Empirical Software Engineering (2023) 28:144
https://doi.org/10.1007/s10664-023-10356-1

Automated NFR testing in continuous integration environments: a multi-case study of Nordic companies

Liang Yu[1] · Emil Alégroth[1] · Panagiota Chatzipetrou[2] · Tony Gorschek[1,3]

Accepted: 7 June 2023 / Published online: 24 October 2023

Abstract

Context Non-functional requirements (NFRs) (also referred to as system qualities) are essential for developing high-quality software. Notwithstanding its importance, NFR testing remains challenging, especially in terms of automation. Compared to manual verification, automated testing shows the potential to improve the efficiency and effectiveness of quality assurance, especially in the context of Continuous Integration (CI). However, studies on how companies manage automated NFR testing through CI are limited.
Objective This study examines how automated NFR testing can be enabled and supported using CI environments in software development companies.
Method We performed a multi-case study at four companies by conducting 22 semi-structured interviews with industrial practitioners.
Results *Maintainability*, *reliability*, *performance*, *security* and *scalability*, were found to be evaluated with automated tests in CI environments. Testing practices, quality metrics, and challenges for measuring NFRs were reported.
Conclusions This study presents an empirically derived model that shows how data produced by CI environments can be used for evaluation and monitoring of implemented NFR quality. Additionally, the manuscript presents explicit metrics, CI components, tools, and challenges that shall be considered while performing NFR testing in practice.

Communicated by: Smita Ghaisas

✉ Liang Yu
liang.yu@bth.se

Emil Alégroth
emil.alegroth@bth.se

Panagiota Chatzipetrou
panagiota.chatzipetrou@oru.se

Tony Gorschek
tony.gorschek@bth.se

[1] Blekinge Institute of Technology, Karlskrona, Sweden

[2] Department of Informatics, CERIS, Örebro University School of Business, SE-701 82 Örebro, Sweden

[3] Fortiss GmbH, 80805 Munich, Germany

Keywords Non-functional requirements · NFR · Continuous integration · CI · Automated testing · Metrics · Case study

1 Introduction

Non-functional requirements (NFRs) are the qualities of a system (Werner et al. 2020), which describe *how* a system fulfills functional requirements. The attention for NFRs increases in software development companies since non-functional qualities are nowadays seen as critical success factors in the marketplace (Werner et al. 2020). Whilst some NFRs (for example, *performance*) are expected by users, others (for example, *privacy*) must be ensured by the software's design if it is to comply with national or international regulations (for example, the General Data Protection Regulation (GDPR)).

Most frequently, NFRs define cross-functional quality aspects of a system and, in some instances, are viewed as being more important than the system's functionalities (Júnior 2020). Due to their cross-functional characteristics, NFR verification and validation (Karhapää et al. 2021) is a challenging task (Alsaqaf et al. 2019). Therefore, conventional NFR testing practices are primarily performed manually, which is neither efficient nor effective (Júnior 2020). In fact, faulty NFRs produce additional work, which can account for 40% to 50% of the total work done in some software projects (Wagner 2006; Boehm and Basili 2005). Given this, we note that many product development organizations lack a shared understanding of NFR validation (Werner et al. 2020) in a continuous software engineering context.

Continuous integration (CI) (Fitzgerald and Stol 2017) has been used to assist automated software verification (Dlugi et al. 2015) and validation (Rathod and Surve 2015). CI environment is an integral part of modern testing teams (Knauss et al. 2016) and contains a set of components (Yu et al. 2020) (for example, tools and frameworks). Dlugi et al. (2015) proposed a CI framework to verify and validate system functionalities in fast iterations, and Knauss et al. (2016) reported that fast code quality feedback enabled by CI can prevent faults slipping through the development phase to later phases. Further, Fitzgerald and Stol (2017) adopted test automation tools to detect bugs in source codes, and Shahin et al. (2017) investigated CI practices associated with tools to improve software testing processes. These studies show the importance of using CI to evaluate functional requirements. However, how one might best capitalize on the CI and CI environments for non-functional requirement testing is still an open area of research (Paixão et al. 2017).

Our interest in this area of research is informed by the results of a systematic literature review (SLR) (Yu et al. 2020) on the topic of utilizing CI environments for NFR testing. The main findings of the SLR were that automated NFR testing can be achieved in an academic setting, and CI can support[1] or enable[2] NFR evaluation. However, the practices of NFR test-automation in industrial contexts were not covered.

In this study, we performed a multi-case study at companies that operate in diverse business domains to investigate how they are using CI environments to identify practices that seek to measure the NFR testing. This research is also perceived to provide guidance for researchers in terms of NFR evaluation through CI environments from industry. The results of the present

[1] support refers to providing inputs or resolving other prerequisites for automated tests

[2] enable explicitly refers to making automated NFR tests possible

study provide practitioners and researchers with a guide on NFR evaluation (Miller 2008) using CI environments.

Based on the theory classification of Gregor (2006), the main contributions of this study are our provision of:

1. An empirically derived model that explains (Gregor 2006) how CI environments produce test data, useful for evaluation of NFRs, and to monitor quality trends, through automated notification systems based on test outcomes.
2. Practical knowledge and actions (Gregor 2006) for NFR testing by using CI environments, that is, involvement of existing CI environments leads to a greater improvement of automated testing for NFRs.

The knowledge includes:

– An understanding of the test capabilities of a CI environment. Study results show correlation between CI components and NFR test capabilities.
– A view of the status on how companies handle NFR evaluation. Common NFR types and NFR metrics were identified in this study.

The actions contain:

– Practices of adding or upgrading components and tools in a CI environment to enable automated verification for particular NFRs (e.g., automated security scans) and support NFR evaluation in terms of fast test execution and continuous analysis and monitoring on test outputs.
– A guideline to collect metric data through CI components to measure NFRs.
– Challenges to be considered while performing NFR testing in practice.

The rest of this paper is structured as follows: Section 2 presents the background and related work. In Section 3, we describe the research methodology that was adopted in this study. In Section 4, we present results of the study, and in Section 5, we discuss threats to the study validity. In Section 6, we discuss the findings and implications of our research, and in Section 7, we share our conclusions and indicate where future research is needed.

2 Related Work

NFRs specify system qualities (Gorschek and Wohlin 2006; Mairiza et al. 2010) – additional to the system's functionality. These qualities can be categorized as either internal qualities (ISO/IEC-25023 2016) (e.g., extendability and testability) or external qualities (Khurum et al. 2014) (e.g., user experience and performance).

To ensure system qualities (Svensson et al. 2011), NFRs are important aspects to assess software products. For example, a system must have sufficient usability (Chung and do Prado Leite 2009), defined as its usefulness and performance, to be usable. If a system does not meet the privacy policy required by users or customers, then the system may not be usable in practice. Moreover, the needs of fast evaluation in the CI environment applies to NFRs, and the potential of using automated testing (Paixão et al. 2017) is important to avoid bottlenecks in CI environment. Thus, warranting additional research into tools, methods, and techniques for automated NFR testing.

However, there are challenges to evaluate some types of NFRs in the industry (Karhapää et al. 2021). As stated, many NFRs (for example, *reliability* (Chen 2015)), specify qualities that are influenced by many aspects of a system (Júnior 2020), thereby placing prerequisites on data capture and storage and even access to the entire system. This circumstance stands in contrast to functional testing, where a function is often isolated or restricted to the testing of a single software component. Moreover, some NFR types are grounded in the users' subjective views (Werner et al. 2020) (e.g. User experience) or other qualitative factors (Werner et al. 2021). These factors can be difficult or impossible to quantify for the automated testing (Werner et al. 2021). For example, whilst one user may perceive a system's performance as adequate, another may perceive it as unsuitable.

Existing literature confirms that NFR testing is challenging. Alsaqaf et al. (2019) reported challenges while managing NFRs in large-scale distributed agile projects. Karhapää et al. (2021) did further industrial case studies and identified obstacles for verifying quality requirements using CI particularly. These studies emphasized the challenges of NFR evaluation in industrial contexts, but did not explore the practices of using CI environments for NFR test-automation.

To explore NFR testing in CI environments, we found relevant studies in our own recent systematic literature review (SLR) (Yu et al. 2020). For example, Rehmann et al. (2016); Nouacer et al. (2016) and Bougouffa et al. (2017) reported three different CI approaches to validate system qualities, such as performance, reliability, and security, in multiple software products. As these studies had a narrower scope focusing only on a particular NFR type, Cannizzo et al. (2008); Janus et al. (2012) and Chen (2015) investigated the possibility of using a CI approach to evaluate multiple NFR types, such as, system's robustness, performance, and maintainability, but they did not offer detail information about how the NFR testing can be measured.

To complement the existing work, Staron et al. (2011) proposed a way of using specific metrics to measure NFRs through CI approaches, and López et al. (2022) explored a systematic mapping between metrics and NFRs. However, their focus was to analyze and visualize metrics through measurement systems instead of practices of automated tests using metrics. Thus, we conduct this research to shed light on automated NFR testing through CI environments.

3 Research Methodology

Our research process, visualized in Fig. 1, consisted of four phases. Phase 1 aimed to define research questions and goals. Phase 2 focused on the case study design, including case study plan (Runeson and Höst 2009), interview questions' preparation, participant selection, and pilot testing of the interview questions. In Phase 3, we conducted interviews and transcribed interview recordings. Phase 4 targeted analysis and synthesis of the collected data. In the continuation of this section, we describe the individual parts of the process in more detail.

3.1 Research Questions

The study is an investigation into how automated NFR testing is achieved in CI environments in software development companies. Then, four research questions are presented:

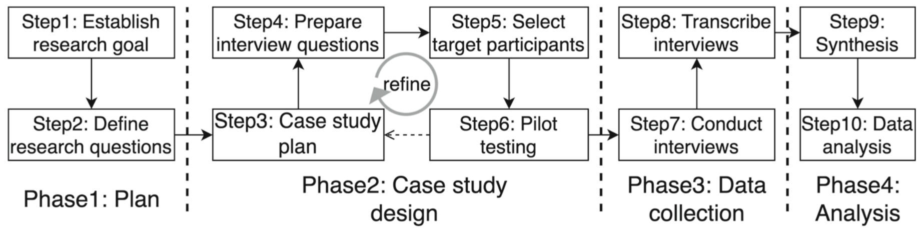

Step1: Establish research goal
Step2: Define research questions
Phase1: Plan
Step4: Prepare interview questions
Step5: Select target participants
refine
Step3: Case study plan
Step6: Pilot testing
Phase2: Case study design
Step8: Transcribe interviews
Step7: Conduct interviews
Phase3: Data collection
Step9: Synthesis
Step10: Data analysis
Phase4: Analysis

Fig. 1 An overview of the research processes

– **RQ1**: What types of NFRs are verified through automated tests?

RQ1 aims to identify what specific types of NFRs have been evaluated in case companies.

– **RQ2**: What metrics are used for automated NFR testing in industrial practice?

RQ2 expands on RQ1 and aims to identify practices including test processes, metrics, and tools that are used for NFR testing. By adopting this higher-level perspective, we examine implicit relationships between NFR metrics and CI environments.

– **RQ3**: How are CI environments contributing to NFR test capabilities?

RQ3 extends the scope of RQ2 by explicitly focusing on how how the components in a CI environment impact the NFR testing. The goal includes examining, for example, how a single, several, or a set of CI components affect NFR testing capabilities.

– **RQ4**: What challenges are associated with automated NFR testing?

RQ4 targets to identify challenges of NFR testing in the studied projects and practices to mitigate the challenges.

3.2 Case Study Design

We conducted a multi-case study (Runeson and Höst 2009) in software development companies. A multi-case study involves the analysis of multiple cases for comparisons, that can provide a more comprehensive understanding of NFR issues in companies and suitable for making generalizations about the issues, when comparing to a case study with analysis of a small number of cases.

3.3 Case Companies

In this study, our case companies are *Qvantel Sweden*, *Ericsson*, *Fortnox*, and *Company Alpha*,[3] where we had access to practitioners, products, testers, test-cases, and processes to study automated NFR testing. As shown in Table 1, company contexts are presented, and the selected projects were randomly assigned index names from A to E to remove traceability back to the case companies.

Ericsson is a multinational company that produces software and services for telecommunications. The studied projects contain a financial system and a large-scale telecommunications

[3] Company Alpha’s is kept anonymous due to a non-disclosure agreement (NDA).

Table 1 Projects and contexts (Petersen and Wohlin 2009) in case companies

Context facet	Context element	Project A	Project B	Project C	Project D	Project E
Project	Business domain	Finance technology	Telecommunications	Finance technology	Digital communication	Health and social affairs
	Project size (Sas and Avgeriou 2020) (number of engineers)	Medium (101 to 200)	Large (>1500)	Medium (51 to 100)	Small (21 to 50)	Larget (>1000)
	Product type	Finance services	Common platform for telecommunications	Banking platform solutions	Digital business support solutions	Common welfare solutions
	Maturity of product	Mature product	Long-lived mature product	Mature product	Mature product	Long-lived mature product
Organization	Size (number of engineers)	∼300	>2500	∼100	∼50	>1000
	Agile adoption	Yes(global)	Yes(global)	Yes(nationally)	Yes(global)	No
	Teams involved	Development & test teams from one location	Development & test teams from one location	Development teams from one location	Development & test teams from one location	Development & test teams from one location
	Distributed development	Yes	Yes	No	Yes	Yes
Market	Type of customer	Global organizations	Global organizations	National individuals	Global organizations	National citizens
NFRs	Testing process	Plan-driven and agile-based continuous development	Plan-driven and agile-based continuous and incremental development	Incremental development	Mixture of Scrum and Kanban for incremental development	Mixture of Scrum and Kanban for incremental development
	Testing demands	Need more	Need more	Need more	Need more	Need improvements
CI environment	CI maturity	Mature	Mature	Fast growing	Fast expansion	Mature

service. They have different CI environments which have been used for both functional and non-functional requirements' evaluation. The CI environments steer practices for global teams, and provide a shared process with practices including code reviews, compilation, building, integration, and automated testing.

Qvantel Sweden is a fast-growing company that provides cloud-native business support services, which help customers run their business for a new level of autonomy, faster time to market, and flexibility to adapt to changing business needs. The CI environment supplies a solid CI process allows developers to deliver smaller portions of code. It allows for non-duplicated work for developers to ensure that the code they commit is as bug-free as possible, and reduces the number of manual and repetitive tasks that developers face.

Fortnox is a cloud-based platform that meets the needs of small businesses and accounting agencies managing their finances efficiently. The CI environment helps developers catch security issues earlier through automated CI tests.

Company Alpha is a welfare agency. The project in this company is a legacy system that concerns businesses on insurance services through web interfaces. The CI environment in this project increases transparency between team members and covers automated performance testing. It provides neutral feedback continuously, which could help to mitigate the damage of future build failures or merging issues, and teams could consider feedback as a way to learn and improve rather than something that is negative.

3.3.1 Prepare Interview Questions

An interview guide, as shown in Appendix A, was constructed to ensure that multiple researchers could perform the interviews in parallel. The guide is divided into distinct parts that include (i) a description of the purpose of the study, (ii) the study procedure, (iii) a description of how the participants' confidentiality is assured, (iv) how the participants can give feedback or add to their interview answers (Runeson and Höst 2009), and finally, (v) a list of predefined interview questions for data collection. The interview questions were divided across three main areas: (a) types of NFRs that are verified or validated with automated tests, (b) test practices that describe how the automated NFR tests are performed, (c) challenges associated with the NFR testing.

3.3.2 Select Participants

We adopted *convenience sampling* (Wohlin et al. 2012), selecting participants based on their availability and willingness to participate in this study.

We first identified the population of interest, which is a group of industrial engineers who seek to use or improve their NFR testing by using the CI environment. To ensure the target participants can provide information of NFR testing through CI, we defined specific sampling criteria, such as working experience and job role, as shown in Table 2, aiming to increase the representativeness of the sample. The working experience includes domain knowledge in both NFR testing and using CI techniques for test automation, and at the time of this study, 86% of selected participants had more than five years of work experience, which could be an indicator of the participants' ability to answer our interview questions. Moreover, the job role contains tester, product owner, software architect, and developer to collect data from different perspectives to minimize the bias of data sources.

Table 2 Participant details: ID, Role, Work experience, and Project name

Participant ID	Participant role	Work experience (years)	Project name
T1	Lead developer	10 - 15	Project A
T2	Software architect	5 - 10	
T3	Senior developer	10 - 15	
T4	Developer	3 - 5	
T5	Software architect	5 - 10	
T6	Tester	3 - 5	
T7	Tester	5 - 10	
T8	Senior tester	10 - 15	
T9	Developer	3 - 5	Project B
T10	Developer	5 - 10	
T11	Product owner	10 - 15	
T12	Tester	5 - 10	
T13	Principal developer	15 - 20	Project C
T14	Senior architect	10 - 15	
T15	Software architect	5 - 10	
T16	Tester	5 - 10	
T17	Developer	10 - 15	Project D
T18	Tester	5 - 10	
T19	Tester	5 - 10	
T20	Developer	5 - 10	Project E
T21	Software architect	5 - 10	
T22	Tester	5 - 10	

We contacted target engineers through email, phone calls, or in person and asked them if they would be willing to participate in the study. We explained the purpose of the study, study procedure, and how the participants data would be used. After the data collection and analysis, we reported our synthesized results back to the participants to acquire feedback.

3.3.3 Pilot Testing

The interview guide was tested in four pilot rounds, each involving a distinct set of participants. The aim of these pilot tests was to gather information that would help refine our interview questions.

The first author conducted two pilot interviews with industrial professionals, while the second author conducted the remaining two interviews with external researchers. During the first pilot test, it was identified that two questions concerning the “waiting-time to get feedback from the CI server” produced overlapping semantic outcomes. As a result, these questions were refined. In the second pilot test, the question pertaining to “feedback time” was found to be ambiguous and was subsequently rephrased. The third pilot test revealed that a few participants did not understand the abbreviation “NFR.” To address this, a definition was added to the guidelines. The fourth pilot test did not find any additional issues, and all participants from the previous pilots were included to ensure their agreement with the revised interview questions.

It is important to note that these pilot tests were not included in the study findings.

3.4 Data Collection

We collected qualitative data through semi-structured interviews of 22 participants from the case companies. A summary of the interviewees can be found in Table 2. The interviews were conducted by two authors with each participant in person. They lasted between 33 to 54 minutes, and the average interview time was 46 minutes. The time duration does not include the “warm up” and “cool down” parts. The first author worked as an embedded researcher in Ericsson, and the second author conducted three interviews together at Fortnox to extract information. After the interviews, two separate meetings were conducted by the authors to discuss and merge the collected data. With this experience and learning, the first author executed the rest of interviews in the other companies.

We started each interview by going through the questions defined in our interview template. We followed with additional probing questions or open discussions on more specific subjects, depending on an interviewee’s role. For example, an interview with a software architect involved extended questioning on NFR test processes and designs to interpret what was done and how. All interviews were recorded with audio and notes.

3.5 Data Analysis

We employed thematic analysis (Cruzes and Dyba 2011), an established data analysis method to identify themes and patterns in our collected data, and the procedure is presented in Fig. 2.

As can be seen in the figure, the thematic analysis contains four stages. Stage 1 focuses on collecting codes from interview recordings. Stage 2 is about how we grouped codes into different categories based on semantic equivalence or a shared contextual relationship (Petersen and Wohlin 2009) (for example, NFR types, tools, and CI components). In Stage 3, we synthesized codes into categories in higher-level themes (Cruzes and Dyba 2011). Stage 4 is where we drew conclusions using the synthesized themes.

Our analysis involved inductively identifying themes from raw transcripts, that related to practices and challenges of NFR testing in the studied projects. The open coding (Corbin and Strauss 1990) approach was used to minimize the bias of analysis, and we used the constant comparison method, whereby codes were added and merged based on discussions between authors. In our initial coding phase, the first two authors coded dialog segments in our interview template independently, and a meeting was conducted with all authors to discuss their understanding of the codes after coding each transcript.

The initial codes and transcripts were stored in an Excel file with tables, example shown in Table 3. The codes and quotes were incrementally developed from examining interview transcripts. Each transcript that was mapped to a code was tagged with the ID of a participant, which allows us to record how many individual statements (made by different participants)

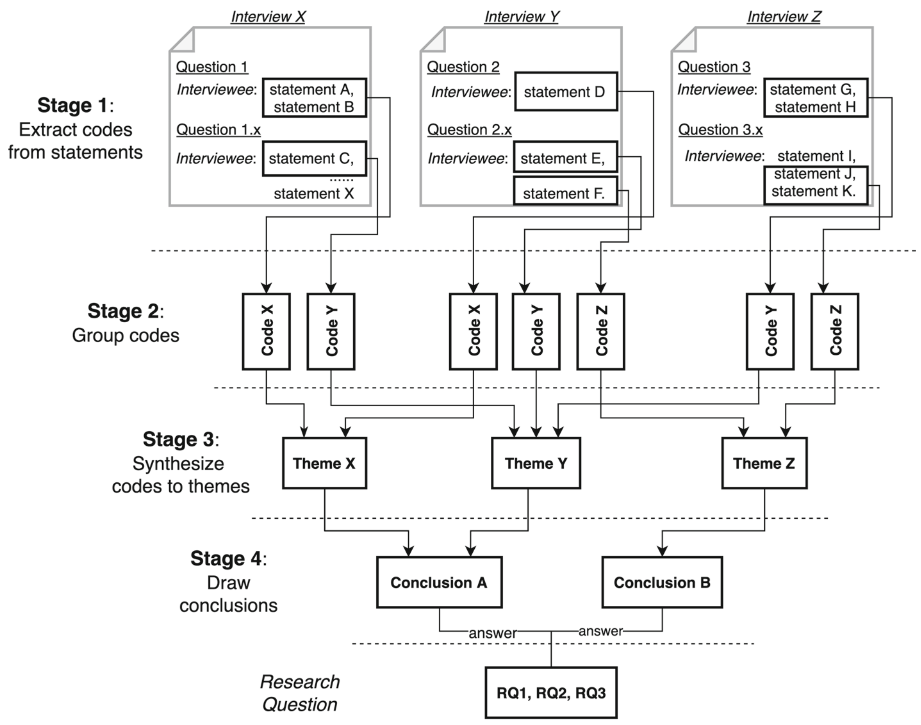

Interview X
Question 1
Interviewee: statement A, statement B
Question 1.x
Interviewee: statement C, ... statement X
Interview Y
Question 2
Interviewee: statement D
Question 2.x
Interviewee: statement E, statement F.
Interview Z
Question 3
Interviewee: statement G, statement H
Question 3.x
Interviewee: statement I, statement J, statement K.
Stage 1: Extract codes from statements
Stage 2: Group codes
Code X
Code Y
Code Z
Stage 3: Synthesize codes to themes
Theme X
Theme Y
Theme Z
Stage 4: Draw conclusions
Conclusion A
Conclusion B
answer
answer
Research Question
RQ1, RQ2, RQ3

Fig. 2 Thematic analysis (Cruzes and Dyba 2011) stages adopted for coding during the data analysis phase

supported each code. These unique IDs were of the form “Tx”, where “x” is an integer between 1 and 22, and kept the participants anonymous. The IDs also provided traceability back to the transcripts if needed during the analysis to get more contextual information.

When more interview recordings were transcribed, the number of quotes soon became difficult to overview, defeating the purpose of the coding. Thus, another layer of abstraction was added to mitigate this issue, where quotes were replaced with descriptions that could include one or more of the aforementioned codes. This layer could keep the chain of evidence and

Table 3 Example table showing how initial codes and transcripts were stored

Codes	Interview record ID	Timestamp	Quote	RQ	Interviewee IDs
e.g., Code review	e.g., A1	09:59 - 12:42	Add reviewer manually	RQ2	e.g., T5
			Jenkins verification job gives +1 if success	RQ3	
			Product owners give +2 when a commit is approved	RQ2 RQ3	
			Two +2 is set to be mandatory for each Git commit, to ensure it works and encourage people to review carefully	RQ2	

Table 4 Revised example table for initial coding

ID	Codes	Description	Interviewee IDs
e.g., 10	e.g., Code review	Statements about source code reviews.	T1,T2,T3,T4,T5,T6,T7,T8,T9, T14,T15,T17,T18,T20,T21,T22

reduce the cognitive complexity of overviewing the analyzed interview results. An example of what the coding procedure looked like before the abstraction is shown in Table 3, and the result after abstraction is shown in Table 4.

During the coding phase, addressing intercoder disagreements was a primary focus. Our approach involved open discussion, clarification, and revisiting the data to achieve consensus. Regular meetings and in-depth discussions were conducted to facilitate effective communication and exchange of perspectives, enabling us to gain a comprehensive understanding of each other's interpretations. In cases where differences in coding persisted, we dedicated the necessary time to revisit the specific data segments in question. This included a thorough examination of the relevant data, with the aim of exploring its details and contextual significance at a deeper level. Through this analysis, we sought to enhance our understanding and align our interpretations, ultimately striving to achieve a high level of coding agreement.

To retain the traceability, the complete list of extracted codes can be found in Appendix B. To answer the research questions, we developed themes based on thematic synthesis of our coded data. We discussed similarities and differences between codes to group codes into themes, whereby each theme illustrate how a type of NFR is evaluated using CI components and tools.

Additionally, we performed member checking with the study participants to verify our findings resonate with the context of their organization, and the member checking feedback was used to revise our findings.

Data Availability Statement: The raw data used in this study can be obtained by contacting the corresponding author upon request. We have made efforts to remove direct identifiers, such as names, contact information, and project-related details to address potential concerns of re-identification. Our aim is to prioritize the protection of participant confidentiality while ensuring data availability in accordance with appropriate ethical and legal considerations.

In summary, our primary coding groups, extracted themes (e.g., NFR types, tools, and CI components), and the mapping to research questions present in Fig. 3. We used the code groups, NFR types, and the uses of tools and CI components to understand NFR metrics for RQ2. We also added the number of interviewees that support a particular code group for RQ4.

4 Results

This section presents the results of our study in response to our research questions.

4.1 RQ1:What Types of NFRs are Verified Through Automated Tests?

Through the meetings with practitioners from the studied companies, we identified several NFR types that were verified in an automated manner. These NFR types are *maintainability*, *security*, *performance*, *scalability*, and *stability*.

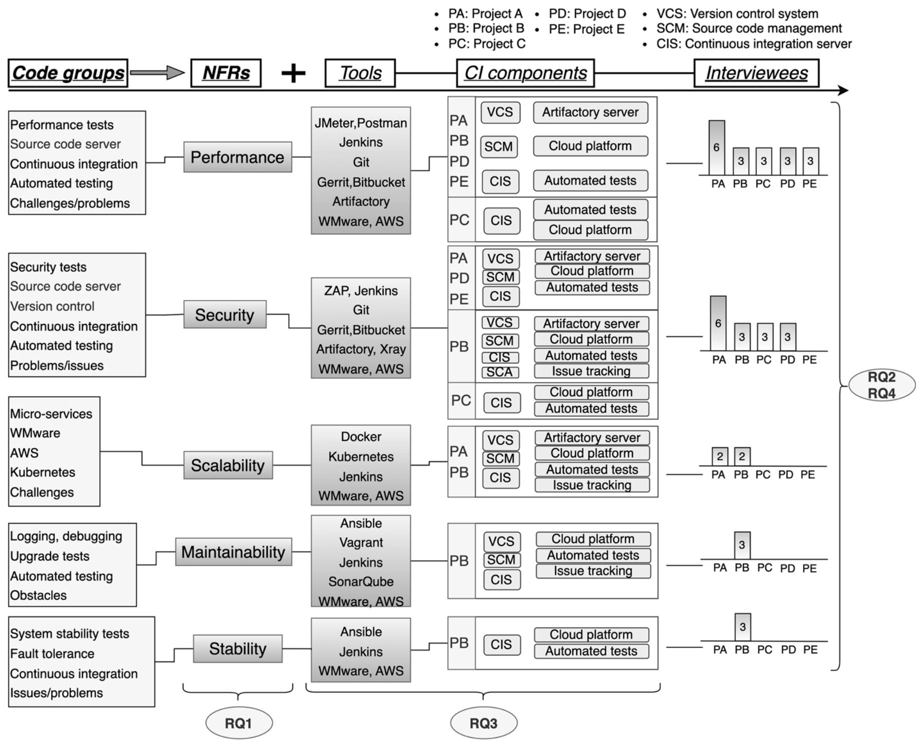


Fig. 3 Synthesis of the collected data including NFR types, tools, and CI components

As shown in Table 5, both internal and external quality attributes (ISO/IEC-25023 2016) were found in the studied projects. From the data in this table, we can see that external attributes of NFRs receive more attention than internal ones in the studied projects. As a practitioner reported that “Customers’ feedback and their priorities play a role in deciding whether NFRs are automated or not.” Low priority or awareness of internal NFRs could cause internal NFRs (e.g., scalability) to not be tested or overlooked (Aljallabi and Mansour 2015), which could result in additional cost. For example, if a system (e.g., banking service) is not built to easily incorporate the functionality required to accommodate different time zones, costly rework may be required (Boehm and Basili 2005).

Moreover, we can see in the above table that *Performance* and *Security* were evaluated more often compared to the other NFRs in the studied projects. This might correlate with how easy the NFR type is to quantify — Performance is perceived as easier to quantify (Rehmann et al. 2016) than many other NFR types. It can also be attributed to growing external factors, such as societal pressures, which may explain the popularity of security testing in the studied companies since user privacy (Remlein and Stachowiak 2021) is expected to be secured in modern software.

Surprisingly, some NFR types (e.g., usability (Cajander et al. 2013), and reliability (López et al. 2022)) are considered important in research but not identified in the studied projects.

Table 5 Identified NFR types and attributes (ISO/IEC-25023 2016)

NFR	Attributes	Internal or external attribute	Project name
Maintainability	Testability Changeability Modifiability	Internal	Project B
Security	Vulnerability Confidentiality Authentication Access control	External	Project A, B, C, and D
Performance	Response time Accuracy Resource utilisation	External	Project A, B, C, D, and E
Stability	Fault tolerance Recoverability	Internal	Project B
Scalability	High availability	External	Project A and B

As practitioners confirmed that "ease of use is critical for software products, but it is not easy to measure the usability of a system with automated tests due to missing a full view of quality requirements and the limited information on using tools to support NFR validation." This finding indicates that aspects, like usability that impact most functionality, are hard to gauge for individual developers focusing on one item at a time often (Alsaqaf et al. 2019), meaning a system view focus on NFRs would be beneficial (Karhapää et al. 2021) for the test-automation of the "missed" NFRs.

Additionally, participants in this study suggested potential improvements for NFR testing, with many commenting on the need for additional metrics to effectively govern NFRs within organizations and improve test skills and knowledge. As a result, the subsequent section explores metrics for NFR evaluation.

4.2 RQ2: What Metrics are Used for Automated NFR Testing in Industrial Practice?

As a way to study how NFR testing were measured, we extracted both NFR types and metrics from industrial practitioners, as shown in Table 6.

It can be seen from the data in Table 6 that multiple metrics are used to measure a NFR type. For example, in the table, performance was measured by using multiple metrics to enable longitudinal evaluation since a single metric could be biased. This indicates that different metrics could provide more comprehensive test results while measuring NFRs.

Moreover, Table 6 also shows that an NFR metric can be implemented with different data that can be collected with CI components through automated jobs. As practitioners reported that CI components hold large amounts of data from source codes/tests, and the data can be used to evaluate, monitor, and analyze NFRs continuously. For example, a CI server can run automated vulnerability assessments against an application and collect the

Table 6 Extracted NFR metrics for automated NFR testing in the studied industrial projects

NFR type	NFR metrics	Data in the studied projects	Related CI components	Project name
Performance	Mean Response Time (MRT): MRT = (A1 + A2 + ...+ An)/n, where *Ai* is the time that a service takes to respond to a request, and *n* is the number of measured responses.	Response time of payment transactions between accounts in a financial service	Source code management Version control system CI server Test automation	Project A
		Response time of managing user subscriptions in a mobile network service	Source code management Version control system CI server Test automation	Project B
		Response time of requests to create users and accounts in a banking system	Source code management Version control system CI server Test automation	Project C
	Processor usage (PU): PU = A/B, where *A* is processor time required to execute a set of tasks, and *B* is operation time that is required to perform the tasks.	Processor time of managing mobile devices and data in a web service	CI server Test automation Cloud platform	Project D
	Accuracy of API requests (AOAR): AOAR = A/B, where *A* is the number of failed API requests, and *B* is the total number of API requests processed by a service.	The accuracy rate of user requests in a welfare system	Source code management Version control system CI server Test automation	Project E
Security	Data encryption (DEC): DEC = A/B, where *A* is the number of user data items encrypted correctly, and *B* is the total number of data items requiring encryption.	User privacy data in a finance system	Source code management Version control system CI server Test automation Issue tracking	Project A

Table 6 continued

NFR type	NFR metrics	Data in the studied projects	Related CI components	Project name
	User access audit, UCA = A/B, where *A* is the number of accesses recorded in security logs, and *B* is the number of executed accesses to data.	User access data in a mobile network service	CI server Test automation Issue tracking	Project B
	Penetration test	Security risks for artifacts in a web application	CI server Artifacts management Cloud platform	Project C
	Vulnerability assessment (VA)	Vulnerabilities in file systems	CI server Artifacts management Issue tracking	Project A, B C, and D
Scalability	Mean recovery time (MRET): MRET = (A1 + A2 + ...+ An)/n, where *Ai* is the total time to recover the failed user requests, and *n* is the number of failed requests.	Recovery time of accessing mobile data from failures in a mobile network service	Source code management Version control system CI server Test automation Artifacts management Cloud platform	Project B
Maintainability	N/A	N/A	Source code management CI server Static code analysis	Project A, B
Stability	Maximum number of requests processed in unit time (MRP): MRP = (A1 +...+ An)/(B1 +...+ Bn), where *Ai* is the total user access that a service can handle within a time period, and *Bi* is the total time to process user accesses.	The capability of a system to handle user access per second in a mobile network service	Source code management Version control system CI server Test automation Cloud platform	Project B

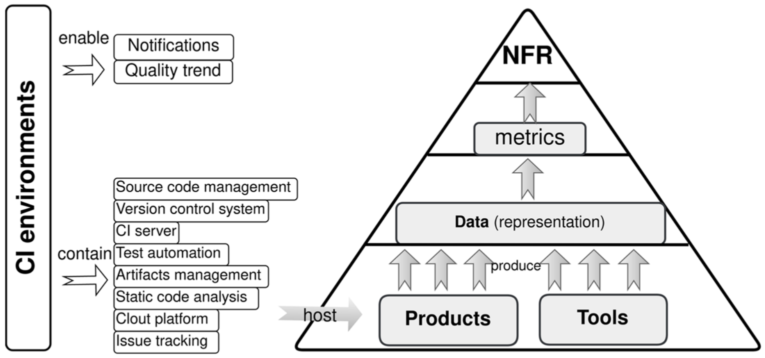


Fig. 4 The uses of CI environments for NFR testing through synthesizing the collected data from industrial participants

number of detected vulnerabilities to support security evaluation. This finding implies that CI environments can be vital in supplying data for NFR measurements and automated test executions.

To demonstrate practitioners' views on how CI contributes to metrics and NFR testing, we transform the collected information into a flowchart, as shown in Fig. 4. What stands out in the figure is that CI environments consist of components that can produce data from products (e.g., source codes) and tools (e.g., CI tools) to develop metrics for NFR testing. Below we describe two typical examples of NFR evaluation to clarify the usage of CI from products and tools' perspectives.

Example 1 CI components produce data from software products to support NFR validation. For instance, the stability measurement steps in Project B are:

1. Developers implemented test cases to measure the maximum number of requests processed in unit time (MRP), and tests are managed in a CI *source code management* component.
2. *CI server* executes the tests and collects data, such as the total number of user requests (e.g., 10000) and test execution period (e.g., 200 seconds), when the source code is changed.
3. CI server uses the collected data and records metric results (e.g., MRP = 10000/200 = 50).
4. CI server notifies developers about the quality impact of their code changes.
5. CI monitors metric results persisted in a *cloud platform* for further quality improvements.

These steps, which are supported by CI components, connect each other and can hardly be replaced by manual work. Without CI components, the extra effort might be required to collect and monitor test data, and the seamless connection between measurement steps would be broken.

Example 2 CI components with associated tools generate data to enable NFR verification. For example, security tools, like Trivy or Anchore, to manage vulnerability assessment in Project A, and the measurement steps are:

1. *CI server* integrates security scanning tools to scan build artifacts (e.g., Jar files) stored in the *artifacts management* component.
2. CI server generates vulnerability reports after the scanning job is finished.

3. CI notifies developers about the new/resolved vulnerabilities.
4. CI produces the vulnerability trend in a chart and uploads it to a *cloud platform*.

Some tools can be deployed outside of CI environments for developers, but additional budget and work could be expected to maintain the tool out of a CI environment for development teams.

Additionally, we noted that two NFRs may influence each other negatively. For example, client-server applications in a banking system could require extra security algorithms or protocols (Remlein and Stachowiak 2021) to exchange data across a network communication for better data integrity. When the system deals with a large number of user requests, the network performance may decline. CI-based metrics for this particular case can be used to catch relevant quality fluctuation and provide developers fast feedback, but significant quality drops or growths requires to be analyzed manually.

The next chapter moves on to investigate how individual and multiple components in a CI environment impact the NFR testing.

4.3 RQ3: How are CI Environments Contributing to NFR Test Capabilities?

We have examined the connection between CI components and NFR testing by referring to the data that we collected. Our examination is presented in terms of the following:

1. Software tools that enable automated NFR testing,
2. Individual CI components used for automated NFR testing,
3. Sets of CI components used for automated NFR testing.

4.3.1 Software Tools that Enable Automated NFR Testing

First, in each of the projects that we studied, we mapped the identified NFRs to the relevant CI tools based on participants' inputs stating which tools were used for a particular NFR type.

As shown in Table 7, each NFR type connects to multiple CI tools. Some tools can enable specific NFR tests, whilst others support NFR evaluation. We observed that using tools can positively impact a test's effectiveness with respect to the implementation effort for NFR metrics. For instance, the Postman tool was used to enable more effective performance tests. Other tools were used to make NFR testing more efficient.

Table 7 A mapping between the identified NFR types and software tools

Types of NFRs	Tools enable NFR testing	Tools support NFR testing
Performance	Postman, JMeter, Junit	Bitbucket, VMware, Jenkins, Jfrog artifactory
Security	ZAP, Anchore, SonarQub	Jenkins, Jfrog artifactory
Scalability	JMeter	Jenkins, VMware
Maintainability	JMeter	Jenkins, VMware
Stability	Junit	Jenkins, VMware

4.3.2 Individual CI Components Used for Automated NFR Testing

To discover which CI components (Yu et al. 2020) have been adopted for automated NFR testing, we analyzed the extracted data using the following steps:

1. List the identified types of NFRs and test tools that were identified.
2. Group the tools into CI tool categories.
3. Map the identified categories of CI tools to CI components (Yu et al. 2020).

CI components, such as *continuous integration server*(CIS), *artifacts management*(AM), and *static code analysis*(SCA), can enable particular NFR evaluation. For example, in Project D, CIS component used the ZAP plugin to scan security vulnerabilities. In Project A, AM component evaluated security issues from a snapshot or release package, thereby enabling security testing. CIS component was instrumental in enabling the automated execution of NFR test cases. In Project B, SCA component verified maintainability issues in the source code.

Other CI components, such as *version control system*, *source code management*, *cloud platform*, and *issue tracking* can be used to efficiently manage the source codes of NFR test cases with version-control efficiently, maximize the use of hardware resources (for example, CPU, memory, and disk space), and track test results.

Some components (for example, *static code analysis*) can be deployed as a standalone application. However, to maximize the capabilities of test tools and to acquire optimal value, it is essential that they can be integrated into a CI environment.

4.3.3 Sets of CI Components for Automated NFR Testing

A single CI component can generate limited value for automated NFR testing. For instance, the *static code analysis* component can enable automated security tests and provide feedback regarding the level of a source code, but this component cannot provide feedback on a service or system level.

Several CI components packaged together in an environment can provide better conditions for automated NFR testing. For example, CIS, AM, and cloud platform components were used in Project C to enable and support system-level security tests. These components thus expand on automated security tests, from the source code to the system level.

A package of CI components brings several benefits to automated testing. Larger CI environments may well be associated with more complexity, but more capability in terms of automated testing can be achieved in return. Some NFRs (for example, performance in the projects studied here) require the presence of five different CI components in order to be fully automated. Note that the same CI components can also be used to evaluate the other NFRs (for example, *security*, *scalability*, and *stability*). This finding suggests that the larger the number of CI components that a project uses, the more capabilities it has for automated NFR testing.

4.4 RQ4: What Challenges are Associated with Automated NFR Testing?

This section presents the challenges for NFR testing as elicited in our study. The main challenges that practitioners described were difficulties in automating NFR tests, barriers to analyzing NFR issues, and challenges with NFR monitoring. We also reflect on the findings of the NFR testing and explain how these challenges can be mitigated.

Challenge 1: NFR test-automation could be difficult for products with complex dependencies between sub-modules. Dependencies exist among system components due to the cross-cutting nature of NFRs. As a practitioner stated, which was also confirmed by the others, "For many cases, when we succeed in making an NFR work well for a system module, unexpected issues appear at some point in another module which is annoying." To mitigate this issue, a higher frequency of software integration through CI environments is needed, which could increase the chances of catching potential interoperability issues as early as possible in the development stage. This finding is consistent with a similar issue reported by Karhapää et al. (2021).

Challenge 2: Resolving NFR issues may require joint teams' effort. Difficulties related to NFR root-cause analysis were reported in all studied projects. As the organizations in our study split NFRs into smaller parts and distributed them to many teams, there were risks that the debugging of NFR issues would be complex while teams were growing. When a fault is found, the root cause analysis is slowed down. To mitigate this challenge, CI components (for example, *issue tracking*) can be used to narrow down NFR matters by using CI test reports, which means practitioners can identify NFR-related issues earlier with the support of CI. This challenge was also reported by Alsaqaf et al. (2019); however, Alsaqaf et al. focused on strategies to manage quality requirements instead of automated testing.

Challenge 3: it is challenging to track the NFR development life-cycle. When we asked about the NFR development life-cycle, practitioners were unanimous in the view that there are lacking information on how to track NFRs. One reason could be that tacking NFRs concerns software design, implementation, testing, and release. Manual work to monitor overall NFRs' state from source codes to release packages might be executable in some cases, but doing so could introduce human faults or inconsistent quality reports. Therefore, monitoring NFRs through CI environments is beneficial, confirmed by prior studies, such as López et al. (2022) and Sas and Avgeriou (2020). Our study results show that CI components can produce data from products' source files to support NFR testing. Moreover, practitioners reported that developers add requirements' index IDs in their source code commits to increase traceability, and the index IDs are included in the release note of a software delivery. Combining the study results and practitioners' feedback, a potential solution to mitigate this challenge is adding the requirements' ID in each source code commit and using CI to collect and monitor this ID across requirements, implementation, test reports, and release packages.

Challenge 4: CI environments' stability affects NFR testing. As a practitioner stated that "unstable CI environments fail NFR tests more often." Hardware resources (e.g., CPU, memory, disk) in a CI environment and network issues between CI components could cause NFR tests to fail, which may result in many false positives in test reports. Such issues become more sensitive for NFR testing, as some NFR evaluation (e.g., system latency) depends on a reliable test environment. Addressing this kind of issue is challenging since a CI environment contains different components and tools with complex data traffic, event-based interactions, and network configurations. Further research that investigates how to maintain stable CI environments for better NFR testing is needed since limited empirical studies were found in this research area.

Challenge 5: NFR test-report analysis is difficult to be performed. We examined multiple automated NFR tests executed in CI environments and where the test results were sent back to the developers. However, the NFR test results were not stored for aggregated visualization and analysis due to lack of knowledge. As a practitioner mentioned that "it is hard to know what information to gather while looking over NFR test reports." Over half of the practitioners confirmed this issue. Practitioners' feedback shows the importance of having a dashboard for storing and analyzing NFR test data, which helps improve system qualities continuously. Therefore, further study with more focus on collecting and analyzing NFR test results is suggested.

5 Validity Threats

We discuss several threats to the validity of our study and present the steps that we took to mitigate these threats.

Internal validity: The following measures were taken to ensure the internal validity of the results. We followed the case study guidelines presented by Runeson and Höst (2009). The interview questions were thoroughly tested and validated to (i) improve their quality (for example, by improving the consistency of the questions and reducing their ambiguity) and (ii) address potential ethical issues (for example, obtaining informed consent, and ensuring data privacy and data anonymity).

Although an interview guide was used by the researchers in this study, some of the interview questions could have been misinterpreted, leading to inaccurate answers that could have introduced errors or other flaws in the data. The questions were included in the interview guide were carefully reviewed and tested by industrial practitioners and the researchers to mitigate this threat. The study results were also verified by sending the study results to the practitioners.

External validity: We investigated different software development companies that contain small, medium, and large software projects from diverse business domains, and all the studied companies operate in different contexts. However, the companies were delimited to those in the Nordic countries. This delimitation influences the generalizability of the results, since geographic location and culture affects work practices. As such, we make no claims, although possible, of the applicability of the presented results outside the Nordic countries.

We strategically selected a diverse range of industrial projects to increase the probability that the data set that was ultimately selected is representative of as many software development companies as possible. However, our data sample is small, therefore, it is important to consider that some projects might use CI environments for NFR testing in other business domains, thereby providing us with supplementary data on automated NFR tests.

Construct validity: Since the aim of this study is to explore NFR testing using CI components in industrial practice, we studied the phenomenon in multiple industrial contexts. All of the participants were screened before we conducted the interviews to ensure that they were suitable candidates for the study. Consequently, we sought to, but never employed, exclusion of employees who did not know CI, who had never done testing before, and also new employees. Another risk was that participants would not want to report on the challenges that the company currently has, which was mitigated by assuring that all collected information would be anonymous and confidential.

Conclusion validity: Three researchers independently devised coding schemes that were used to transcript interview recordings. The coding schemes were discussed to form a common understanding of how the transcripts were coded. However, the synthesis of collected codes is more subjective, and a different set of researchers could generate different results. To mitigate this threat, the results were shared with key actors who had knowledge about the CI and NFR testing in each company. Additionally, the participants reviewed this manuscript to validate that the included information was accurate.

6 Discussion

We have investigated practices for automated NFR evaluation in CI environments and related challenges in software development companies. The practices could help practitioners for their decisions concerning (i) automate NFR evaluation in a CI environment or (ii) improve existing CI environments and NFR testing capabilities. Based on the findings, we discuss the implications for practitioners and researchers and result generalization.

6.1 Implications for Practitioners

Our findings have implications for developers and testers working in software development companies who struggle with NFR quality improvements. For example, most of the participants in this study reviewed our theoretical model and reported that the model we developed improved their ability to understand how CI can contribute to NFR testing. This suggests that our findings could be effective for other engineers to be aware that there are great potential capabilities in CI environments to be used for better-automated NFR testing. Our findings confirm prior publications, such as Garousi et al. (2017) and Wang et al. (2022). The results of identified NFR metrics suggest that using CI data to measure NFRs could lead to reliable and consistent test results. These metrics could be easily implemented in various organizations, including small businesses and large corporations. It is important to note that our study was conducted in limited cases, meaning many other metrics may be used for the same purposes in some cases.

We observe that multiple metrics can evaluate a particular NFR from different perspectives. For example, two metrics, like UCA and VA in Table 6, were used for security testing, and participants from project B stated that positive security impacts were generated by using them. This indicates that using metrics may increase the possibility of detecting issues for developers since different NFR metrics offer a broader view to measure the testing of a specific NFR. Metrics can transform vague NFR evaluation into concrete methods, including measurement definitions (López et al. 2022), formulas, and input data. This finding agrees with the results reported by Colakoglu et al. (2021). However, we did not find detail impacts of using multiple metrics for quality attributes' trade-offs in the previous publications, and more empirical studies in this research area are needed.

We observe that international standards for quality measurements have been used to define specific NFR metrics in our study. For example, in Project B, ISO/IEC 25023 (ISO/IEC-25023 2016) standards were used to define the metric named MRET in Table 6 to measure system scalability. A few participants mentioned that using international standards can reduce the effort of test design and implementation, as the standards already provide detailed steps to define metrics (ISO/IEC-25023 2016). We are not sure why the other selected projects did not use international standards, but there are a few possible explanations. One possibility is that the product in Project B is large and mature serving global customers that may comply with specific standards in their requirements. This finding confirms the value of using international standards for NFR evaluation in the industry.

Furthermore, we learned that NFR priorities impact software quality while interviewing participants. For example, in the studied projects, *security* and *performance* were validated in priority, as they reflect the quality of external interfaces in a system and are visible to customers. While some NFRs, like *maintainability*, are mainly visible to the developer(s) and are thus neglected during testing or in low priority. This suggests that our findings could help practitioners be aware of the importance of reviewing the NFR test plan with relevant stakeholders. As confirmed by Werner et al. (2020), the review allows developers to have a common understanding (Werner et al. 2020) of NFR goals, which helps implement automated tests and spot test obstacles. Moreover, NFRs include many quality attributes, making it difficult to develop automated tests covering all quality attributes. A guide for NFR prioritization can be found in the model presented by Svensson et al. (2011).

6.2 Implications for Researchers

From a *research* perspective, implications for researchers include:

– The study provides insights into the implementation of NFR verification in CI environments, offering a foundation for further research in this area. Researchers can build upon these findings to investigate more extensive connections between CI components, NFR metrics, and challenges, exploring additional possibilities that may exist beyond those identified in this study.
– Further investigation is needed to explain why certain beneficial NFR metrics, which have potential to improve software quality in companies, are not used. This opens opportunities for future research to explore the barriers or reasons behind the underutilization of such metrics, providing a deeper understanding of the factors influencing their adoption.
– The observation that ISO/IEC 25023 standards were utilized in defining specific NFR metrics in one of the studied projects emphasizes the importance of exploring the role of international standards in NFR evaluation. Future research could explore into the benefits and challenges of adopting international standards for the evaluation.

6.3 Result Generalization

We adopt case-based generalization (Wieringa and Daneva 2015) strategies to make inferences about applying study findings in wider software development companies, as the sample for this study consisted of a small group of companies. Ghaisas et al. (2013) have reported seven main lessons learned while generalizing the findings from three industrial case studies to a larger population by examining software components' architectural similarity.

Our inference of generalizing by similarity contain below steps:

1. Triangulation: we collect data from multiple sources and use a thematic coding method transforming CI-related data into a component-based architectural view and NFR evaluation into specific metrics to improve the ability to generalize study findings.
2. Comparative analysis: we compare the findings of the studied projects to identify common CI components/tools and NFR metrics to generate more generalizable conclusions.
3. Participant feedback: we share study results with participants and collect their reviews to validate study findings, which aim to increase the generalizability of the study results.

4. Contextual analysis: we examine the contexts of all studied industrial projects (see Table 1) to identify the factors that may influence the study findings and consider their potential impact on the generalizability.

Based on our analysis of the selected cases, we identified several common themes related to the impact of NFR testing. These themes include the maturity of the CI environment (Garousi et al. 2017) (e.g., immature, growing, expansion, mature), the size of a project (Petersen and Wohlin 2009) (e.g., small, medium, large), and the number of engineers (Petersen and Wohlin 2009). Despite the limited scope of our study, which examined four companies, including two international ones, we believe that the results have wider implications for software development companies that share similar characteristics and themes presented in Table 1. Further research is needed to confirm and extend our findings.

7 Conclusion and Future Work

This study aims to investigate the use of CI environments for automated NFR testing. Through a multi-case study in software development companies, the findings provide a comprehensive overview of the NFR testing practices including test tools, CI components, and metrics used in the industry.

The study results indicate that utilizing metrics with data generated by CI environments can improve the NFR testing. We have found that a variety of metrics using data produced by CI environments support NFR testing through this study. The uses of metrics and CI environments allow for the testing in automated processes and early detection of NFR issues.

However, we identified several challenges associated with using CI environments in NFR testing. These challenges include a lack of knowledge of verifying system qualities through CI environments, difficulties in NFR debugging, and missing issue-tracking processes.

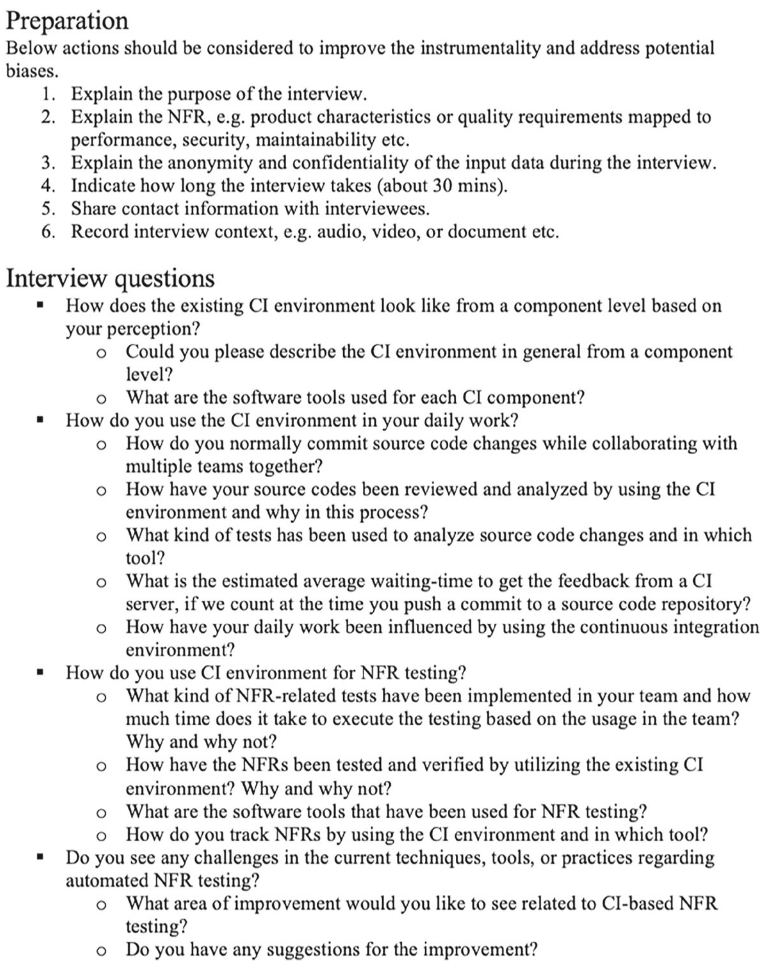
To enhance understanding of the specific contributions of CI to NFR testing, we proposed a theoretical model that reveals the potential of CI components to generate data from product artifacts, which can be used to generate metrics to support NFR evaluation. This model serves as a useful tool for practitioners to understand how metric data generated by CI components, and metric outcomes can be shared and visualized through notifications and quality trends enabled by CI, which are helpful to mitigate the aforementioned challenges of NFR debugging and issue tracking.

Overall, the study's findings suggest that while CI environments can provide many benefits for the NFR testing, they also come with challenges. With the proposed model, which provides understanding of how data from CI components can support NFR metrics, organizations can evolve their existing CI environments to achieve better test processes.

We recognize the limitations of our study, although we examined a heterogeneous set of companies. Additional data collection from more case studies in other domains and contexts is required in the future. Such data would help us better understand the CI environment and identify more NFRs that can undergo CI-based testing.

In future work, we intend to (i) focus on the role of CI components and (ii) implement probes using machine-learning algorithms to deal with NFR test data. The data can then be used to (iii) suggest CI improvements in achieving data-driven testing. This follows from the idea that each CI component produces unique data that can be of value for NFR testing if this data is collected throughout the CI cycle.

Appendix A: Interview Guide



Preparation

Below actions should be considered to improve the instrumentality and address potential biases.

1. Explain the purpose of the interview.
2. Explain the NFR, e.g. product characteristics or quality requirements mapped to performance, security, maintainability etc.
3. Explain the anonymity and confidentiality of the input data during the interview.
4. Indicate how long the interview takes (about 30 mins).
5. Share contact information with interviewees.
6. Record interview context, e.g. audio, video, or document etc.

Interview questions

- How does the existing CI environment look like from a component level based on your perception?
  - Could you please describe the CI environment in general from a component level?
  - What are the software tools used for each CI component?
- How do you use the CI environment in your daily work?
  - How do you normally commit source code changes while collaborating with multiple teams together?
  - How have your source codes been reviewed and analyzed by using the CI environment and why in this process?
  - What kind of tests has been used to analyze source code changes and in which tool?
  - What is the estimated average waiting-time to get the feedback from a CI server, if we count at the time you push a commit to a source code repository?
  - How have your daily work been influenced by using the continuous integration environment?
- How do you use CI environment for NFR testing?
  - What kind of NFR-related tests have been implemented in your team and how much time does it take to execute the testing based on the usage in the team? Why and why not?
  - How have the NFRs been tested and verified by utilizing the existing CI environment? Why and why not?
  - What are the software tools that have been used for NFR testing?
  - How do you track NFRs by using the CI environment and in which tool?
- Do you see any challenges in the current techniques, tools, or practices regarding automated NFR testing?
  - What area of improvement would you like to see related to CI-based NFR testing?
  - Do you have any suggestions for the improvement?

Appendix B: Primary Codes Extracted from Interviews

ID	Code name	Description	Interviewee ID
1	Performance tests	Statements about both manual and automated tests on performance.	T2,T4,T5,T7,T8,T11,T12,T13, T14,T15,T17,T18,T19,T20,T22
2	Security tests	Statements about both manual and automated tests on the security.	T1,T4,T5,T6,T7,T8,T9,T10,T13, T14,T15,T17,T18,T19
3	Upgrade tests	Statements about the test of upgrading a system.	T3,T10
4	Maintainability	Statements relating to debug and repair.	T9,T11,T12
5	Stability tests	Statements about the test of a system load in a limited time period.	T3,T10,T12
6	Scalability	Statements relating to the high availability of services in production.	T2,T5,T9,T12
7	Fault tolerance	Statements about a system continue to operate well in the event of a failure.	T12,T14
8	Automated testing	Statements about test cases that executed by using CI jobs.	T1,T2,T3,T4,T5,T6,T7,T8,T9,T12, T14,T16,T18,T19,T20,T21,T22
9	Source codes	Statements related to codes of test cases e.g., JAVA, Python, C++, etc.	T1,T2,T3,T4,T5,T6,T14,T15,T16, T17,T18,T19,T20,T21,T22
10	Code review	Statements about the code reviews.	T1,T2,T3,T4,T5,T6,T7,T8,T9, T14,T15,T17,T18,T20,T21,T22
11	Version control	Statements about the version control system in a CI environment.	T1,T2,T3,T7,T8,T14,T15,T17, T18,T19,T20,T21,T22
12	Continuous integration	Statements about continuous integration solution or strategy.	T1,T2,T4,T5,T6,T7,T8,T9,T13, T14,T15,T17,T18,T19,T20,T21,T22
13	Code analysis	Statements about the static code analysis.	T1,T2,T3,T4,T5,T6,T9,T12,T13, T14,T15,T20,T21
14	Artifacts management	Statements about how to handle build snapshots or release packages.	T2,T3,T5,T9,T10,T11,T12,T13,T16 T17,T18,T19,T20,T22
15	Issue tracking	Statements on how to handle failed tests.	T2,T3,T4,T5,T6,T7,T8,T10,T11, T14,T15,T20,T21,T22
16	Feedback time	Statements about the average execution time for CI jobs.	T1,T2,T3,T4,T5,T6,T7,T8,T9,T13, T14,T15,T17,T18,T19,T20,T21
17	Gerrit Bitbucket	Statements about a tool to manage source codes or files.	T1,T2,T3,T4,T5,T6,T14,T15,T16, T17,T18,T19,T20,T21,T22
18	Jenkins	Statements about a CI tool.	T1,T2,T3,T4,T5,T6,T7,T8,T9,T10, T11,T12,T17,T18,T19,T20,T21,T22

ID	Code name	Description	Interviewee ID
19	Xray ZAP	Statements about a tool to check security vulnerability risks.	T2,T3,T4,T5,T7,T8,T10,T11 T17,T18,T19
20	Postman JMeter	Statements about a tool to test performance of REST API requests.	T2,T4,T5,T7,T8,T11,T12,T13, T14,T15,T17,T18,T19,T20,T22
21	SonarQube	Statements about the SonarQube for scanning source codes.	T3,T7,T9,T10,T11,T12,T14,T15 T20,T21
22	Jfrog Artifactory	Statements about a tool to handle build artifacts.	T2,T3,T5,T9,T10,T11,T12,T13,T16 T17,T18,T19,T20,T22
23	Kubernetes	Statements relating to the technology to manage containerized applications.	T2,T3,T6,T7,T9,T12,T13,T14,T15
24	VMware AWS	Statements about a cloud platform to manage hardware resources.	T2,T3,T5,T9,T10,T11,T12,T13,T16 T17,T18,T19,T20,T22
25	Trouble report JIRA	Statements about the tools to report bugs or issues.	T7,T8,T9,T10,T11,T12,T14,T15 T17,T18,T20,T22
26	Ansible Vagrant	Statements about a tool to automate processes or events.	T2,T3,T6,T7,T9,T12,T13,T14,T15

Acknowledgements We acknowledge support from the KKS Foundation through the S.E.R.T. Research Profile Project and the KKS PLEng 2.0 at Blekinge Institute of Technology.

Funding Open access funding provided by Blekinge Institute of Technology.

Declarations

Conflicts of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Aljallabi, BM, Mansour, A: Enhancement approach for non-functional requirements analysis in agile environment. In: 2015 international conference on computing, control, networking, electronics and embedded systems engineering (ICCNEEE), pp 428–433. IEEE (2015)

Alsaqaf W, Daneva M, Wieringa R (2019) Quality requirements challenges in the context of large-scale distributed agile: An empirical study. Information and software technology 110:39–55

Boehm, B, Basili, VR: Software defect reduction top 10 list. Foundations of empirical software engineering: the legacy of Victor R. Basili **426**(37), 426–431 (2005)

Bougouffa, S, Diehm, S, Schwarz, M, Vogel-Heuser, B: Scalable cloud based semantic code analysis to support continuous integration of industrial plc code. In: 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), pp 621–627. IEEE (2017)

Cajander, Å, Larusdottir, M, Gulliksen, J: Existing but not explicit-the user perspective in scrum projects in practice. In: IFIP Conference on Human-Computer Interaction, pp 762–779. Springer (2013)
Cannizzo, F, Clutton, R, Ramesh, R: Pushing the boundaries of testing and continuous integration. In: Agile 2008 Conference, pp 501–505. IEEE (2008)
Chen L (2015) Continuous delivery: Huge benefits, but challenges too. IEEE software 32(2):50–54
Chung, L, do Prado Leite, JCS: On non-functional requirements in software engineering. In: Conceptual modeling: Foundations and applications, pp 363–379. Springer (2009)
Colakoglu FN, Yazici A, Mishra A (2021) Software Product Quality Metrics: A Systematic Mapping Study. IEEE Access 9:44647–44670. https://doi.org/10.1109/ACCESS.2021.3054730, https://ieeexplore.ieee.org/document/9336003/
Corbin JM, Strauss A (1990) Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative sociology 13(1):3–21
Cruzes, DS, Dyba, T: Recommended steps for thematic synthesis in software engineering. In: 2011 International Symposium on Empirical Software Engineering and Measurement, pp 275–284. IEEE (2011)
Dlugi, M, Brunnert, A, Krcmar, H: Model-based performance evaluations in continuous delivery pipelines. In: Proceedings of the 1st International Workshop on Quality-Aware DevOps, pp. 25–26. ACM (2015)
Fitzgerald B, Stol KJ (2017) Continuous software engineering: A roadmap and agenda. Journal of Systems and Software 123:176–189
Garousi V, Felderer M, Hacaloglu T (2017) Software test maturity assessment and test process improvement: A multivocal literature review. Information and Software Technology 85:16–42. https://doi.org/10.1016/j.infsof.2017.01.001, . https://linkinghub.elsevier.com/retrieve/pii/S0950584917300162
Ghaisas, S, Rose, P, Daneva, M, Sikkel, K, Wieringa, RJ: Generalizing by similarity: Lessons learnt from industrial case studies. In: 2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI), pp 37–42. IEEE (2013)
Gorschek T, Wohlin C (2006) Requirements abstraction model. Requirements Engineering 11(1):79–101
Gregor, S: The nature of theory in information systems. MIS quarterly pp 611–642 (2006)
ISO/IEC-25023: Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): Measurement of System and Software Product Quality. ISO (2016)
Janus, A, Schmietendorf, A, Dumke, R, Jäger, J: The 3c approach for agile quality assurance. In: Proceedings of the 3rd international workshop on emerging trends in software metrics, pp 9–13. IEEE Press (2012)
Júnior, MC: Automated verification of compliance of non-functional requirements on mobile applications through metamorphic testing. In: 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp 421–423. IEEE (2020)
Karhapää P, Behutiye W, Rodríguez P, Oivo M, Costal D, Franch X, Aaramaa S, Choraś M, Partanen J, Abherve A (2021) Strategies to manage quality requirements in agile software development: a multiple case study. Empirical Software Engineering 26(2):1–59
Khurum M, Petersen K, Gorschek T (2014) Extending value stream mapping through waste definition beyond customer perspective. Journal of Software: Evolution and Process 26(12):1074–1105
Knauss, E, Pelliccione, P, Heldal, R, Ågren, M, Hellman, S, Maniette, D: Continuous integration beyond the team: a tooling perspective on challenges in the automotive industry. In: Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement, p 43. ACM (2016)
López L, Burgués X, Martínez-Fernández S, Vollmer AM, Behutiye W, Karhapää P, Franch X, Rodríguez P, Oivo M (2022) Quality measurement in agile and rapid software development: A systematic mapping. Journal of Systems and Software 186:111187
Mairiza, D, Zowghi, D, Nurmuliani, N: An investigation into the notion of non-functional requirements. In: Proceedings of the 2010 ACM symposium on applied computing, pp 311–317. ACM (2010)
Miller, A: A hundred days of continuous integration. In: Agile, 2008. AGILE'08. Conference, pp 289–293. IEEE (2008)
Nouacer R, Djemal M, Niar S, Mouchard G, Rapin N, Gallois JP, Fiani P, Chastrette F, Lapitre A, Adriano T et al (2016) Equitas: A tool-chain for functional safety and reliability improvement in automotive systems. Microprocessors and Microsystems 47:252–261
Paixão, KV, Felício, CZ, Delfim, FM, de A Maia, M: On the interplay between non-functional requirements and builds on continuous integration. In: Proceedings of the 14th international conference on mining software repositories, pp 479–482. IEEE Press (2017)
Petersen, K, Wohlin, C: Context in industrial software engineering research. In: 2009 3rd international symposium on empirical software engineering and measurement, pp 401–404. IEEE (2009)
Rathod, N, Surve, A: Test orchestration a framework for continuous integration and continuous deployment. In: Pervasive Computing (ICPC), 2015 International Conference on, pp 1–5. IEEE (2015)

Rehmann KT, Seo C, Hwang D, Truong BT, Boehm A, Lee DH (2016) Performance monitoring in sap hana's continuous integration process. ACM SIGMETRICS Performance Evaluation Review 43(4):43–52

Remlein, P, Stachowiak, U: Security verification in the context of 5g sensor networks. Journal of telecommunications and information technology (2021)

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2):131

Sas D, Avgeriou P (2020) Quality attribute trade-offs in the embedded systems industry: an exploratory case study. Software Quality Journal 28(2):505–534

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

Staron M, Meding W, Karlsson G, Nilsson C (2011) Developing measurement systems: an industrial case study. Journal of Software Maintenance and Evolution: Research and Practice 23(2):89–107

Svensson, RB, Gorschek, T, Regnell, B, Torkar, R, Shahrokni, A, Feldt, R, Aurum, A: Prioritization of quality requirements: State of practice in eleven companies. In: 2011 IEEE 19th international requirements engineering conference, pp 69–78 (2011). https://doi.org/10.1109/RE.2011.6051652

Wagner, S: A literature survey of the quality economics of defect-detection techniques. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, pp 194–203 (2006)

Wang Y, Mäntylä MV, Liu Z, Markkula J (2022) Test automation maturity improves product quality-Quantitative study of open source projects using continuous integration. Journal of Systems and Software 188:111259. https://doi.org/10.1016/j.jss.2022.111259, . https://linkinghub.elsevier.com/retrieve/pii/S0164121222000280

Werner, C, Li, ZS, Ernst, N., Damian, D.: The lack of shared understanding of non-functional requirements in continuous software engineering: Accidental or essential? In: 2020 IEEE 28th international requirements engineering conference (RE), pp 90–101. IEEE (2020)

Werner, C, Li, ZS, Lowlind, D, Elazhary, O, Ernst, NA, Damian, D: Continuously managing nfrs: Opportunities and challenges in practice. IEEE Transactions on Software Engineering (2021)

Wieringa R, Daneva M (2015) Six strategies for generalizing software engineering theories. Science of Computer Programming 101:136–152

Wohlin, C, Runeson, P, Höst, M, Ohlsson, MC, Regnell, B, Wesslén, A: Experimentation in software engineering. Springer Science & Business Media (2012)

Yu L, Alégroth E, Chatzipetrou P, Gorschek T (2020) Utilising ci environment for efficient and effective testing of nfrs. Information and Software Technology 117:106199

Liang Yu is a Ph.D student at Blekinge Institute of Technology. His research has been focused on automated testing and software CI/CD processes. He has twelve years of industrial experience as a software developer in Sweden.

Emil Alégroth is a senior lecturer at Blekinge Institute of Technology. His research has been focused on automated testing, in particular GUI test automation, with several impactful publications in the area. Emil's research has been primarily empirical in nature, conducted in co-production with industry with companies such as Saab, Ericsson and Spotify. He has also operated for several years as the CEO of a company developing test solutions for industry and co-founded companies in the domain of software testing.

Panagiota Chatzipetrou Chatzipetrou is an Associate Professor (docent) at the department of Informatics at Örebro University in Örebro, Sweden. As a researcher, she mainly focuses on empirical studies under the different perspectives of software development. Her research interests include - but are not limited to- applications of statistical methods to quality problems in software engineering and especially to requirements engineering and the exploitation of human factor and the different views that ultimately determine the quality of a software product and the product development. Also, she has been working with decision support systems for the development of software-intensive systems, large-scale agile (and global) software development, and behavioral software engineering. She has collaborated with a number of different Universities in Sweden and in Europe.

Tony Gorschek is a Professor of Software Engineering at Blekinge Institute of Technology (Sweden) and part time at Chalmers University. He has over fifteen years of industrial experience as a CTO, senior executive consultant and engineer, but also as chief architect and product manager. In addition, he has built up six startups in fields ranging from logistics to Internet based services and algorithmic stock trading. Currently he manages his own consultancy company, works as a CTO, and serves on several boards in companies developing cutting edge technology and products. His research interests include empirical software engineering, engineering security, technology and product management, process assessment and improvement, quality assurance, and value based lean development of software intensive products and services.