjoint analysis, a participant is presented different product features or attributes and tasked to rate them. Such technique shows how customers value individual features that make up the product. However, if the participant is presented too many options, they become unable to properly rate them and instead resort to simplified options [33].
Wants & needs analysis	Wants and needs analysis involves brainstorming sessions with a participant of their wants and needs in a product and prioritizing them afterwards. This approach is useful for ideation, but at the same time, people might not always know what they really want, resulting in too little options [31].
Card sorting	In card sorting, the product features are written onto cards that the participant is tasked to sort into meaningful groups. With this technique, it is possible to gain insight on how the user believes the product operates. The method, however, is narrow focused, because it only produces a user's conceptual model [31].

Continued on the next page

Group task analysis	In a group task analysis, the participants are collaborating on figuring out the steps on how to reach a particular goal. The technique sheds light on possible restrictions the user can encounter, their preferences and behavior. However, to utilize this method, preliminary information should be gathered beforehand to learn about the users and formulate the tasks they can perform [34].
Diary study	For probes and diary studies, participants are given tools to document their interaction with a product. The technique gathers detailed data on how the user interacts and what their feeling and experiences are towards and with the product. One should note that the gathered data is difficult to analyze, and this method requires a prototype to be available beforehand [35].

The advantage of behavioral user research is the possibility to explore the real way people use the service or a product. There is no unconscious bias that is present in attitudinal methods. They allow seeing people's reaction and feedback in-action and therefore collect the most reliable data. At the same time, behavioral methods don't tell user's beliefs, and understanding their thought process is much harder [30].

The summary of main behavioral methods is presented in Table 3.2 [28] [30]. As stated before, the table focuses on the most common ones.

3.3 Survey as a research method

Before proceeding to the implementation, context of use and user requirements must be set via conducting user research. Because this project is tasked by the company based on the developers' requests, observing the users is not required, since the preliminary feedback displays formed opinions and desires on how the project should be implemented. That means the attitudinal group of methods is preferred.

Even though only a single department at the company will utilize the new CI workflow, people with different background and job positions will work with it. Developers, test engineers and managers have diverse opinions on how CI should look like, and it is preferable to reveal the needs of as many users as possible. So, for this project, it is best to gather a large data sample. Surveys are most suitable for such criteria [38], while the rest of the methods presented in Table 3.1 are more time-consuming.

Additionally, the conjoint technique method requires at least an initial prototype to be available, and the focus group method works best with people interacting on-site as that provides deeper answers [39], which is not advisable during the COVID-19 pandemic. Surveys, on the other hand, can be organized online without prototypes.

Furthermore, surveys are more preferable since there is no need to schedule them, unlike, e.g., interviews. Generally, a developer's schedule is busy, and booking a meeting should

Table 3.2. *Common behavioral user research methods.*

Method	**Description**
Observations	Observations are used to record user's behavior. They reveal illogical and natural behavior, highlight unconscious bias and measure product use time. Observations also showcase the behaviors that the user can't describe. At the same time, being observed can impact person's natural behavior, while natural environment of the observation can make analysis complex due to the lack of an expert control [33].
Documentation study	In this method, document analysis is performed, which may include standards, regulations and rules. The method is often used paired with other techniques. It helps understands values of a specific group and provide possible constraints and procedures within predefined context. This method's benefit is being unbiased, however, there can be errors in the documents, or they can be out of context, providing no useful data [31].
Video ethnography	The method involves video recording of the user's behavior in a natural context. The method is used as a part of the process of user research instead of independent research. Video ethnography showcases user's customs and routines and reveals unexpected behaviors. Additionally, it helps recognize foreign customs. The disadvantage of the method is that the interpretation of video data greatly relies on the researcher and therefore can be biased. [36].
Shadowing	In the shadowing technique, the researcher follows the user over a period of time and documents their behavior. The method reveals what the users actually do and helps the researchers understand the reason behind such behaviors. At the same time, the method can be deemed unethical [37].
User testing	In user or usability testing, the participant needs to perform a series of tasks. The method reveals user's way of achieving a specific goal, shows what actions they make during the process, how they approach a task and what constraints they face. However, the recruitment and the testing process can be time-consuming [33].
Emphatic design	In this method, the researcher tries to gain empathy for the user via observation, role playing and other techniques. This allows for input about participant's feelings about the surroundings in different context. However, when feeling empathetic towards a participant, the researcher can become biased [35]
Culture-focused research	This method provides demographic information via census-taking that is useful when designing for an international community. It is important to notice that a researcher can unintentionally misinterpret data when working with a foreign culture [36].

be done well in advance.

3.4 Qualitative data analysis

Collecting purely quantitative data via survey will provide limited answers, while asking only open-ended questions to collect qualitative data may result is people giving very short answers or never finishing the survey altogether [31]. A combination of both quantitative and qualitative data collection can be a solution.

Qualitative data, however, requires additional processing. For this, several approaches are available [40]. Below are the most frequently used ones.

Thematic content analysis is one of the most common approaches for analysis of qualitative data. Its aim is to find common themes or patterns across data, label them throughout the whole text and find dependencies or a common narrative [40].

Narrative analysis is used to analyze stories. A researcher looks for insights in each gathered story, compares and interprets them. Then, he or she creates a new narrative that connects the stories in a new way, highlighting important aspects of previous narratives [40].

A deductive approach is helpful when a researcher already has existing theories. Hypothesis or themes are determined before the analysis starts, and then are imposed onto the data. So, the researcher maps existing categories with the material. This analysis, however, does not provide new understanding or nuanced findings [40].

For the thesis, the gathered survey data should be analyzed for common key ideas. This way, the survey can reveal what are the most common areas in CI that require improvement. No narrative data will be present, and no predetermined hypotheses are available. For this context, thematic analysis is the most suitable approach to analyzing survey data.

3.5 Final methodology steps

The final methodology includes a survey that gathers quantitative and qualitative data. The data is analyzed via thematic matrix. The methodology steps are presented in Fig. 3.2.

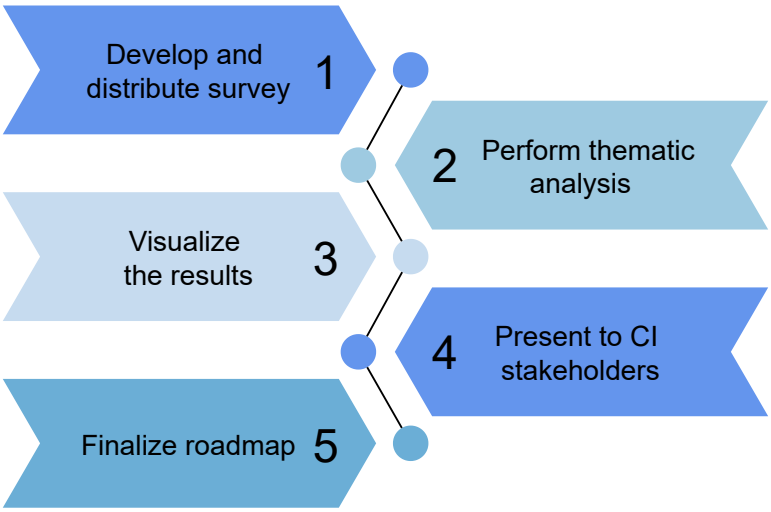

Figure 3.2. *Steps of the final thesis methodology.*

First, the author develops the survey. The survey covers topics such as participant background, general attitude towards CI, opinions about its usability, experienced issues when working with CI and possible new functionality ideas. After completing and testing the survey, it is distributed among the employees of the department via email.

Second, the gathered qualitative data is analyzed with a thematic matrix. The author finds common themes in the responses to determine the most requested functionalities and frequent experienced issues.

Third, the author visualizes the results to help highlight areas of concern. Additionally, visual data is easier to present to stakeholders. The tool to plot the graphs is Python's library Matplotlib [41]. The tool is chosen due to the author's familiarity with it and the ability to plot complex graphs.

Fourth, the results are presented to the department's CI team. The author holds meetings to discuss what areas of CI should be improved and how.

Fifth, the author produces the CI improvement roadmap based on the held meetings. The roadmap includes tasks aimed at improving the usability of CI. This step concludes the thesis work and fulfills the final research objective.

3.6 Chapter summary

HCD approach has four main activities: specifying context of use, specifying user requirements, ideation of solution to meet user requirements, and evaluation against user requirements. Thesis' scope involves the first three activities. The last activity, evaluation, is planned as a future work. Context of use and user requirements are specified with user research.

There are two user study approaches. Attitudinal user research investigates self-reported data, while behavioral research analyzes user observation data. Thesis requires data reported by the employees, so attitudinal research is preferred. There are many attitudinal methods. Among them, an online survey is chosen, because it is suitable for a large data sample. Additionally, it can collect both qualitative and quantitative data.

Qualitative data requires processing. Thematic analysis is most suitable, because it highlights common ideas. Thematic matrix can reveal most frequent requests, thus revealing their urgency.

The final methodology is the following: (i) develop and distribute the survey; (ii) perform thematic analysis; (iii) visualize the results; (iv) present to CI stakeholders; (v) finalize the roadmap.

4. PROBLEM ANALYSIS VIA SURVEY

This chapter features the implementation of the survey as means of user research. Section 4.1 explains the state of the current CI tools and workflow. Next, Section 4.2 describes the survey, listing the questions and their significance. Section 4.3 discusses stages of analyzing the survey. Finally, Section 4.4 shows the results of the survey. At the end of the chapter, a summary is provided in Section 4.5.

4.1 Current state

Before describing the survey and the process of its creation, it is important to explain the tools utilized in department's CI implementation, as well as the workflow. The survey questions feature the names of the tools and references the workflow processes, so understanding the role they play should be helpful for the reader.

The next subsection lists the main tools used in department's CI. Afterwards, Subsection 4.1.2 explains how these tools interact with each other to create a CI workflow.

4.1.1 Tools

As a part of the testing workflow, the company's System-on-Chip Software department uses the following tools: Gerrit, Zuul, Jenkins, Jenkins Blue Ocean, Robot Framework, and Gtest.

Gerrit [42] is a code review tool and a Version Control System (VCS) built on Git [43], a Source Code Management (SCM) tool. Gerrit provides a web platform for developers to discuss and review code. Git is integrated into Gerrit, and it allows easy tracking of code changes and numerous parallel workflows in a form of branches.

Zuul [44], a project gating system, works closely with Gerrit in the department's setup. Zuul provides several pipelines – workflow processes – that run specific jobs for given projects. For example, when new code is uploaded to Gerrit, Zuul triggers a pipeline associated with this event, the pipeline runs required jobs, and then Zuul reports the result. The main purpose of Zuul is to minimize the possibility of introducing software regression with newly submitted code.

Jenkins [26] is an automation server that facilitates CI/CD by automating building, testing and deployment processes. In the current Jenkins configuration for the department, Jenkins scripts build software images for the IPs, checks them with test suites implemented in Robot Framework, and generates test result reports.

Blue Ocean [45] is a plugin for Jenkins that visualizes the CI processes. It shows pipeline status in an accessible and intuitive way, provides a UI for a pipeline editor, and pinpoints the location in the logs where issues arise. Blue Ocean's goal is to reduce visual clutter and enhance developer's productivity. Its use in the workflow is optional, but it can greatly ease the exception handling process.

Robot Framework [46] is a test automation framework with an emphasis on human-readable keywords and tabular syntax. It is closely integrated with Jenkins, which allows for visualization of test status in a form of graphs that display test result tendency.

Google Test, also known as Gtest [47], is a unit testing framework. Unit testing is the lowest level of testing and involves checking the smallest part of software independently. Google Test's main advantages are independence and repeatability, as the framework runs tests isolated.

4.1.2 Workflow

The current testing workflow for IP repositories is as follows. First, a software developer commits new code to the VCS Gerrit. Next, a CI/CD gating system Zuul triggers jobs in Jenkins, an automation server, in the "check" pipeline. A check pipeline is dedicated to test new changes with unit tests. They test individual units of source code. Jenkins jobs feature these unit tests implemented by test engineers in Gtest environment. If a test pass rate is below the determined threshold, Jenkins jobs fail. Then, Zuul posts status messages in Gerrit. The developer is notified about a failed status and commits a fix until the tests pass. At this point, other developer(s) should do a code review of the change, where they read and assess the code for quality assurance and suggest improvements from their perspective.

Next, once Jenkins jobs are successful, Zuul triggers Jenkins jobs in the "gate" pipeline. A gate pipeline prevents changes that might introduce regression from merging, and involves system component tests that are usually implemented with Robot Framework. System component tests check separate components of the software isolated from other components. If the test pass rate is below a threshold, then the developer needs to update the change, and a check pipeline is triggered again. If the tests are successful and the code review is positive, the change is merged to a master branch in a Gerrit repository. Finally, Zuul triggers an optional "post" pipeline. A post pipeline updates the documentation.

4.2 Survey description

In order to gather user's feedback on the usability of the current CI workflow, a questionnaire is conducted. The main focus of the questionnaire is to evaluate whether the available CI tools are easy to use, what issues the users experience with them, and what functionality the users would like to see implemented into the CI workflow.

The questionnaire is aimed at developers working in the company's System-on-Chip Software department. It was implemented in Microsoft Forms and was available for three weeks overall. The survey was sent to 187 employees via email, among whom 56 submitted the answers. A movie ticket lottery was organized among the participants as an incentive to gather as many responses as possible.

The questionnaire went through several development iterations. First, the survey draft was sent to a limited number of the author's coworkers to receive preliminary feedback. Second, after addressing the reviews, a new version was implemented in Microsoft Forms. This version received a second round of reviews where the test participants had to fill in the questions instead of simply reading them. Following the new feedback, a final version was produced.

The final survey has 19 scale-based questions and 3 open-ended questions. Additionally, the questionnaire contains 3 background questions. The questions and the reasoning behind them is available below. The questions are mandatory for respondents to answer, except for the last free-form question. First, the questionnaire starts with the background questions. For the ease of further reference, these questions are given a code BQ (Background question).

BQ1: **What better describes your level of experience?** This is a multiple-choice question designed to evaluate the level of expertise of the participant. They can select only one option. The following choices are presented:

- 0 to 2 years
- 2 to 5 years
- 5 to 10 years
- 10+ years

BQ2: **What type of development do you do?** This multiple-answer question accepts several choices and aims to understand the area of work the participant is focused on. This background helps evaluate how familiar they are with certain CI products. The choices available for the respondents are presented below. An option to specify the answer manually is also included:

- Modelling

- Software testing
- Software design
- Kernel design
- Other [please specify]

BQ3: **What CI products do you use?** Depending on the work the participant does, they may require different CI tools and therefore are only able to provide feedback specifically on these mentioned products. The respondents can choose multiple answers aa well as specify their own:

- Jenkins
- Jenkins Blue Ocean
- Robot framework
- Gerrit Zuul pipeline
- Other [please specify]

Afterwards, the participant navigates to the next page, where he or she needs to answer Likert scale questions. Likert scale requires the user to rate a statement. The following options are presented in the rating scale: (i) strongly disagree; (ii) disagree; (iii) neutral; (iv) agree; (v) strongly agree; (vi) N/A (not applicable). These options provide enough flexibility, and the N/A option is useful if the respondent does not have experience with the discussed product. Likert scale is easier to analyze in comparison to open-ended questions, as it produces quantitative data. Additionally, rating scale surveys are familiar to many employees, allowing for a decreased survey completion time.

For the ease of reference, the questions below are assigned a code SBQ (Scale-based question).

SBQ1: **I think that CI benefits my development workflow.** The statement reveals the general attitude of developers towards CI.

SBQ2: **I find the test results quickly and easily.** The ability to quickly find test results is valuable, because the changes are submitted continuously and frequently in CI.

SBQ3: **I find the reason for test failure easily.** Finding the reason for the test failure quickly accelerates the development process.

SBQ4: **I can easily trigger tests from the version control system (Gitlab, Gerrit).** Triggering the tests is required to assess the changes before submitting them to the master branch of the repository, i.e., the place where the most stable version of the code is stored. Therefore, the process of test launch should be easy and straight-forward.

SBQ5: **I can easily restart specific tests if needed.** In case only specific tests fail when testing a change, it might not be required to run the whole test set. To reduce the test running time, developers should have the ability to trigger specific tests.

SBQ6: **I think that the Robot test reports are well-structured.** Robot Framework generates a structured report that aims to showcase the data in a user-friendly way, for example, by providing error messages at the top of the report to draw the attention. Because many developers need to use this framework, it is important to assess the overall feedback of the tool.

SBQ7: **I think that the error messages in the logs and reports are clear.** Error messages are vital to finding out the reason behind test failure. Insufficient logs make fixing the source code a long and frustrating procedure.

SBQ8: **I think that the logs provide sufficient information (Jenkins console output, HTML logs, etc.).** Omitting information in logs complicates test failure analysis.

SBQ9: **I think that the Jenkins job names are easy to understand.** Jenkins job naming conventions may be difficult to understand, creating a high entry threshold to utilizing Jenkins and making it difficult to find needed jobs.

The following Likert scale questions were separated to the next page in order to make the survey less overwhelming.

SBQ10: **I think that the logs are overly detailed, and they should be shortened.** On the contrary to one of the previous statements, there might be an issue of overly detailed logs instead of insufficient logs. In case the logs provide too much information, finding the required data may become complicated and consume additional time.

SBQ11: **I think that Zuul posts too many comments in Gerrit.** Zuul tool posts messages in VCS Gerrit, stating the progress of tests. By default, the users receive email notification about each Zuul message. Too many messages from the tool may result in developers not paying attention to them at all, thus missing important ones.

SBQ12: **I feel the need to use additional software to process logs (e.g. parsing scripts).** This statement evaluates whether it is comfortable to work with logs as they are, or is additional software required to make the log analysis easier.

SBQ13: **I feel the need for instructions when using CI.** The need for instructions may point to the CI workflow lacking usability, making it hard to follow without a manual. Additionally, it can also mean that the users can't find existing manuals, or instructions don't provide enough information.

SBQ14: **I feel the need to learn the test framework in order to run tests.** Having to

learn the test framework before running tests results in a how entry threshold, slowing down CI.

SBQ15: **I would like to create new Jenkins jobs on my own instead of asking the CI team.** There might be a demand among the developers to make Jenkins jobs themselves. Allowing them to do that might speed up CI, as the developers wouldn't need to wait for the CI team to respond to their request.

SBQ16: **I make new Jenkins jobs myself, and I find the process complicated and time-consuming.** The statement reveals whether there's a demand to automate or simplify the Jenkins job creation process.

SBQ17: **I feel the need to have ready-made templates in order to make the creation of new Jenkins jobs easier.** Providing templates for developers to create their own Jenkins jobs speeds up CI, gives developers autonomy, and unifies the scripts among different teams.

SBQ18: **I would like the CI workflow to be similar across all teams.** The statement can show whether is it better to allow the teams to implement their own version of CI or to make it more universal, so that the users can easily explore the project status of other teams.

SBQ19: **I would like new CI tools to be introduced to the workflow.** The statement can reveal the demand for new tools to be implemented to the CI or, on the opposite, show that the developers are content with the current tool setup.

Finally, on the last page of the survey, the user needs to answer two required open-ended questions and optionally fill in one field dedicated for additional feedback. In this thesis, these questions are referred to with the code OEQ (Open-ended question).

OEQ1: **What functionality would you like to see added to the current CI workflow?** The only way to get ideas for new functionalities via survey, if any functionalities are requested by the developers, is through a free-form question instead of a scale or multiple choice question.

OEQ2: **What issues do you experience when using the current CI?** This question can help gather data on what needs to be improved in the current CI workflow before any new features are implemented.

OEQ3: **Please share any other feedback.** The non-mandatory form welcomes explanation on the scaled questions and also allows for additional comments.

The survey participant can submit the answers afterwards. It is not possible to record partially-completed surveys, so only a fully-answered questionnaire is submitted in the end. The expected completion time is 5-10 minutes. The estimation is based on the Microsoft Forms' suggestion as well as initial test runs.

4.3 Analysis steps

The questionnaire was open for three weeks. Overall, 56 responses were collected. The median survey completion time is 8 minutes, which corresponds to the estimation.

The survey is analyzed in these stages: (i) screening for invalid responses and discarding of them; (ii) sorting of responses to open-ended questions; (iii) construction of a thematic matrix.

For the first stage, the aim is to look for the answers with extremely low completion time, suspicious answering patterns and incorrect organization department, i.e., any department other than the company's System-on-Chip Software. The screening resulted into discarding of one response, where the respondent completed the survey under 2 minutes instead of expected 5-10 minutes, answered "Neutral" to every scaled question and specified an area of work that did not correspond to the aimed organization department. This step shortened the number of responses to be analyzed to 55.

The second stage revolves around screening of answers to open-ended questions. The aim is to move the answers to the correct questions is case they do not correspond. For example, multiple respondents listed functionalities they wished to see in the field intended for specifying experienced issues, and vice versa. This step is required for the ease of further analysis.

The third stage is the construction of a thematic matrix. In this thematic analysis, the responses are assigned themes in a matrix format to identify common patterns. Additionally, thematic analysis helps identify the frequency of certain topics occurrences, which also aids in drawing the conclusions on the urgency and prevalence of issues. That information will further guide the process of identifying the priority of the tasks to be implemented.

The first open-ended question has a much lower participation rate compared to the second one. Overall, 27 people, or 49.1% of respondents, left meaningful answers, i.e., left a request for a functionality. The second question, however, received meaningful answers from 49 respondents, which means 89.1% of participants reported issues that they encountered. In the end, 19 themes are identified for the first question about requested functionalities and 18 themes for the second question regarding experienced issues. The last question about additional comments received only 13 responses. It isn't included for the thematic analysis, as the answers mostly contain notes that can not be used to improve CI.

4.4 Analysis results

After preparing the data for analysis and creating a thematic matrix, the data is visualized by plotting graphs. The graphs help draw conclusions from the survey responses. The

results are presented in this section.

Subsection 4.4.1 showcases the backgrounds of the participants. Subsection 4.4.2 analyzes the results of Scale-based questions. Finally, Subsection 4.4.3 reveals the responses to Open-ended questions.

4.4.1 Participant background

Judging by answers to BQ1, the majority of participants (58.2%) have at least 10 years of relevant work experience. A significant number of respondents are also junior employees with up to 2 years of experience (18.2%). Employees with 2-5 and 5-10 years of experience were a minority in the survey (14.5% and 9.1% respectively). The experience distribution is showcased in Fig. 4.1.

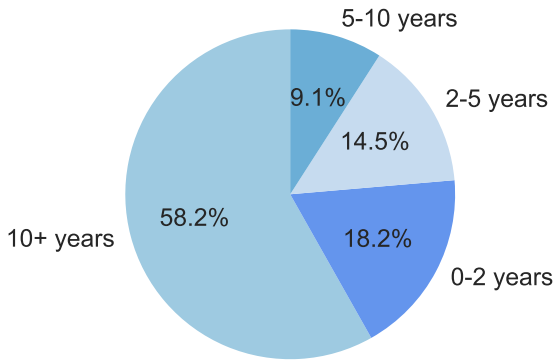

Figure 4.1. *Years of work experience of the survey participants.*

Based on the multiple-choice BQ2, the participants are currently doing mostly software design (69.1%) or software testing (47.3%). Kernel design and modelling were unpopular choices (12.7% and 9.1% respectively). Outside the options presented for BQ2, the participants also indicated additional roles. Overall, based on these extra responses, we can conclude that 10.9% of survey respondents perform engineering tasks other than mentioned, and 7.3% have managing roles (see Fig. 4.2). The additional prompts include the following: "SW [Software] Architecture", "Build system integration", "DevOps", "Release engineer", "Project management & reporting", "Leadership", "CI", "SoC [System-on-Chip] Configuring / coding", "Scrum master and LPO [Local product owner]", and "Management". Even though the questionnaire was originally designed for engineers, it was decided that answers from employees performing management tasks would provide diversity to the response pool. These responses can potentially highlight the differences in the requirements of the participants in regard to their role in the company.

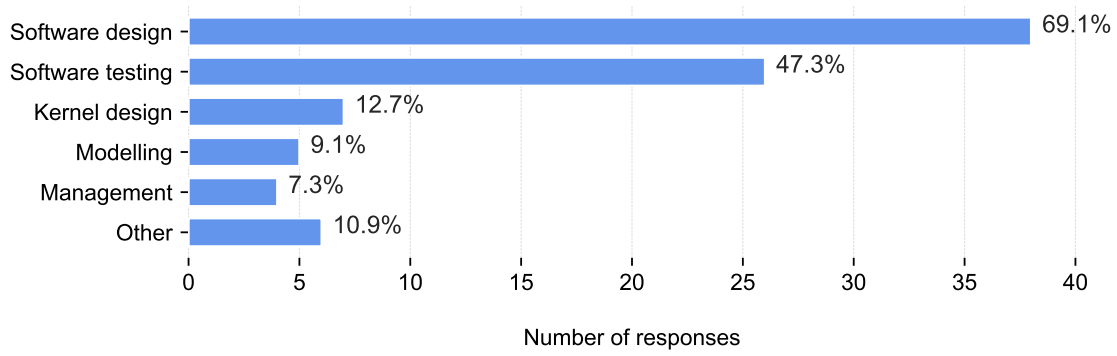

Figure 4.2. *Development fields of the survey participants.*

For BQ3, where people could choose multiple answers, the most widely used products turned out to be Jenkins (used by 89.1% of survey participants), Gerrit Zuul pipeline (76.4%) and Robot Framework (49.1%). The least popular tool is Jenkins Blue Ocean (34.5%). Additionally, 7 participants specified extra tools that they use: Yocto (2 answers), Coop (2 answers), Coverity (2 answers), SonarQube (2 answers), Artifactory, Git CI, Docker, Kubernetes, Power BI, WFT. See Fig. 4.3 for the summary.

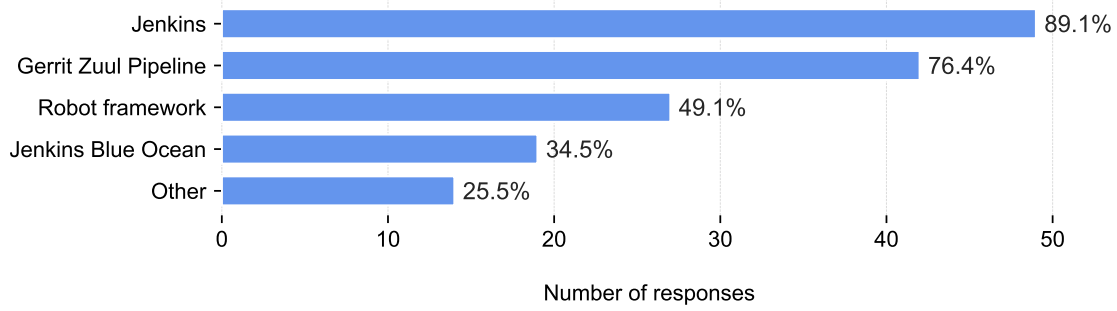

Figure 4.3. *CI tools used by the survey participants*

The extra tools used by the employees can be elaborated further. The Yocto project [48] is a tool for creating custom Linux distributions. These custom builds are used for running the tests in an independent environment. Coop is a tool internally developed at the company designed to visualize the various CI data; a similar role is also played by Power BI [49] developed by Microsoft. Coverity [50] does code analysis and helps catch defects, security risks and bugs early. SonarQube [51] inspects code quality with code static analysis to help developers write cleaner and safer code. Artifactory [52] is a DevOps tool that acts as a repository for artifacts generated by the whole CI/CD workflow. Git CI likely refers to GitLab CI/CD, which is a tool integrated to GitLab's VCS intended for software development with the continuous technologies [53]. Docker is a DevOps product that provides virtualization of packages and their dependencies to allow running them on any machine [54]. Kubernetes [55] is a system that automates deployment and management

of applications, and it plays an integral part in CD. Finally, WFT – Work Flow Tool – is developed internally at the company for build management.

All of these extra tools are either used by a small number of developers, designed for managers, relate more to CD than CI, or aren't related to them at all. Therefore, the utilization of these tools and its improvement are out of scale of this thesis.

4.4.2 Scale-based questions

The scale-based questions revealed that the absolute majority of participants agrees or strongly agrees (89.2% overall) with the statement SBQ1 that CI benefits their development workflow. This review indicates that the developers understand the need for CI, utilize it and see this system as advantageous to their work.

Statements SBQ2, SBQ4, SBQ6, SBQ8, SBQ13, SBQ14, SBQ17 and SBQ18 got favorable feedback. The first half of the statements discuss the everyday usage of tests to check the code quality. Because tests need to be triggered every time a developer pushes code to the Version Control System (VCS), it is important to provide an easily-understandable and convenient way for the said developers to interact with those tests. Based on the above-mentioned SBQ2, SBQ4, SBQ6 and SBQ8, the survey participants feel that tests are easily triggered (60.7% (strongly) agree and 12.5% (strongly) disagree), their results are easy to find (71.4% and 10.8%) and are well-structured (39.3% and 10.8%), and the logs provide sufficient information (53.6% and 17.9%). Meeting these attributes is core for an efficient and effective experience for the users.

The second half of the mentioned favorable SBQs (SBQ13, SBQ14, SBQ17 and SBQ18) highlight the weak points of the current CI workflow. The employees agree that instructions on how to use CI are needed (53.6% in favor and 19.7% against), and they feel obligated to learn the test framework before executing the tests (59.0% and 17.9%). Additionally, 60.8% of the respondents want to see templates that would help with creation of new scripts that run the tests, while only 10.7% do not agree with it. Lastly, 67.9% of participants want the CI workflow to be similar across all teams, while to 3.6% of the respondents disagreed with that statement; none of them chose the "strongly disagree" option in this question, which is the only case in this survey. All of these problems point to the lack of tutorials or instructions, and, in consequence, these issues result in a high entry point to the workflow. The developers have to research the test frameworks and figure out on their own how the workflow operates. To further complicate an already demanding CI workflow state, all the teams have their own implementation of CI with little documentation. These user needs should be taken into account when developing solutions for CI improvement.

However, statements SBQ3, SBQ5, SBQ7, SBQ9, SBQ11, SBQ15, SBQ16 received

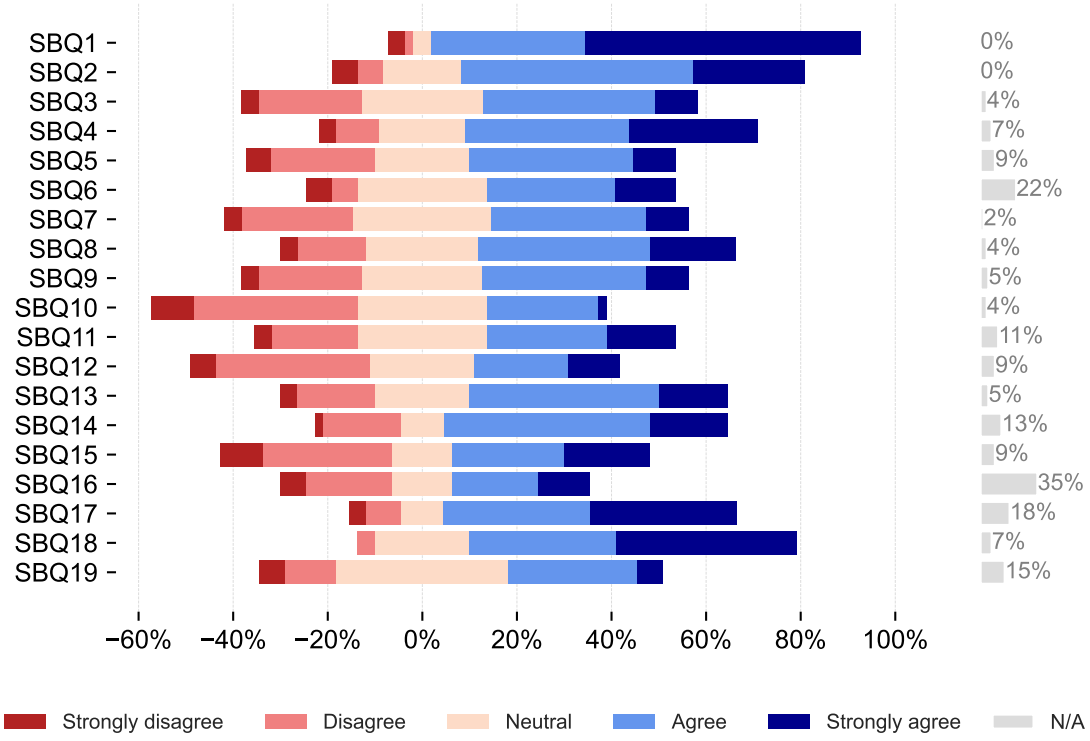

Figure 4.4. *Graph depicting the answers to the scale-based questions.*

mixed opinions from the respondents. The results of these SBQs should be taken with lower priority, as the participants do not have an agreement over them. The employees do not agree over how easy it is to figure out the reason for the test failure (SBQ3, 44.6% positive and 30.4% negative responses), restart specific tests (SBQ5, 42.8% and 26.8%), and whether the error messages are clear (SBQ7, 41.0% and 26.8%). They also have mixed opinion whether the Jenkins job names, i.e., the names of the scripts that run the tests, are easy to understand (SBQ9, 42.8% positive and 25.0% negative responses). The Zuul pipeline posts messages in the Gerrit Version Control System (VCS) related to the test status (SBQ11), and there is a big number of positive, neutral and negative responses to this statement about the message frequency (39.3%, 28.6% and 21.5% respectively). The survey also revealed that a big number of employees would prefer to create Jenkins scripts themselves instead of the CI team (SBQ15, 40.1% positive and 35.4% negative responses); the opinions regarding this statement are still mixed, but there are much more positive responses than anticipated. Finally, for the last mixed-opinion statement SBQ16 (complicated Jenkins jobs creation), a big number of participants replied with "Not applicable" (33.9%), which is the highest number in this survey; the reason for it is understandable: the majority of developers do not create new Jenkins jobs themselves. There is a slight favor towards "agree" option in this SBQ (28.6% in favor and 23.3% against).

The results of these mixed-received SBQs show that there's still room for improvement

regarding test interaction, as well as the creation of new test running scripts for Jenkins.

There is one statement, SBQ19, where most participants felt neutral towards it (37.5%). There is a higher number of people in favor for the new tools to be introduced to CI than opposed to the idea (26.8% against 10.7%). Still, most participants neither agree nor disagree. This is further confirmed by the little participation rate for OEQ1, where only half of the employees who participated in the survey could suggest new functionality ideas.

Finally, most respondents do not agree with the statements SBQ10, SBQ12. Negative answers to these questions indicate that the employees are satisfied with the level of detail of the logs, and do not require additional software to process them. The logs of the process of building and testing the image are very important to the developer, as they are the main source of input when it comes to debugging. Survey participants feeling satisfied with the logs can be interpreted as positive feedback.

4.4.3 Open-ended questions

Both OEQ1 and OEQ2 are mandatory questions. However, not every participant left meaningful answers to them. To elaborate, OEQ1 received 30 prompts that suggested a functionality to be added. The rest of the answers consisted of prompts where people insisted that no new functionality is required, none come to their mind, etc. Overall, 29 people, or 53% of the respondents, did not specify any functionality. That means only 26 people, 47% of respondents, contributed to OEQ1. The differences in the number of respondents (26) and responses (30) is due to some participants leaving more than one answer.

OEQ1: Functionalities

At the end of the thematic analysis, 19 functionalities were uncovered for OEQ1. The graph depicting the themes and the corresponding number of responses is available in Fig. 4.5. The majority of them consisted of just one suggestion; however, these functionalities are very diverse in nature, so it was still important not to combine certain single-response suggestions into one theme.

Participants argue that adding code style check will reduce developer's review time, essentially automating this task. Quality gates would calculate code coverage and prevent the code from being submitted if the code coverage metric is below an established threshold. There is support for code coverage already implemented in the CI, but, as evident by the survey responses, the developers are not aware of it. As for the style check, the functionality can be considered for the roadmap.

A variety of log enhancements are suggested by the respondents to shorten the time

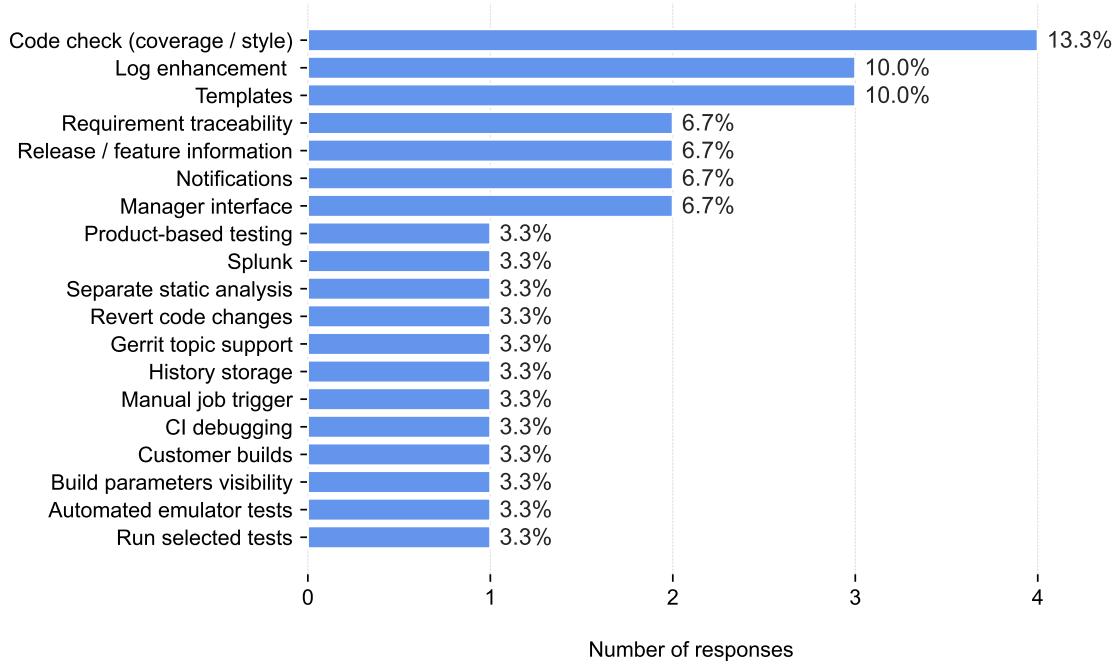

Figure 4.5. *Themes derived from 30 functionalities requested by the participants.*

spent on analyzing and parsing logs. The techniques include pinpointing errors and warning messages, having more detailed logs for test cases, and log filtering via flags. Unfortunately, log filtering isn't supported by the environment, and detailing test case logs does not relate to CI – it falls under the responsibility of testing teams. However, pinpointing errors is possible by parsing the log and posting the messages in the VCS.

Three people support the idea of introducing CI script templates mentioned in SBQ17. The employees highlight that the templates will make CI pipelines similar across the teams, making the maintenance easier, as well as provide best practices and reference for the script developers.

The participants also suggest including test case mapping against development requirements, i.e., requirement traceability. This proposal is indeed helpful, and it can possibly be added to the roadmap.

Getting the information from CI regarding project status, release information and tested features is another suggestion. This information is useful to check what features are left to test and quickly understand the general status of the projects outside the team.

The developers also would like to see notifications sent via email in case the tests are not passing, or regarding the test status overall. This feature is supported, but, judging by this feedback, not every developer knows about it.

Two people with managing background requested a "manager interface", i.e., a tool or a view for displaying test results on all CI levels and generating reports of them. At the same

time, one developer left a counter-argument: they prefer the current setup to work with the highest quality before moving to any other task, making a point that managers can wait for managing dashboards. Because the survey was intended to make developers' voices heard, their requests will be prioritized, and a manager interface is not currently planned for implementation.

Product-based testing is a feature suggestion that did not receive much elaboration. Nevertheless, it has been available in CI for a while now. The fact that a survey participant requested an existing functionality further indicates a problem with knowledge sharing in the company.

Splunk is another tool for data visualization and a variety of other features [56] which is also used by different teams within the company. It is suggested for investigation of issues in tests or CI. The possibility of adding this tool into the workflow can be taken into account.

One suggestion is to separate static analysis from the rest of the tests, as these tools can take up a considerable amount of time. In reality, these tools are often separated into their own Jenkins jobs. Evidently, this is not the case for every team in the department. The capacity for this feature is available, but it looks like the teams either don't know that or do not communicate their needs to the according personnel who set up the static analysis.

The possibility to revert code changes is suggested to be implemented in a form of a web interface. This functionality is intended for easy and quick revert of the latest changes to the code and consequential rerun of the Jenkins job. However, there's no support for this implementation in Jenkins, so this feature cannot be done.

There's a request for support of topics in Gerrit VCS. In Gerrit, a topic is used to group changes together [57]. Topics are useful when one change is dependent on another change. This way, the changes can be tested together. Unfortunately, at the moment, the company's Gerrit configuration does not support topics across all the repositories, even though the need for them is present. This functionality should be taken into consideration.

History storage is another pain point. Currently, the volume storage for Jenkins jobs artifacts is limited, resulting in a short period of storage of history. A limited number of storage days can disrupt the process of analyzing the reason for failed jobs, as the period of analysis can outlast the period of history storage. The possibility of implementing this feature depends on the availability of a budget for new equipment.

One person is suggesting a functionality to manually trigger Jenkins jobs instead of only VCS trigger. This functionality is also available. The respondents might not know about it, or the team might not have it implemented. Either way, it points out the area of improvement for knowledge sharing.

CI debugging is a suggestion to implement a temporary permission to debug test cases in CI environment in case of a failed test. Though useful in theory, the feature is not supported in the Jenkins interface.

One participant left a request to implement builds in CI according to customer requirements, so that any issues that might arise can be caught earlier, as currently the department creates only own test images. However, it will not be possible to implement due to the lack of clear customer requirements and use cases.

Another request is to see the parameters of the builds in CI in an easier way. The survey participant wants to see what is being built and run, which versions, and in what environment. The visibility of the build parameters is helpful, because when any problem arises, it is easier for a developer to repeat the build steps manually after quickly looking at the parameters of the build. This feature is supported by the environment, but it's the responsibility of the teams to enable it.

"Automated emulator tests" suggestion did not have any elaboration to it. Unfortunately, the possibility to reach out to the survey participants was not intended in the survey. Emulator tests are already automated, so this request is implemented, unless the respondent meant something else.

Finally, the last suggestion is the ability to run selected tests that are affected by new code changes. This feature would surely decrease the amount of test run time, accelerating the time of code submission. However, this feature is not CI-specific, but instead, it should be implemented in a test framework by test engineers.

OEQ2: Issues

As for OEQ2, the thematic matrix revealed 17 themes of issues based on 88 responses submitted by the survey participants. This question received much more replies than the previous one. Overall, 89.0% of the respondents (49 participants) indicated that they experience issues when working with CI. The number of submitted responses is almost double the number of respondents with issues, implying that, on average, people have to cope with not just one, but several problems. Fig. 4.6 lists the uncovered themes and their frequency.

The most common issue that people experience is the instability or unreliability of CI. The respondents list the following examples of instability: Jenkins jobs not triggering; failed tests due to external reasons; test framework not functioning correctly; having a lot of CI steps resulting in a dependency chain where an error can create a domino effect of failures. The main message that the survey participants try to communicate is that the current CI implementation often fails while the developer's code is not faulty. The major consequence of unreliability is lost hours and mistrust of CI, which in return motivates the

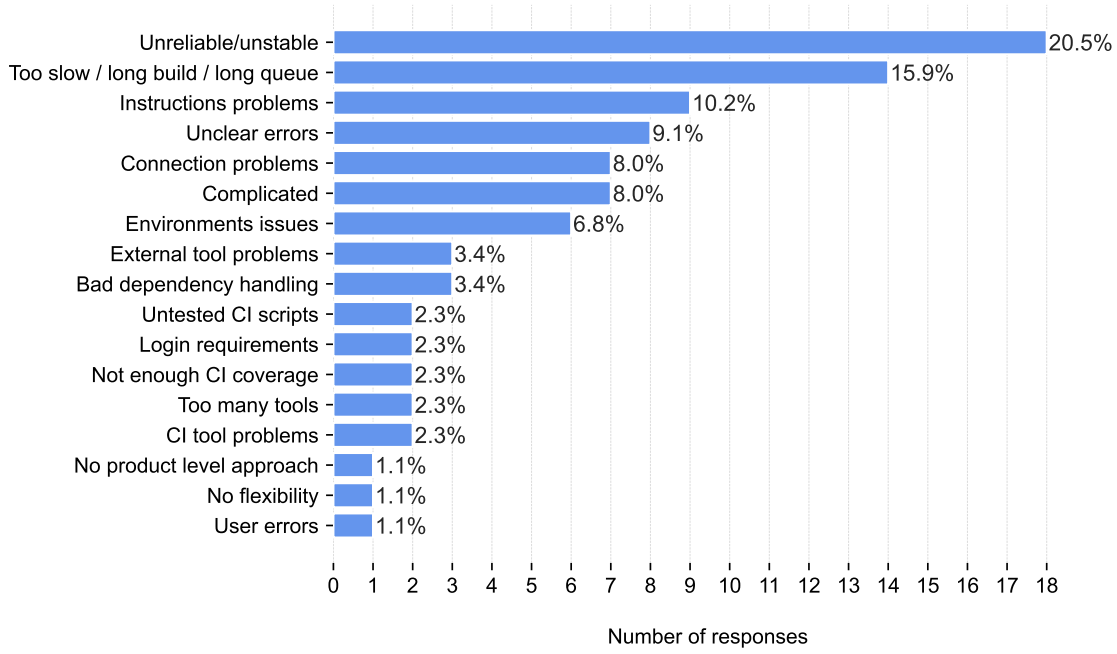

Figure 4.6. *Themes derived from 88 issues reported by the participants.*

developers to skip CI entirely. This issue is complicated to resolve as there are many root causes to it. However, since this is the most common CI problem by a wide margin, it must be prioritized.

The second most common issue is time related. The participants highlight that the images build for too much time and the build queue lasts for too long. As a consequence, following agile practice of committing frequent code becomes inconvenient. Additionally, the submission of urgent changes may be compromised. This problem is also ahead of the next one by a large margin, and it should be prioritized as well.

Next, the respondents accentuate the importance of having documentation. This topic was already brought up in SBQ13, where a majority of participants agreed that they require instructions when using CI. The employees point out that the documentation is scattered, incomplete or absent altogether. Furthermore, the respondents have a demand for training. There are no guides on best code practices, and and the code lacks comments. Therefore, the developers have to spend much more time than required on creating new Jenkins scripts and working with Jenkins in general. As the size of the company grows, the lack of manuals will become an even more alarming issue.

The employees also list unclear error messages as another CI issue. When tests or Jenkins scripts fail, the errors in the log do not clearly indicate the root cause for it, halting the fixing process. One person even mentioned that these messages can be misleading. Developers can spend less time on fixing errors if the CI team focuses more on error handling.

Connection problems are another issue mentioned by the survey participants. People point out that the infrastructure regularly suffers from unstable connection. As a consequence, the tests can take more time than intended or fail altogether. Additionally, connection shut down can restrict access to certain resources. The issue is much wider and goes beyond the CI workflow, and therefore would require help from additional departments that oversee computer networks in the company.

With an equal number of responses as the previous prompt, complicated workflow is the next issue to tackle. A big number of steps and dependencies make the workflow hard to understand, thus the users start requesting instructions and training, as evident from the previous prompts. Furthermore, this many steps make debugging and figuring out the cause of errors, which are already unclear, harder. Additionally, it creates more points of failure, possibly making CI unstable. Coupled with connectivity problems that cause even more unreliability, complicated workflow becomes a major issue.

The participants point out to issues related to the environment. To be precise, they explain that there are not enough environment setups for certain tests or products, and that the CI environment is different compared to the one used in development. The issue should be investigated, however, it might be under the responsibilities of the department that overlooks the lab.

There are several reported issues that are caused by external tools, i.e., not CI-specific, but nonetheless related to it. For example, some of the additional tools listed for BQ3 (see Subsection 4.4.1) that analyze test result data may display the data incorrectly. The personnel responsible for these tools should be notified, and the issue investigated.

The next problem concerns the handling of the dependencies between projects. A lot of code changes are dependent on other changes that are not yet submitted. Unfortunately, dependency handling is incomplete and does not work for every project. Also, two-way dependency is unavailable yet. This issue should be tackled in the future.

Two employees report that often Jenkins jobs fail not because of their own code changes, but due to changes to the CI scripts. The cause of the issue is that CI itself does not have a platform to test the changes in the same capacity as usual developers do. The CI team should do its best to test the changes to the possible degree.

Another complaint is the requirements to log in to CI. In order to access it, a user should first log in via another platform. Even though this requirements might seem irritable, it is required for security reasons.

Next, there is an issue regarding incomplete CI coverage. For example, as the respondents explain, not all repositories in the VCS have CI. The support for it is a work in progress.

CI involving too many tools is the next issue reported by the survey participants. They reason that the role of certain tools that perform quality control is questionable, and they occupy the time that could be spent on development. However, one should note that quality control is a vital part of quality assurance. If that time is spent on development, the quality of resulting code would drop. A better solution might be to communicate to the employees the importance of retaining these tools.

The next issue is related to specific tools involved in CI. The respondents point out some errors in Robot Framework and Jenkins Blue Ocean. However, reported issues are vaguely explained. Unfortunately, there isn't a way to connect to the said respondents. What can be done instead is encouraging the people to contact the support via other means whener such issues arise.

At the bottom of the list are the issues reported just once in the whole survey. One of them is the lack of product-level approach in the CI. Instead, the department relies on IP-level approach. The decision for the approach is not made by the CI team. However, Jenkins jobs for specific products are still available.

Then, there's a complaint about CI not having enough flexibility. The respondent provides an argument that with flexibility, the overall development and testing efforts will be reduced. However, they did not explain what CI areas specifically require flexibility and in what form.

Finally, the last prompt highlights the existence of user errors. Unfortunately, the participant did not provide any more details.

4.5 Chapter summary

The department utilizes Gerrit as a VCS, Zuul as a project gating system, Jenkins for build and test automation, Jenkins Blue Ocean for visualization of CI processes, Robot framework for system tests and Gtest for unit tests.

The current workflow consists of the following. A developer submits code to Gerrit. Zuul triggers Jenkins to make a build and run tests on it. The tests are designed in Robot framework and Gtest. If the build or the tests fails, the developer submits new code, and Zuul triggers Jenkins again until all tests pass.

The Microsoft Forms survey consists of 3 background questions, 19 scale-based questions, and 3 open-ended questions. The last ones gather feedback on experienced issues, functionality requests and any other possible comments. Scale-based questions gather opinions about used tools and the workflow. The survey is distributed to 187 employees via email, among whom 56 submit responses.

The responses are screened, sorted and analyzed via thematic matrix. One response

is discarded. If the respondents switch around the answers to open-ended questions, the responses are sorted correctly. Overall, 49% of respondents left a functionality suggestion, and 89% reported issues. The analysis revealed 19 functionality themes 18 issues themes.

Majority of participants have at least 10 year of work experience. Most are either software designers or testers. Most often used tools are Jenkins and Zuul.

The absolute majority of respondents believe CI is benefitting their development. They feel that the tests are easily triggered, their results are easy to find, test reports are well-structured, the logs provide sufficient information and don't require additional processing. There is also a big number of employees who wish to write Jenkins scripts themselves instead of appointed people.

There are weak points to CI as well. Respondents think that CI instructions and Jenkins templates need to be added, they feel obligated to learn the test framework, and that the CI workflow should become consistent across teams.

Some statements received mixed responses. The employees do not agree over the complexity of figuring out test failure reasons, restarting specific tests and creating Jenkins scripts, whether the error messages and Jenkins job names are clear, and the desired frequency of Zuul messages.

Respondents are mostly neutral towards the idea of new tools. There is a slightly bigger number of people disliking the idea than agreeing with it.

Most common functionality requests are code coverage and style check, log enhancement and Jenkins templates. Most experienced issues are unreliability of CI, its slowness, unclear or lack of instructions, vague error messages, unstable connection and complicated workflow.

5. RESULTS AND DISCUSSION

This chapter features the final results of the thesis. Section 5.1 tells about the discussions held with CI stakeholders. Section 5.2 reveals unexpected findings from the survey. Finally, Section 5.3 discusses the final CI improvement roadmap. At last, there is a chapter summary in the concluding Section 5.4.

5.1 Internal discussion

After conducting the thematic analysis, the author held four internal sessions with the CI team members, an hour and a half each. The goal of those meetings was to discuss the results of the analysis. In the end, the sessions intend to fulfill the following tasks: (i) notify the CI team about the general attitude of the employees regarding their work; (ii) assess the features suggested by the survey participants; (iii) address the reported issues; (iv) choose tasks to be added to the roadmap; (v) determine the priority of the tasks and assign a sprint to them; (vi) allocate assignees to complete the tasks.

In the first meeting, the author presented visualized survey data. The team got introduced to the background of the participants, their answers to the scale-based questions, and the first themes of OEQ1. The participants' background was important to understand, so that the team would know who they are developing the Jenkins pipeline for. The answers to the scale-based questions gave a general understanding of the participants' attitude towards CI, which provided feedback to the team's efforts.

For the following meeting, the CI team and the author focused on the rest of the themes of OEQ1 as well as the first half of the themes for OEQ2. With the OEQ1 themes finished, the team grasped the idea of users' requirements.

In the third meeting, the attendees finished analyzing the last data, i.e. the second half of the themes for OEQ2. The team realized the drawbacks of the current state of CI, and figured the area of improvement.

Finally, for the last meeting, the author compiled a summary of the above-mentioned three sessions. The author selected a list of items derived from the previous analysis to be added to the roadmap. The team determined the status of the items, chose a priority for them, and created the final roadmap list.

5.2 Findings

Beside the resulting data described in Section 4.4, i.e., general attitude towards CI, requested functionalities and reported issues, the survey revealed additional unexpected data. In this section, the author interprets the answers that the respondents gave and pinpoints relations between people's background and their responses to the survey.

In general, there were much less functionality suggestions compared to reported problems. As the answers to SBQ19 show, the respondents are also mostly neutral towards the idea of adding new tools to the workflow. With that said, one can deduce that the employees are not striving towards new tools or features. As several comments showed, at least a couple of people prefer the issues to be fixed first before the implementation of new functionalities. It is likely that the needs for functionalities and tools are satisfied, and so, the CI team can indeed focus more on improving the existing setup. Still, if necessary, new tools can be introduced, as the employees are not opposed to the idea.

The author with the CI team came to a conclusion that a lot of reported issues were out of their responsibility, or they did not have the agency to implement certain suggestions. However, we can use the data as evidence to push the agenda to other departments, motivating them to take action.

Another revealed point is that the communication channel between the CI team and the developers has room for improvement. Some reported problems were small and team-specific, and, ideally, should have been reported not via organized survey, but by contacting the CI staff directly. Furthermore, there were issues that are direct responsibility of the team, not CI people. It shows that the CI staff did not communicate properly the responsibility delegation, and doesn't have an easily-approachable channel where the teams can ask for help.

The lack of a proper communication channel also results in a lack of knowledge sharing. For example, as previously discussed in 4.4.3, there were a couple of requested functionalities that were already implemented (e.g., receiving notifications about the test results). One can conclude that the respondents did not know that a functionality existed. Additionally, as Fig. 4.3 shows, there are tools that aren't often used, which also can be explained by the lack of awareness of the tool. Furthermore, several people believe there are too many quality control tools. Quality control, however, is mandatory; communicating the message of importance of quality is essential, but it seems that it was not done correctly. Awareness can be raised with trainings and documentation, which is exactly what the respondents requested to add.

When the author raised the issue of lacking documentation, the CI team commented that there aren't enough personnel to write it. If the documentation tasks are delegated to the current employees, the implementation or fixing tasks have to be postponed. Which

points out to the staff shortage, a more global, deep-rooted issue that the author and the CI people consequentially reported to the managers.

As for the background of the respondents, in general, it did not have much of an effect on the answers. The employees with the same level of expertise of field or work did not have a consensus on any question. However, there were slight inclinations towards certain answers or patterns.

For example, several respondents with an engineering background felt like managers have a priority when it comes to new feature implementation as well as more tools. At the same time, managers had complaints about the tools they use. However, there were too little respondents with managing background, so there wasn't enough data to analyze their needs.

For SBQ19, a question about introduction of new tools to the workflow, respondents gave slightly different answers depending on their years of experience. The people who didn't want new tools were senior or middle employees. Junior employees, on the other hand, either agreed with the statement or felt neutral towards it; none disagreed. However, it is incorrect to say that senior or middle employees oppose the idea; they still majorly went with a neutral option.

Interestingly, some people who indicated that they create Jenkins jobs themselves in SBQ16 did not list testing or CI as their responsibility in BQ2. Unless the respondents forgot to tick that option or answered incorrectly, this further confirms that there is an unclear delegation of responsibilities. Once again, knowledge sharing needs to be improved.

Finally, the last background finding relates to the instruction request for OEQ1. The answers to this question are from employees with diverse level of experience, from juniors to seniors. It indicates that regardless whether the person is new to CI or not, they still require manuals, as the workflow is designed in a complicated way.

Even still, with all the remarks and issues, the respondents consider CI to be beneficial to their workflow, as evident from SBQ1.

5.3 Implications

Overall, the sessions described in Section 5.1 yielded 30 possible roadmap items that should be logged into the issue tracking tool. However, the investigation revealed that several items were already implemented, added to the roadmap, in progress or invalid. In the end, 5 items were deemed done, 11 items were already planned, 3 items were in development, and 7 items needed to be scheduled for development. Additionally, 4 items needed to be assigned to employees outside the CI team, because these tasks fall outside the CI responsibilities.

So, after conducting a survey, filtering and analyzing the data and having group discussion sessions, the author with the CI team created 7 tasks for implementation. The results of the whole study process are summarized in Fig. 5.1 according to the methodology explained in Section 3.5. The selected tasks for the implementation are described below.

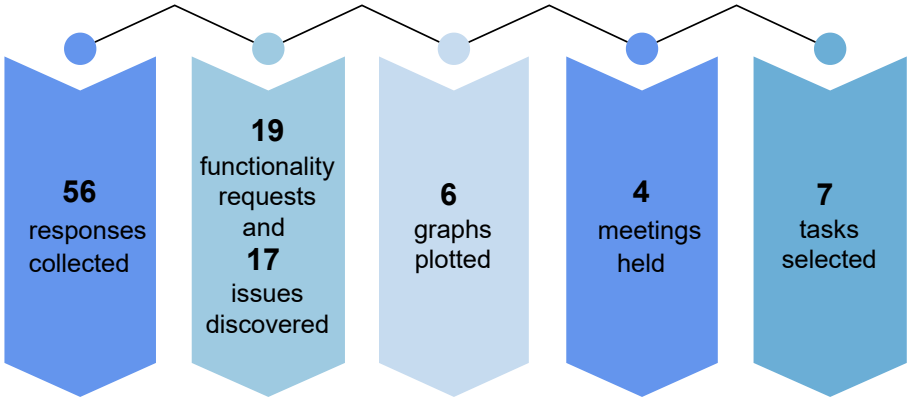

Figure 5.1. *Results of the methodology steps.*

Code coverage test in pipeline. Code coverage, alongside code style check, is the most popular feature request by the survey participants. Code coverage measures how comprehensibly the software is verified by tests. This measurement tool, however, is already implemented. But what can be added is a check of whether the code coverage decreased or increased with the new code. In case code coverage decreases below a set threshold, the developer won't be allowed to submit code. This way, test engineers will need to update test cases to verify the software, ensuring the code coverage is at a favorable value.

Code style check enabling. The company's developers use a variety of programming languages. This fact complicates the implementation of code style checking, as it needs to be tailored for every team and repository. Instead, the author and the CI team decided to enable the code style feature, but leave the implementation to the teams, so that they could choose their preferred style checking tool. Code style tools enable developers to write clean code that adhere to the programming style standards. This practice makes it easier for developers to understand someone else's code, accelerating the development process.

VCS log enhancement. Log enhancement is the second most requested feature. Changing the log structure in the used tools is not possible most of the time. However, we can add additional messages as a part of a code review that are posted in the VCS. One of the survey responses requested some way to pinpoint error messages to speed up debugging. The CI team can create a utility that goes thought the log and searches for error messages with a keyword. Then, the first error message can be posted in the VCS. The reason for posting only the first error is to make the log less misleading, as some respon-

dents pointed out that error log can mislead the developers. An error at the beginning of a test job can lead to a cascade effect of errors. However, in general, is it the very first error that helps identify the root cause.

Groovy script templates for Jenkins. The next most common request for a new feature is templates. Indeed, templates for Groovy script that are utilized in Jenkins can greatly speed up the test creation process as well as provide best practices. Additionally, this feature is quick to implement relative to the other discussed items. The templates should include a basic Jenkins job structure to provide guidance to the engineers. It can also include an example on how to include a stage dedicated to static analysis, as one of the respondents has requested a separate static analysis feature. This feature is already available, but, evidently, instructions are needed on how to implement it.

Single documentation platform. Documentation and instruction problems are one of the most common ones. As the respondents explain, the manuals are often scattered in different platforms. To fix the issue, we chose a single platform to unite all the documentation regarding CI. Additionally, the number of guides will increase to ease the complexity of the system.

Common errors guide. According to the employees, the pipeline is unstable and complicated, the errors are unclear, there aren't enough instructions, and there are too many tools that create many points of failure. The internal discussion lead to the idea of creating a guide that would help deal with common errors to tackle the above-mentioned complaints. With this documentation, the developers would know how to resolve frequent issues in CI, which in return speeds up the debugging process.

Trainings. As requested, the team is scheduling trainings on the already existing tools. Additionally, we plan on having introduction sessions with every new tool or substantial feature introduced to the workflow.

For each task, we specified a priority and an assignee. With these two variables set, we could specify a sprint, i.e., a timeframe within which the assignee should complete the task.

Regarding the other user feedback, we have notified the networking department about connection instability. In order to avoid requests for already implemented features, communication between the CI team and developers will be improved. For example, each new feature is planned to be accommodated with an email notification with release notes as well as possible scheduled presentation meeting, where the employees can ask questions. For the lack of staff to write documentation, the CI team will contact upper management and explain the situation. The CI team will also notify developer teams regarding the 4 tasks that are not CI responsibilities mentioned in the beginning of this subsection.

Finally, the CI team was very happy to receive the feedback, find its weak and strong

points and have a clear understanding of area of focus.

5.4 Chapter summary

The author held four meetings with the CI team. The purpose is to notify the CI team about the general attitude of the employees regarding CI, assess the features suggestions, address reported issues, choose tasks for the roadmap, determine task priority and schedule, and allocate assignees.

The survey revealed unexpected findings. it is likely that the need for new tools or functionalities is satisfied, but still, the employees are not opposed to the idea. A lot of reported issues are out of CI team's responsibility, showing incorrect understanding of CI work. There is a lack of a communication channel between the developers and the CI team. Knowledge sharing about CI is poor. There aren't enough employees to implement the suggestions from the survey.

Background slightly affected the answers. Some engineers believe that managers have a priority when asking for new functionalities. Senior and middle employees are less inclined to use new tools compared to junior employees. Some tasks did not align with employee background, which might indicate an unclear delegation of responsibilities. Employees asked for better documentation and manuals regardless of level of experience.

In the end, the roadmap consists of 7 tasks: (i) add code coverage test to pipeline; (ii) enable code style check; (iii) enhance the log of the VCS; (iv) write Jenkins script templates; (v) create a single documentation platform; (vi) write a guide for common errors; (vii) hold additional trainings.

6. CONCLUSION

The final chapter concludes the thesis. Section 6.1 summarizes and reflects upon the whole project, and Section 6.2 discusses possible future work.

6.1 Summary

The aim of the thesis is to improve the usability of a Continuous Integration workflow within System-on-Chip Software department of the company. To achieve the first research objective (identification of experienced issues and functionality ideas), the author developed an online survey to gather feedback on CI and distributed it among 187 employees. Then, for the second research objective (analysis of user study data), the author analyzed 56 survey responses with a thematic matrix and visualized the results. Finally, the third objective (development of usability improvement roadmap) was achieved by holding meetings with CI stakeholders. In the end, a roadmap was produced aimed at improving the usability of CI based on employees' requests. Therefore, all three research objectives have been completed.

As the survey suggests, even though **employees report favorable attitude towards CI, they still experience issues with it**, like instability and slowness. Furthermore, **there is a demand for new features**, specifically code analysis and log enhancement, among other requests.

The thesis shows the importance of assessment of established development practices, including continuous practices. It proved that the company's CI implementation has potential for improvement, as 89% of survey respondents indicated that they experience issues with the workflow. Besides achieving the research objectives, the survey also revealed deep-rooted problems like shortage of employees and a lack of a communication channel between the developers and the CI team.

There were challenges during the implementation of the thesis. The analysis showed that the department faces technical limitations, causing network failures, slow image building and long test running time. These limitations are not straight-forward to solve and require additional resources.

For practical application, the thesis may serve as an example of an approach to CI us-

ability assessment. At the same time, one must note that this thesis is company-specific.

6.2 Future work

In order to validate the work, evaluation of the CI workflow is required once the roadmap tasks are completed. For evaluation, the author plans to utilize other user research methods. For example, the author can conduct a follow-up survey or user testing, where participants would need to perform pre-determined tasks. User testing shows how users achieve their goals in-action and gives direct feedback on usability [33]. An additional interview after user testing can further reveal user's opinion about CI improvement.

In case the evaluation proves that more improvement work is required, the author plans to repeat the process. Thus, usability assessment can become an iterative cycle. This way, the employees can regularly provide feedback and fresh ideas.

REFERENCES

[1] Phillips, A., Sens, M., De Jonge, A. and Van Holsteijn, M. The IT Manager's Guide to Continuous Delivery: Delivering business value in hours, not months. *XebiaLabs* (2015).

[2] Humble, J. and Farley, D. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[3] Meyer, M. Continuous integration and its tools. *IEEE software* 31.3 (2014), pp. 14–16.

[4] *CI/CD - Explore - Google Trends*. URL: `https://trends.google.com/trends/explore?date=2017-07-21%202022-07-21&q=%2Fg%2F11c2p8c42b` (visited on 07/21/2022).

[5] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. Quality and productivity outcomes relating to continuous integration in GitHub. *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. 2015, pp. 805–816.

[6] Ståhl, D. and Bosch, J. Experienced benefits of continuous integration in industry software product development: A case study. *The 12th IASTED International Conference on Software Engineering*. 2013, pp. 736–743.

[7] Zampetti, F., Geremia, S., Bavota, G. and Di Penta, M. Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2021, pp. 471–482.

[8] Dix, A., Finlay, J., Abowd, G. D. and Beale, R. *Human-computer interaction*. Pearson Education, 2003.

[9] *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*. Standard. Geneva, CH: International Organization for Standardization, July 2019.

[10] Maguire, M. Methods to support human-centred design. *International journal of human-computer studies* 55.4 (2001), pp. 587–634.

[11] *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. Standard. Geneva, CH: International Organization for Standardization, Mar. 2018.

[12] Shackel, B. 1984. The concept of usability. *Proceedings of the IBM Software and Information Usability Symposium. IBM Corporation, Poughkeepsie, NY*. 1981, pp. 1–30.

[13] Whiteside, J., Bennett, J. and Holtzblatt, K. Usability engineering: Our experience and evolution. *Handbook of human-computer interaction*. Elsevier, 1988, pp. 791–817.

[14] Nielsen, J. Usability laboratories. *Behaviour & Information Technology* 13.1-2 (1994), pp. 3–8.

[15] Beck, K. Embracing change with extreme programming. *Computer* 32.10 (1999), pp. 70–77.

[16] Beck, K. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[17] Hilton, M., Tunnell, T., Huang, K., Marinov, D. and Dig, D. Usage, costs, and benefits of continuous integration in open-source projects. *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2016, pp. 426–437.

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

[19] Humble, J. *Continuous Delivery vs Continuous Deployment*. Mar. 1, 2016. URL: https://continuousdelivery.com/%202010/08/continuous-delivery-vs-continuous-deployment/ (visited on 12/03/2021).

[20] Homès, B. *Fundamentals of software testing*. John Wiley & Sons, 2013.

[21] Chacon, S. and Straub, B. *Pro Git. Git Branching - Branches in a Nutshell*. 2009. URL: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell (visited on 07/14/2022).

[22] Egele, M., Scholte, T., Kirda, E. and Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)* 44.2 (2008), pp. 1–42.

[23] Smith, P. *Software build systems: principles and experience*. Addison-Wesley Professional, 2011.

[24] Müller, T., Graham, D., Friedenberg, D. and Veedendal, E. van. ISTQB certified tester-foundation level syllabus. (2007).

[25] Taipale, O., Kasurinen, J., Karhu, K. and Smolander, K. Trade-off between automated and manual software testing. *International Journal of System Assurance Engineering and Management* 2.2 (2011), pp. 114–125.

[26] *Jenkins*. URL: https://www.jenkins.io/ (visited on 07/12/2022).

[27] Krusche, S. and Alperowitz, L. Introduction of continuous delivery in multi-customer project courses. *Companion Proceedings of the 36th International Conference on Software Engineering*. 2014, pp. 335–343.

[28] Harte, R., Glynn, L., Rodrıéguez-Molinero, A., Baker, P. M., Scharf, T., Quinlan, L. R. and ÓLaighin, G. A human-centered design methodology to enhance the usability,

human factors, and user experience of connected health systems: a three-phase methodology. *JMIR human factors* 4.1 (2017), e8.

[29] Jackson, T. Motivating sustainable consumption. *A review of evidence on consumer behaviour and behavioural change In: A report to the Sustainable Development Research Network, as part of the ESRC Sustainable Technologies Programme, Centre for Environmental Strategy, University of Surrey, Guildford* (2004).

[30] Rohrer, C. When to use which user-experience research methods. *Nielsen Norman Group* 12 (2014).

[31] Courage, C. and Baxter, K. *Understanding your users: A practical guide to user requirements methods, tools, and techniques*. Gulf Professional Publishing, 2005.

[32] Love, S. *Understanding mobile human-computer interaction*. Elsevier, 2005.

[33] Aldersey-Williams, H., Bound, J. and Coleman, R. *The methods lab: user research for design*. Design for Ageing Network (DAN), 1999.

[34] Crystal, A. and Ellington, B. Task analysis and human-computer interaction: approaches, techniques, and levels of analysis. *AMCIS 2004 Proceedings* (2004), p. 391.

[35] Steen, M. The fragility of human-centred design. (2008).

[36] Kumar, V. and Whitney, P. Faster, cheaper, deeper user research. *Design Management Journal (Former Series)* 14.2 (2003), pp. 50–57.

[37] Brun-Cottan, F. and Wall, P. Using video to re-present the user. *Communications of the ACM* 38.5 (1995), pp. 61–71.

[38] Laugwitz, B., Held, T. and Schrepp, M. Construction and evaluation of a user experience questionnaire. *Symposium of the Austrian HCI and usability engineering group*. Springer. 2008, pp. 63–76.

[39] Brüggen, E. and Willems, P. A critical comparison of offline focus groups, online focus groups and e-Delphi. *International Journal of Market Research* 51.3 (2009), pp. 1–15.

[40] Lacey, A. and Luff, D. *Qualitative data analysis*. Trent focus Sheffield, 2001.

[41] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visited on 07/21/2022).

[42] Gerrit. *Gerrit Code Review*. 2022. URL: https://www.gerritcodereview.com/index.html (visited on 07/12/2022).

[43] Git. *Git*. 2022. URL: https://git-scm.com/ (visited on 07/12/2022).

[44] Zuul Contributors. *Zuul is an open-source CI tool*. 2022. URL: https://zuul-ci.org/ (visited on 07/12/2022).

[45] *Blue Ocean*. URL: https://www.jenkins.io/projects/blueocean/ (visited on 07/12/2022).

[46] *Robot Framework*. URL: https://robotframework.org/ (visited on 07/12/2022).

[47] *GitHub - google/googletest: GoogleTest - Google Testing and Mocking Framework*. URL: https://github.com/google/googletest (visited on 07/12/2022).

[48] Yocto Project. *Yocto Project – It's not an embedded Linux distribution – it creates a custom one for you*. 2022. URL: https://www.yoctoproject.org/ (visited on 05/24/2022).

[49] Microsoft. *Data Visualisation | Microsoft Power BI*. 2022. URL: https://powerbi.microsoft.com/en-au/ (visited on 05/24/2022).

[50] Synopsys, Inc. *Coverity Scan - Static Analysis*. 2022. URL: https://scan.coverity.com/ (visited on 05/24/2022).

[51] SonarSource S.A. *Code Quality and Code Security*. 2022. URL: https://www.sonarqube.org/ (visited on 05/24/2022).

[52] JFrog Ltd. *Artifactory - Universal Artifact Repository Manager - JFrog*. 2022. URL: https://jfrog.com/artifactory/ (visited on 05/24/2022).

[53] GitLab B.V. *GitLab CI/CD | GitLab*. 2022. URL: https://docs.gitlab.com/ee/ci/ (visited on 05/24/2022).

[54] Docker Inc. *Home - Docker*. 2022. URL: https://www.docker.com/ (visited on 05/24/2022).

[55] The Kubernetes Authors. *Kubernetes*. 2022. URL: https://kubernetes.io/ (visited on 05/24/2022).

[56] Splunk Inc. *Splunk | The Data Platform for the Hybrid World*. 2022. URL: https://www.splunk.com/ (visited on 06/01/2022).

[57] Gerrit Code Review. *Using Topics - Gerrit User Guide*. 2022. URL: https://gerrit-review.googlesource.com/Documentation/intro-user.html#topics (visited on 06/03/2022).