Information and Software Technology 117 (2020) 106199

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Utilising CI environment for efficient and effective testing of NFRs☆

Liang Yu[a,b,*], Emil Alégroth[b], Panagiota Chatzipetrou[c], Tony Gorschek[b]

[a] Qvantel Sweden AB, Campus Gräsvik 12, Karlskrona 371 80, Sweden
[b] Blekinge Institute of Technology, Karlskrona, Sweden
[c] Department of Informatics, CERIS, Örebro University School of Business, Örebro SE-701 82, Sweden

ARTICLE INFO

Keywords:
Agile
Continuous integration
CI
DevOps
Non-functional requirement
NFR
Scaled agile framework
SAFe

ABSTRACT

Context: Continuous integration (CI) is a practice that aims to continuously verify quality aspects of a software intensive system both for functional and non-functional requirements (NFRs). Functional requirements are the inputs of development and can be tested in isolation, utilising either manual or automated tests. In contrast, some NFRs are difficult to test without functionality, for NFRs are often aspects of functionality and express quality aspects. Lacking this testability attribute makes NFR testing complicated and, therefore, underrepresented in industrial practice. However, the emergence of CI has radically affected software development and created new avenues for software quality evaluation and quality information acquisition. Research has, consequently, been devoted to the utilisation of this additional information for more efficient and effective NFR verification.

Objective: We aim to identify the state-of-the-art of utilising the CI environment for NFR testing, hereinafter referred to as CI-NFR testing.

Method: Through rigorous selection, from an initial set of 747 papers, we identified 47 papers that describe how NFRs are tested in a CI environment. Evidence-based analysis, through coding, is performed on the identified papers in this SLR.

Results: Firstly, ten CI approaches are described by the papers selected, each describing different tools and nine different NFRs where reported to be tested. Secondly, although possible, CI-NFR testing is associated with eight challenges that adversely affect its adoption. Thirdly, the identified CI-NFR testing processes are tool-driven, but there is a lack of NFR testing tools that can be used in the CI environment. Finally, we proposed a CI framework for NFRs testing.

Conclusion: A synthesised CI framework is proposed for testing various NFRs, and associated CI tools are also mapped. This contribution is valuable as results of the study also show that CI-NFR testing can help improve the quality of NFR testing in practices.

1. Introduction

Frequent integration and automation testing [1] in a rapid iteration to enhance software quality becomes an important competence in software development [2].

Continuous Integration (CI) [2,3] is widely adopted to improve software quality [4–8] through verifying and validating each change of a product in fast iterations by using automation tools and technologies [9].

By utilising the CI environment, developers' commits or pull requests are verified and validated continuously as early as possible against the requirements. Generally, a CI environment contains at least a CI server like Jenkins, a source code management system that hosts source code [10], and hardware infrastructure to execute several types of tests [11]. For example, first developers push a commit as a pull request to Github, then this pull request is tested by the Jenkins tool [10] on the Github infrastructure cloud. After the test, developers collect test results in the Github project.

Consequently, the developers get fast feedback about their changes and the quality of the source code, which prevents faults slipping through to the later stages of the software development lifecycle.

However, the existing CI environments mainly focus on software functionality in many software companies [2,5,8], but the non-functional requirements (NFRs), i.e. qualities of the system [12], are still mostly tested manually [13].

NFRs can be divided into internal and external qualities, where external NFRs include user experience, performance and security and internal NFRs include maintainability, extendability, and testability of the software [14]. Moreover, the functionality of a software system may not be usable without some necessary NFRs [12]. This implies that the qualities of the system, or NFRs if you will, provide at least as much, or even

☆ CI: Continuous Integration; NFRs: Non-Functional Requirements.

* Corresponding author at: Qvantel Sweden AB, Campus Gräsvik 12, 371 80 Karlskrona, Sweden.
E-mail addresses: liang.yu@bth.se (L. Yu), emil.alegroth@bth.se (E. Alégroth), panagiota.chatzipetrou@oru.se (P. Chatzipetrou), tony.gorschek@bth.se (T. Gorschek).

https://doi.org/10.1016/j.infsof.2019.106199
Received 10 May 2019; Received in revised form 28 August 2019; Accepted 10 October 2019
Available online 11 October 2019
0950-5849/

Table 1
The overview of related work.

Year	Paper	Focused area
2014	Stahl and Bosch [15]	Continuous integration
2015	Rodríguez et al. [16]	Continuous deployment
2016	Laukkanen et al. [17]	Continuous integration Continuous delivery
2017	Shahin et al. [9]	Continuous integration Continuous delivery Continuous deployment

more, value than the functionality to customers and developers. Thus, making research into better quality evaluation, analysis and assurance of importance for industrial practice.

According to the research of Mairiza et al. [14], the most commonly considered NFRs in the academic literature contain performance, reliability, usability, security, and maintainability, as shown in Table 2. However, the list of NFR types is considerably larger, and given the importance of software quality, research into more types is warranted.

Some NFRs are difficult to test independently of having functionality, because we are not good at testing certain items as they are not one item or part of one functional part, but rather on an overall system level (e.g. performance) [11]. Some NFRs are just hard to test (e.g. hedonic aspects, usability), and they are not easy to automate from the quality assurance perspective. There is research that suggests that NFRs can efficiently and effectively be tested by making use of a CI environment [13], but to our knowledge no comprehensive study has been performed to evaluate the state-of-the-art (SOTA) on the topic.

The underutilisation of the CI environment for NFR testing, in continuation referred to as CI-NFR testing, and its confirmed benefits [13] in terms of effectiveness and efficiency indicates that more work should be done on the NFRs' aspects. This leads us to investigate what CI approaches and tools are effective to evaluate NFRs; which industrial practices and challenges have been reported and documented in this area. To understand the existing knowledge of using a CI environment and associated tests to verify NFRs, we conducted this Systematic Literature Review (SLR). Furthermore, to clarify, in this study we are not interested in the NFRs of the CI environment itself, but the NFRs of a system or project which is developed using a CI environment.

This paper is organised as follows: Section 2 introduces related work. Section 3 illustrates the systematic literature review (SLR) method. Section 4 presents the results of SLR based on extracted data. Section 5 discusses the synthesis and the proposed CI framework for NFR testing. The conclusions and ideas for future work are in Section 7.

2. Related work

We identified four existing SLRs related to the topic of our SLR as shown in Table 1. These SLRs, being trinary data, were not included in our selected papers since we focused on the secondary data.

The four SLRs in Table 1 were summarised as follows:

In 2014 Stahl and Bosch [15] investigated software continuous integration (CI) and proposed a descriptive model by focussing on CI processes for industrial practices. They mentioned that non-functional system tests are a part of the CI automation testing phase, but no further analysis about how to measure and evaluate non-functional requirements in system tests on CI environments was reported.

In 2015 Rodríguez et al. [16] studied 50 primary studies from the year 2001 to 2014 about software continuous deployment and presented a set of benefits and challenges of adopting it.

- Challenge 1: Transforming towards continuous deployment is a challenge, since it requires significant investments in deployment processes, changes of people's mindsets, and adjustments in working habits.
- Challenge 2: The increased quality assurance efforts required for continuous deployment are associated with significant challenges.
- Benefit 1: CI provides fast feedback in the development stage.
- Benefit 2: Customers' satisfaction increase when valuable products are continuously deployed.
- Benefit 3: Time-to-market decreased since the capability to release software increased.
- Benefit 4: Continuous testing and automation improved release reliability and quality.

In addition, this paper mentions the trade-off of NFRs in software architecture design, e.g. to improve security, some software performance might have to be sacrificed due to additional checks or encryption. However, the study does not report which NFRs were considered or evaluated by using the continuous deployment approach.

In 2016 Laukkanen et al. [17] conducted a systematic review focussing on problems, causes, and solutions while adopting continuous delivery (CD). They identified a total of 40 problems, 28 causal relationships and 29 solutions related to the adoption of continuous delivery [17]. The identified problems are related to software integration and testing while adopting CD in practice, but not NFRs. In our SLR we focus on NFR testing by using CI environments from the technical perspective.

In 2017 Shahin et al. [9] published a study on continuous practices including continuous integration (CI), continuous delivery (CD) and continuous deployment focussed on approaches, tools, challenges, and practices. They identified 30 continuous-practice approaches and associated tools from 69 selected papers between the year 2004 and 2016. Shahin et al. [9] emphasised to adopt CI, CD, and continuous deployment approaches, but we aimed to investigate the NFRs testing by using CI environments.

In summary, Stahl and Bosch [15] presented the practices of adopting a CI solution in an industrial case, which shows the CI is applicable. Further, Rodríguez et al. [16], Laukkanen et al. [17], and Shahin et al. [9] unveiled the problems and challenges of applying CI, CD and continuous deployment, which can tell that the CI adoption has both values and challenges. Moreover, Shahin et al. [9] identified a list of tools for CI, CD and continuous deployment, which points to the tooling perspective to evaluate the CI environment. The values, challenges, and tools of these SLRs are based on functional requirements' testing by using the CI environments. However, none of the previous studies investigated which CI environments and associated tests have been utilised for NFR testing.

3. Methodology

In this section we introduce our research questions, search strategies, inclusion and exclusion criteria, inter-rater reliability analysis, data collection, and synthesis.

3.1. Research processes overview

The research process is outlined in Fig. 1. First of all, we define the research goal and research questions. Secondly, we apply the search strings on well-chosen bibliographic sources. Thirdly, the inclusion and exclusion criteria is applied to refine the search results. At last, we analyse and synthesise the selected papers.

3.2. Definitions

In Section 3.3, both "***continuous integration environment***" and "***non-functional requirements***" are mentioned several times. There is, however, a lack of general definitions of these concepts and, for clarity this section will, therefore, provide the definitions we used in this SLR.

Continuous integration environment

A continuous integration environment is a platform delimited to technical perspective, which consists of a set of integrated and

Table 2
Commonly considered NFRs in the scholarly literature [14].

NFRs	Definition	Attributes
Performance	Requirements that specify the capability of a software product to provide appropriate performance relative to the amount of resources needed to perform full functionality under stated conditions [14].	response time, latency, resource utilisation, accuracy, efficiency compliance, data loss, concurrent transaction processing.
Reliability	Requirements that specify the capability of a software product to operate without failure and maintain a specified level of performance when used under specified normal conditions during a given time period [14].	accuracy, consistency, availability, integrity, correctness, maturity, fault tolerance, recoverability, compliance, failure rate.
Usability	Requirements that specify the end-user-interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system [14].	usability compliance, ease of use, user friendliness, efficiency, user productivity, user reaction time.
Security	Requirements that are concerned with preventing unauthorised access to the system, programs, and data [14].	confidentiality, availability, access control, authentication.
Maintainability	Requirements that describe the capability of the software product to be modified that may include correcting a defect or making an improvement or change in the software [14].	testability, modifiability, analysability, changeability, stability, maintainability compliance.

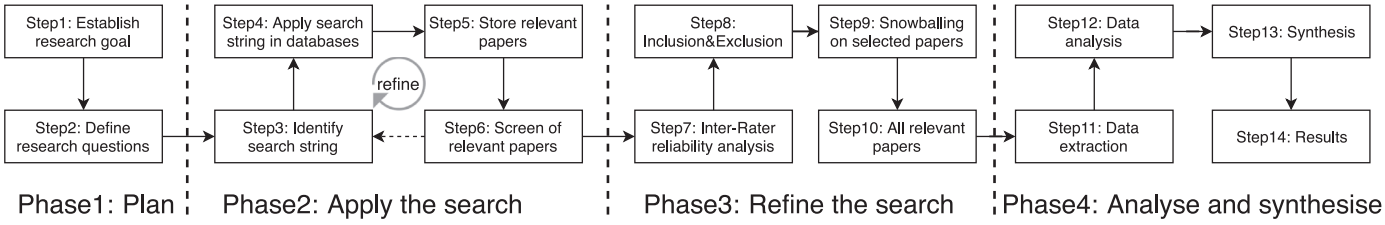

Fig. 1. Research processes overview.

configured tools [5]. It contains a pipeline process to automatically compile, build, and test software continuously.

Non-functional requirements

A non-functional requirement is defined as a property or quality attribute (e.g. performance, reliability, usability, security, maintainability) in this SLR, which a software system is designed to exhibit, or it is a constraint that the system has to comply with.

E2E

The E2E is defined as a CI pipeline flow which is from a software source code change to a software release candidate ready in software development.

3.3. Research goal and questions

The goal of this SLR is to find out how the CI environments can be used for NFR testing according to academia. This objective has been broken down into two research questions with sub-questions:

- RQ1: What is state-of-the-art (SOTA) in research for utilising continuous integration (CI) environments to test NFRs?
  - RQ1.1: What CI approaches have been proposed for NFRs testing in SOTA?
  - RQ1.2: What tools have been proposed or used in CI environments reported in SOTA?
- RQ2: What is state-of-the-practice (SOTP) in research for utilising continuous integration (CI) environments to test NFRs?
  - RQ2.1: What NFRs have been reported as testable by utilising a CI environment?

These two main research questions were separated into both SOTA and SOTP since we wanted to compare what academic studies state about CI-NFR testing to what actual industrial practices have been applied in production.

3.4. Search string and bibliographic sources

We piloted several experimental searches from April to May 2018 by following the steps in Fig. 1:

1. Identify a search string by using the keywords related to the research area, e.g. “continuous integration”, “non-functional requirement” etc.
2. Apply the search string in the selected bibliographic sources.
3. Store the search results.
4. Screen the relevant papers.

In order to cover the search scope better and describe the phenomenon appropriately, we iterated the above steps many times during the pilot searches and refined the search string based on the search results. The final search string is presented as follows:

> (“software”) AND (“continuous integration” OR “continuous deployment” OR “DevOps” OR “continuous delivery” OR “CI tools” OR “CI technology” OR “CI software” OR “CI cloud” OR “CI framework” OR “CI application” OR “CI system” OR “continuous software development”) AND (“quality attributes” OR “non-functional requirements” OR “nonfunctional requirements” OR “quality characteristics” OR “quality properties” OR “quality requirements”)

Four major bibliographic sources (**IEEE, Scopus, ACM**, and **Wiley**) were chosen in this SLR. These four bibliographic sources have sufficient coverage of the software engineering literature, are commonly used for software engineering SLRs, and contain “keywords” metadata search and “command search” with a search string, for instance, a search string: (“NFR” OR “quality attribute”) AND “software engineering.”

3.5. Search result

Firstly, we selected the “command search” in each chosen bibliographic source shown in Table 3 and applied the search string to titles, abstracts, and keywords. Then we exported the search result as CSV files from each bibliographic source and imported the CSV files into the

Table 3
Search results for each bibliographic source from 2008 to 2018.

Bibliographic source name	Number of papers
IEEE	137
Scopus	135
ACM	422
Wiley	76

Mendeley[1] tool in one folder. We selected all imported papers in Mendeley and used the "merge duplicated papers" feature to remove potential duplicates. As a result, the total number of papers was 747. The number of found papers in each bibliographic source is visualised in Table 3.

3.6. Inclusion and exclusion criteria

We delimited our SLR to papers published between 2008 to 2018 that had been peer-reviewed and written in the English language including conference papers, journals and book chapters. As stated previously, SLRs were excluded from the reviews. The inclusion criteria used during our SLR is defined in Table 4, and the exclusion criteria is shown in Table 5.

As we mentioned in Section 3.5, all papers of the search result from selected bibliographic sources were saved in a folder in the Mendeley tool. The inclusion and exclusion criteria were then applied in three levels. In level 1 we excluded all papers that did not fit the basic inclusion criteria (e.g. age, written in the English language). Next, in Level 2 we analysed the title and abstract, and, finally, in Level 3, we read the papers entirely. The level 1 inclusion/exclusion was performed manually by using Mendeley tool:

1. Check publish year column in Mendeley, and keep papers from 2008 to 2018 only.
2. Check paper type and delete non-peer-reviewed papers in Mendeley.
3. Keep workshops, conferences, journals, and book chapters, and remove the others.
4. Delete papers that have been published in a non-English language.
5. Move the papers containing "systematic literature review" in the title or abstract to a separate folder in Mendeley.

As a result of the level 1 inclusion/exclusion, 490 papers were kept, from a total of 747 papers, for further analysis.

3.7. Inter-Rater reliability analysis

To ensure that the inclusion and exclusion criteria are understandable and agreed by all involved researchers in this SLR, and thereby external researchers, we did inter-rater reliability analysis [18].

We used the Fleiss' Kappa coefficient [19–21] by following the guidelines from Altman [22] and Landis and Koch [23] to assess how strong the level of agreement was among the researchers.

The degree of agreement [23] shows if we managed to filter out the most relevant primary studies by using inclusion and exclusion criteria from the search result. Additionally, it validated whether the definitions in this SLR are precise and understandable by all researchers.

This analysis was performed by three researchers independently, and the steps are:

1. Pick out 10% of selected papers (49 out of 490) randomly by using the python random[2] library
2. Apply "*inclusion and exclusion criteria level 2 and 3*" on 49 randomly selected papers independently and separately from the participating researchers.
   - Read and understand the definitions in Section 3.2.
   - Read and understand the inclusion and exclusion description in Tables 4 and 5.
   - List all 49 randomly selected papers in an Excel file.
   - Give the value of "1," meaning a paper is included, otherwise "0," meaning a paper is excluded.
   - If both "*inclusion criteria level 2*" and "*inclusion criteria level 3*" is "1," then this paper is included, otherwise, the paper is excluded.
3. Collect all results from the participating researchers.
4. Calculate the coefficient value based on researchers' inclusion and exclusion results.

Fleiss' kappa [20] was run twice in this SLR. At the first round Fleiss' kappa $(\kappa) = 0.708, p < 0.05$ which represents a "good" strength of agreement among the researchers' opinions [22,23]. However, the analysis showed that there were misconceptions about the definition of the continuous integration environment, and it was consequently refined.

After the "continuous integration environment" definition was refined and discussed among the researchers, we performed the inter-rater agreement analysis again with a new set of randomly selected papers, once more with Fleiss' Kappa [20]. The result in the second run was $(\kappa) = 0.841, p < 0.05$, which represents "very good" strength agreement among our researchers' opinions [22,23].

This high agreement indicates that the inclusion and exclusion criteria are unambiguously defined such that the literature extraction could be replicated by other researchers with the same results.

3.8. Inclusion and exclusion result

The first author of this SLR applied the inclusion and exclusion criteria level 2 and 3 manually by following the steps below:

1. Manually read all exported papers' tiles, abstracts, and keywords in Menledey tool.
2. Apply inclusion and exclusion level 2.
3. Move all level 2 included papers to a new separate folder in Mendeley.
4. Download all level 2 included papers as PDF files.
5. Read the body content of all level 2 included PDF files manually.
6. Apply inclusion and exclusion level 3.

After this step, 45 out of the 490 remaining papers were kept for in-depth analysis to answer the research questions.

3.9. Snowballing

In a SLR there is always the threat that some relevant papers are missed from the selected bibliographic sources. For instance, 4 bibliographic sources were selected in this SLR, but there is no 100% guarantee that all primary studies exist in the 4 chosen bibliographic sources. To reduce this possibility, we applied many pilot searches and refined the search string to cover as many primary studies as possible. In addition, we also used backward snowballing [24] on the selected 45 papers in the Google scholar database to complement the search result of our SLR.

After applying the snowballing procedure defined by Wohlin et al. [24], we found two extra articles, which were not included in the original search result. The first paper was not in the "software engineering" category, which explains why it was missed in the four chosen bibliographic sources. The second paper was missed since it contained the continuous integration keywords in the body of the paper only, but not in the paper title, keywords, or abstract. By adding the complementary two papers, the final total number of selected papers was 47.

3.10. Data collection

We read all the selected papers and collected the below four primary types of data to analyse listed below:

[1] https://www.mendeley.com.
[2] https://docs.python.org/2/library/random.html.

Table 4
Inclusion criteria.

Inclusion criteria level 1
- Published from 2008 to 2018
- Peer-reviewed
- English language
- Including "workshop," "conference," "journals," "book chapters"
Inclusion criteria level 2
- A study is related to both "Continuous integration environment" and "Non-functional requirements" in its title, keywords, or abstract.
Inclusion criteria level 3
- A study investigates, evaluates, validates software "continuous integration environment" from a technical perspective to evaluate NFRs defined in Table 2.

Table 5
Exclusion criteria.

Exclusion Criteria level 1
- Books
- Non-peer-reviewed papers like keynotes, editorials, master thesis, presentations, PHD thesis, lecture notes, patents, reviews, discussions
- A paper containing "systematic literature review" in the title or abstract
Exclusion Criteria level 2
- A paper investigates, evaluates, or validates quality attributes for "continuous integration environment" itself instead of quality attributes of a software product, for example, assuming there is a study evaluating or validating a software tool improves the stability of Jenkins in a CI environment, it should be excluded, due to "Jenkins" stability being about the CI environment itself instead of a software product.
Exclusion Criteria level 3
- A paper evaluates, validates, or investigates quality attributes for "continuous integration environment" itself instead of quality attributes of a software product.
- A paper without evidence showing explicit connection between continuous integration environment and quality attributes of a product. For example, a study discusses CI environment improvements; a disconnected discussion about software quality or how they perceive the CI environment to add or improve the software quality.

Table 6
Example Excel table for collecting paper properties.

Index	Author	Title	Paper type	Publish year	Research type	Comment
Pn	e.g. YL	e.g. X	e.g. journal	e.g. 2016	e.g. case study	highlights

1. The properties of all selected papers e.g. author(s), article title, type of paper, year published, type of research. The details are listed in Appendix A.
2. The CI approaches for NFR testing (e.g. methods, techniques, frameworks, processes) and associated software tools (RQ1).
3. The studies presenting challenges, problems, findings, and solutions in NFR testing by using a CI environment (RQ1) or discussing practices, guidelines, and lessons learned for this purpose (RQ2).
4. The non-functional quality attributes tested by using a CI environment in case studies or industrial practices (RQ2).

We stored the collected data in an Excel file with two tables for further analysis, and the table headers with an example content shown in Tables 6 and 7.

3.11. Synthesis

Based on the collected data, we classified the selected papers, listed in Appendix A, into three research types: theory research, case studies, and industrial studies. Secondly, we applied the *thematic coding approach* [25,26] to extract codes:(1) CI environment codes, (2) NFR codes and (3) Challenges of testing NFRs by using CI environment. The extracted codes were combined in five levels to create the chain of evidence, which were synthesised to unveil the connections between the CI environment and NFR testing.

3.11.1. CI environment codes

As shown in Fig. 2, the top level of CI environment code was "continuous software engineering." The second level of codes was "continuous integration," "continuous delivery," "DevOps," "rapid system development," and "cloud environment". The third level of codes extracted as "automation" and "tool-chain". The fourth level of codes consisted of "tool-chain processes" and "tools" used in CI environments, for instance, "source code management" (Gerrit, Bitbucket, etc.); "version control system" (Git, SVN, StarTeam); "static code analysis" (SonarQube, FindBug, CheckStyle); "CI tool" (Jenkins, Hudson, Travis-CI, TeamCity, Bamboo, etc.); "cloud platform" (Docker, vSphere, Openstack); "Artifacts/dependencies management" (Artifactory, Maven, Ant); "flow automation" (Ansible, Vagrant, Chef); automation testing (Junit, Jmeter, Selenium, RobotFramework); "issue tracking" (JIRA, Github issue, MantisBT).

These codes show the chain of evidence we used to find "CI tool-chain processes" and "tools." In addition, we mapped the codes to the selected papers.

Knauss et al. [5] investigated CI environment from a tooling perspective beyond a team level in an automotive industry; Deissenboeck et al. [27] presented a tool-chain quality model to assess the quality of a software system; Soni Mitesh [28] also proposed a CI environment framework from tooling perspective to improve software quality.

We extracted CI environment codes from a tooling perspective, e.g. "tool-chain," "auto build," "E2E automation," "deploy pipeline," CI processes, and software tools used in a CI environment.

3.11.2. NFR codes

As shown in Fig. 2, we extracted nine non-functional quality attributes from the collected data: efficiency, reliability, productivity, latency, availability, performance, maintenance, scalability, and stability. The NFR attributes and mapped paper index are shown in Table 8.

3.11.3. Challenges of testing NFRs by using CI environments

The challenge codes are descriptive texts that collected during the data collection phase from the selected papers and stored in Excel tables, e.g. Tables 6 and 7. In the synthesis phase, we read and group the descriptive texts into categories manually based on our interpretations. As a result, eight categories of challenges have been synthesised in general from the collected data and presented in Fig. 3:

- A set of tools cause unstable CI environment: NFR testing requires more software tools integrated into the existing CI environments, which could cause CI environments unstable [5].

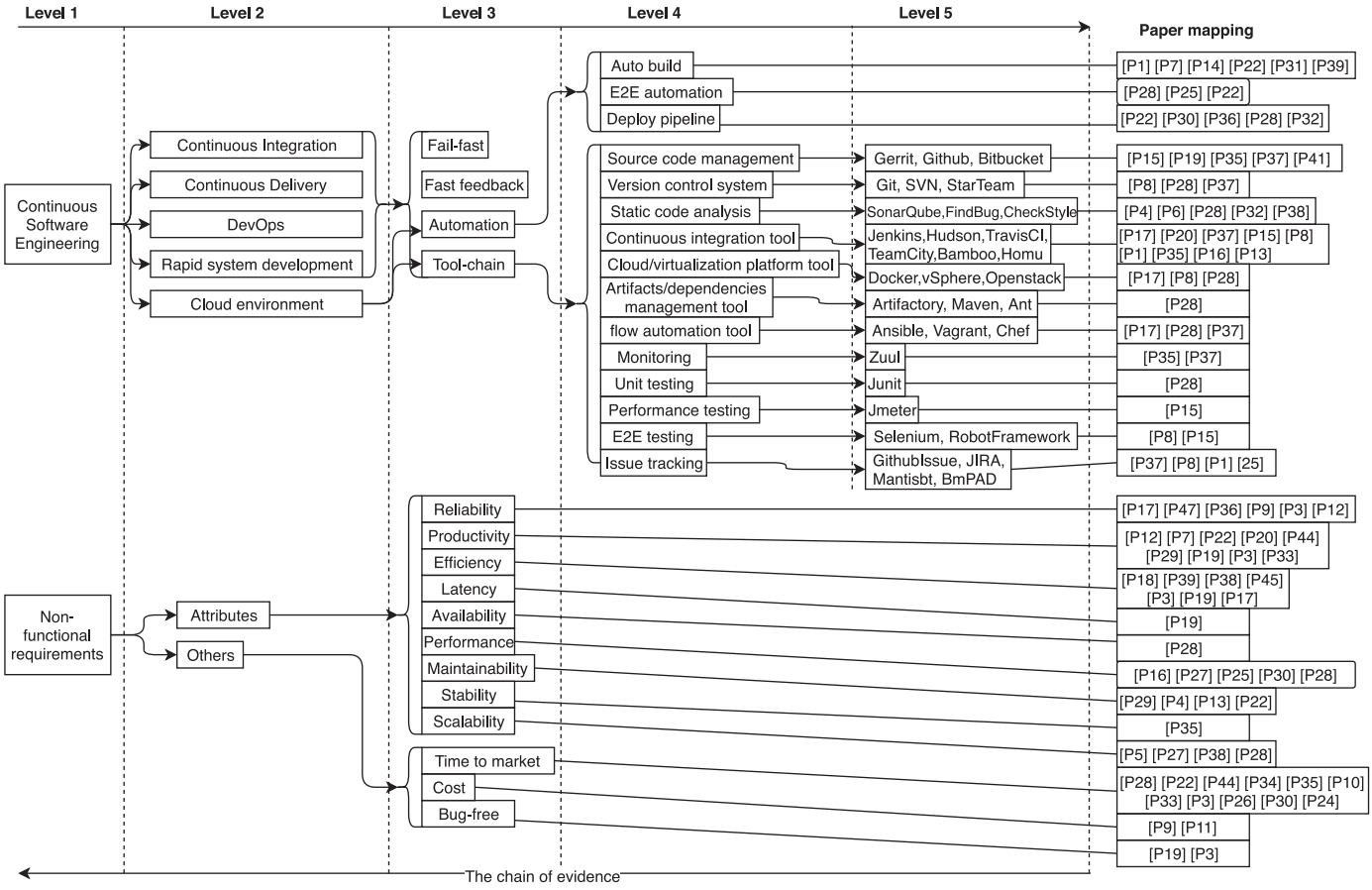

Fig. 2. Extracted codes mapping with selected papers.

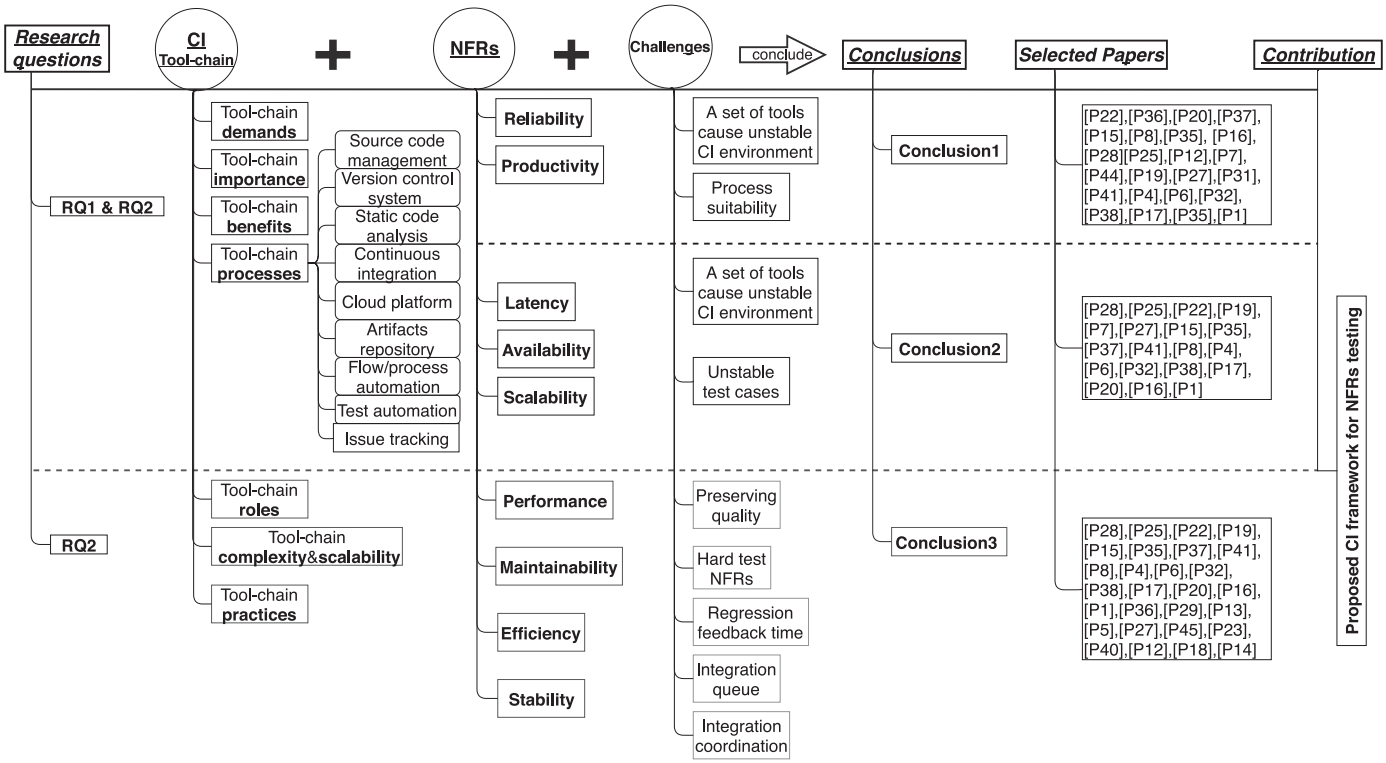

Fig. 3. Synthesis of collected data.

Table 7
Example Excel table for collecting contextual codes.

Keyword	Referral sentence(s)	Paper index	Paper page	Conclusion	RQ
e.g NFR	e.g. XX examined a large set of projects to expose the relationship between NFR and CI build statuses.	e.g. P(1...47)	e.g. page 2	e.g. ideas	RQx

Table 8
NFR attributes reported in selected papers.

NFR attribute	Index of selected paper
Reliability	P12, P47[29], P36, P3, P17, P9[30]
Productivity	P44, P19, P12, P29, P33[31], P3, P7[32], P22, P20
Efficiency	P3, P18[33], P38, P39[34], P17, P19, P45
Latency	P19
Availability	P28
Performance	P16[35], P28, P25, P27, P30[36]
Maintenance	P4, P29[37], P13, P22
Scalability	P5, P27[38], P28, P38
Stability	P35

- Process suitability is a challenge: A single CI process does not fit all NFR tests. It is a challenge to support many test processes in a CI environment.
- Unstable test cases: The CI environment changes or issues, network impacts, and hardware failures could cause test failures.
- Preserving quality: Using the CI environment to maintain a level of quality is a challenge.
- Some NFRs are hard to test: For example, the usability is difficult to automate and test by using a CI environment.
- Regression feedback time: Some NFR regression testing requires the full system is up running which takes long time to feedback developers.
- Integration queue: NFR tests normally have dependent components which leads to integration queue issue.
- Integration coordination: The integration dependencies of NFRs are not easy to handle by using the CI environment.

3.11.4. Code analysis

We followed the steps below to analyse the extracted codes.

1. Familiarise with extracted codes: we read and examined all extracted codes, e.g. CI tool-chain approaches and tools, NFR attributes, and challenges to form initial ideas for analysis.
2. Review initial codes: we reviewed and refined all codes extracted from the selected papers. In some cases, we had to read the papers again.
3. Connect initial codes: we tried to combine different initial codes of CI environments, NFRs and challenges to get potential themes.
4. Summarise contributions based on themes: we concluded different contributions based on CI environment codes, NFR codes, and challenges.
5. Map research questions: we connected the contributions with each research question and the selected papers.

As shown in Fig. 3, we categorised and joined the extracted codes "CI tool-chain," "NFRs," "Challenges," and "Selected Papers" to group keywords into more general keywords that later helped support the conclusions. There are three synthesised conclusions in Fig. 3, and they are mapped to the findings described in Section 7.1.

Based on our findings for "CI environments," "NFRs," and "Challenges," we proposed a general CI framework for NFR testing (see details in Section 6).

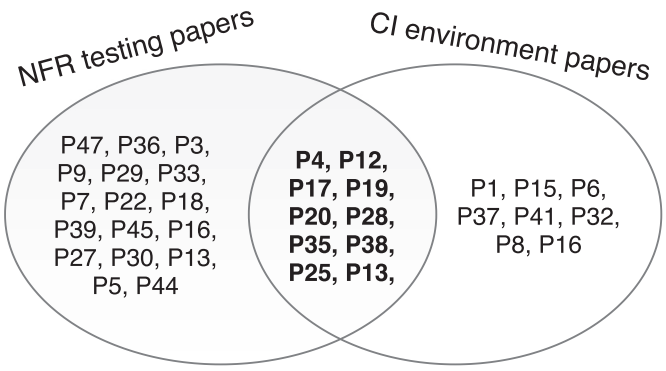

Fig. 4. Papers for NFR testing by using CI environments.

4. Results

This section will report the results from analysing and synthesising the extracted data to answer the research questions. The results are based on the synthesised data directly with our interpretations.

4.1. RQ1: What is state-of-the-art (SOTA) of utilising CI environments to test NFRs?

We found 27 papers, listed in Fig. 2, that reported approaches and associated tools for NFR testing, and 18 articles that investigated the CI approaches from a tooling perspective. The overlapped papers between NFR testing and CI approaches are presented in Fig. 4.

We provided a reference list in Appendix A as to where to find the title of the selected papers denoted as PX, e.g. P4, P12, etc. In Fig. 4, we identified ten papers that related to NFR testing by using CI environments, and the index name of the ten papers are P4, P12, P13, P17, P19, P20, P25, P28, P35, and P38.

In summary, as shown in Fig. 5, 10/27 (37%) papers used CI approaches for NFR testing. In all those CI-NFR testing papers, the NFR coverage percentage is that latency, availability, and stability take six percent each; maintainability and reliability is 12 percent; scalability, performance, and productivity cover 13 percent; efficiency is the highest at 19%.

Our SLR focused on utilising CI environments to test NFRs. We categorised the papers that investigated NFRs testing by using CI approaches into two types: state-of-the-art research (Table 9) and state-of-the-practice report (Table 11). The extracted data included:

- which year the paper was published.
- which type of article it is.
- which research type it is.
- which NFRs were tested by the research.
- which CI approaches were used to test NFRs.
- which CI tools were used for NFR testing.

4.1.1. RQ1.1: What CI approaches have been proposed for NFRs testing in the SOTA?

First of all, our motivations of utilising CI environment tool-chain for NFR testing are listed as follows:

Table 9
State-of-the-art of the CI-NFR testing.

Index	Publish year	Paper type	Research type	NFR focused	CI approach	CI tool(s)
P4	2012	Conference	Case study	Maintainability	3C approach	CruiseControl Findbug Cobertura
P12	2015	Conference	Case study	Reliability	EQUITAS toolchain	TeamCity SVN JIRA
P17	2015	Conference	Theory Case study	Efficiency Reliability	HARNESS	BuildBot Git Docker Vagrant
P19	2015	Journal	Theory Case study	Efficiency Productivity Latency	GitHub CI environment	Travis-CI Github Issue
P20	2015	Conference	Theory	Productivity	CodeAware	Jenkins Git Gerrit
P28	2016	Journal	Theory	Performance Availability Scalability	E2E automation	Jenkins Git Maven, Ant Artifactory WMware Chef
P35	2016	Conference	Theory	Stability	GitWaterFlow	Homu Zuul Gerrit Openstack
P38	2017	Conference	Theory	Scalability Efficiency	Cloud-based Code-analysis	Jenkins Git SonarQube

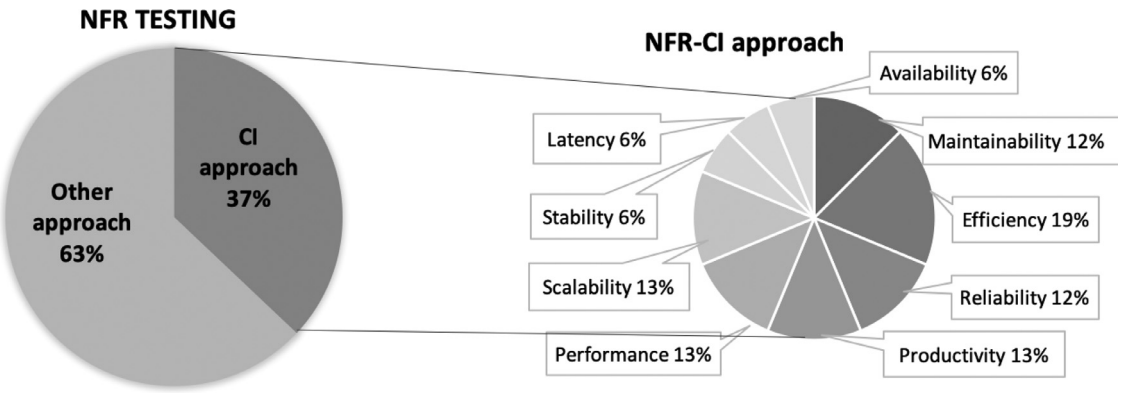

Fig. 5. The coverage of CI-NFR testing.

- **CI Benefits**: Existing CI tools, e.g., Jenkins, lead to significantly reduced integration problems and allows a team to develop cohesive software more rapidly P22[4].
- **CI demands**: High demand exists in software development for automated tool-chain for packaging, testing, and deploying software to ensure the latest working version of the software can be deployed at any given moment P36[39].
- **CI importance**: Automated software engineering tools play crucial roles in managing software development processes efficiently and effectively. They help to manage project schedules, tracking defects or events, leaving fast feedback and various reports, controlling software configuration, and automating the build and deployment process P36[39].

Secondly, we illustrated the existing state-of-the-art CI approaches (The 3C approach [40], the EQUITAS tool-chain [41], the Quality and productivity [42], CodeAware [10], the E2E automation [28], GitWaterFlow [43], Cloud-based code-analysis [44], and the HARNESS [45]) used for testing NFRs shown in Table 9.

The 3C approach [40]: This includes "continuous integration," "continuous measurement," and "continuous improvement." This CI approach introduces four components: "version control system" (SVN tool), "continuous integration" (CruiseControl tool), "static code analysis" (Findbugs, PMD, and Checkstyle), and "dynamic code analysis" (Junit tool). The 3C approach works in the steps as follows:

1. Developers push NFR commits.
2. The commits trigger static code analysis with tools: Findbugs, PMD, and Checkstyle.
3. The commits trigger dynamic code analysis with tool Junit.
4. Report the testing results.
5. Sync test results including build and test result, unit test result, lines of code result and test coverage result to quality manager.
6. Quality manager drives codes refactoring as continuous improvement.

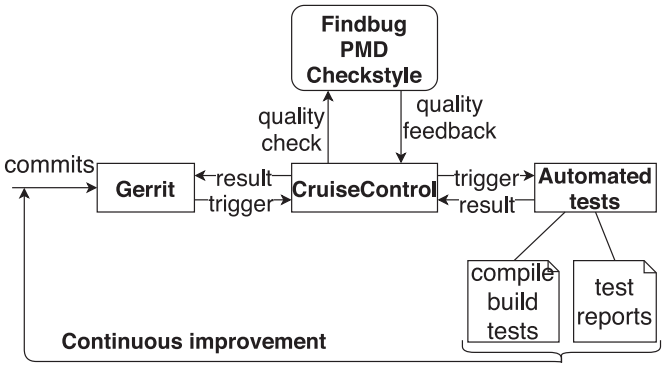

Fig. 6. The 3C approach [40].

7. Go to the first step working as an Eco-system.

The 3C approach and components are shown in Fig. 6. Janus et al. [40] presented that this approach was used as an eco-system for static code analysis and dynamic code analysis against each commit including both functional requirements and NFRs. They mentioned that the code quality was continuously improved and the maintenance work was reduced significantly by applying 3C approach [40].

The EQUITAS tool-chain [41]: EQUITAS represents enhanced quality using intensive test analysis on simulators. This approach is used to limit the impact of software verification and validation on the cost and time-to-market of embedded systems while improving reliability and safety [41]. The workflow of EQUITAS is shown in Fig. 7.

In the EQUITAS [41] workflow, project activities include:

1. Continuous tool-chain development to automate the verification and validation processes.
2. Continuously detecting redundant tests.

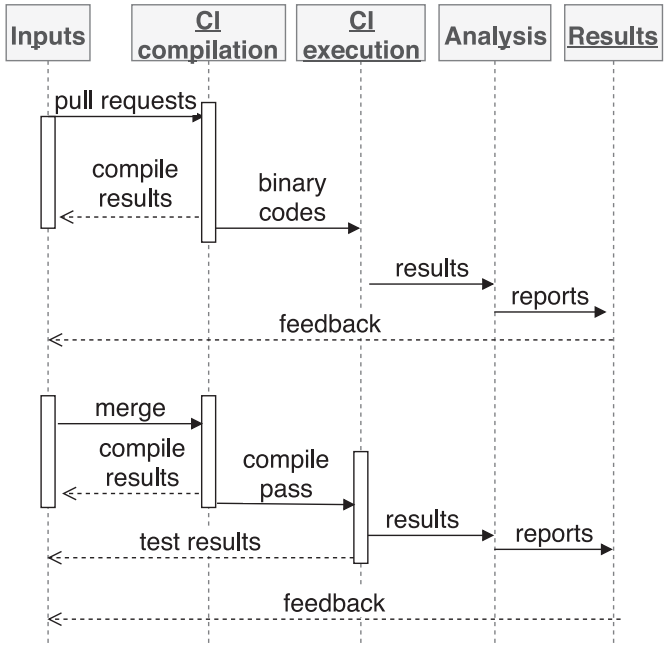

Fig. 7. The EQUITAS workflow[41].

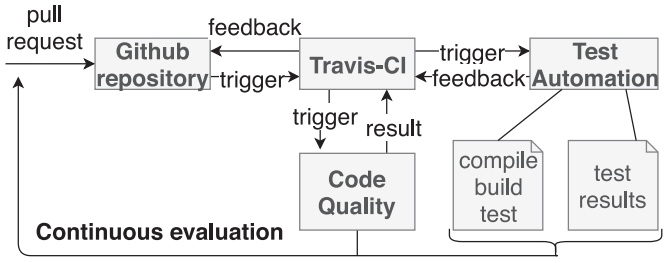

Fig. 8. The Quality and productivity CI workflow[42].

3. Continuously improving hardware failures.
4. Using virtual platforms to validate use cases.

Quality and productivity approach [42]: Vasilescu et al. [42] demonstrated a CI approach based on a number of Github projects focussed on process automation, and they found continuous integration improved the productivity of development teams.

Vasilescu et al. [42] investigated the pull request process and test automation by introducing continuous integration solution. As shown in Fig. 8, the CI workflow is that firstly developers create a pull request in the source code repository on Github, then the CI tool Travis-CI automatically triggers tests and sends results back to developers. Based on the CI feedback of code quality and testing results, developers create more pull requests.

The productivity of developers was increased without reducing code quality by applying this approach presented by Vasilescu et al. [42].

CodeAware [10]: CodeAware is a flexible and scalable ecosystem consisting of monitors and actuators which aims at improving code quality and team productivity.

The workflow of CodeAware [10] approach is shown in Fig. 9. the CodeAware [10] approach improves pull requests' reviews through source code repository and code quality by using CI server. It also fails tests quickly and notifies developers via feedback as fast as possible, in order to prevent faults merged into source code repository and help developers to fix the failed tests quickly [10].

Abreu et al. [10] presented that both the source code quality and maintainability of projects were improved by using the CodeAware [10] approach. This approach focussed on source code management component in a CI environment, and it indirectly improved NFR maintenance [10].

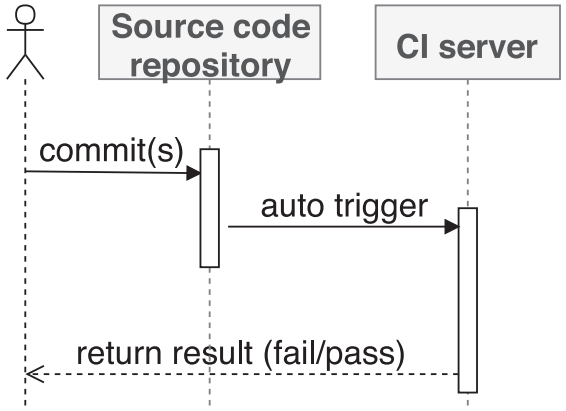

Fig. 9. The CodeAware workflow[10].

The E2E automation [28]: The E2E automation solution includes coding, building, integrating, testing, troubleshooting, infrastructure provisioning, configuration management, setting up the run-time environment, and deploying applications in different environments [28].

The E2E automation [28] approach is shown in Fig. 10, and the E2E automation activities include:

1. Integrate source code repository to CI server.
2. Run build and deployment job automatically whenever code changes are detected.
3. CI server configurations.
4. Create build pipeline for quality assurance.
5. Integrate CI server with a static code analysis tool.
6. Integrate CI server with an artifact repository.
7. Configuration of CI server and cloud environment.
8. Infrastructure provisioning with configuration management tool.
9. Install and configure the run-time environment.
10. Deploy applications to run-time servers.

Benefits for testing NFRs by using the E2E automation [28]:

- Faster delivery of builds, features, and bug fixing.
- Accelerated customer feedback cycles and collaboration results into high quality for feature/function delivery.
- Streamlining of the deployment process and improving the quality of scripts and integration.
- High availability of resources.

GitWaterFlow [43]: It is a Git-based version control and branching model to provide time-critical delivery of bug fixes.

This approach aims to ensure high quality and stability to achieve short and predictable release cycles with minimised developmental disturbance. The workflow of GitWaterFlow [43] showns in Fig. 11. Developers firstly make pull requests to Bitbucket server, and tests are triggered automatically by using the Jenkins tool.

Rayana et al. [43] presented that both the quality and efficiency of delivery was enhanced by using the tool "bitbucket", "Bert-E", "Jenkins", and "JIRA".

Cloud-based code-analysis [44]: It is a scalable cloud-based platform providing accessibility for quality managers and efficient use of the analysis data in a continuous integration system for trend analysis in combination with the software quality model that can indicate the overall quality level of the software [44]. The framework of the cloud-based code-analysis approach is shown in Fig. 12.

The key benefit of this approach presented by Bougouffa et al. [44] is the ability to integrate the code-analysis feature in a CI environment in order to run a regular source code analysis. The system executes a set

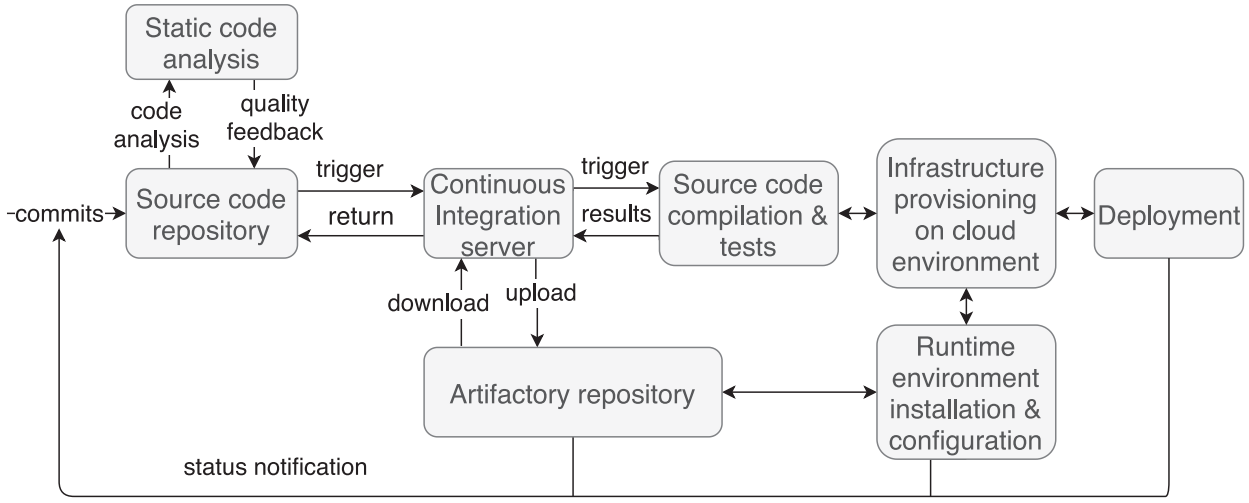

Fig. 10. The E2E automation approach[28].

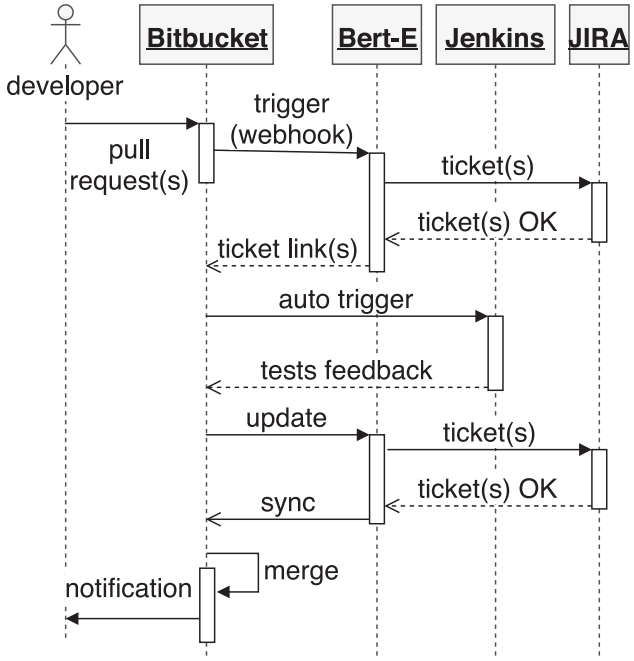

Fig. 11. The GitWaterFlow workflow[43].

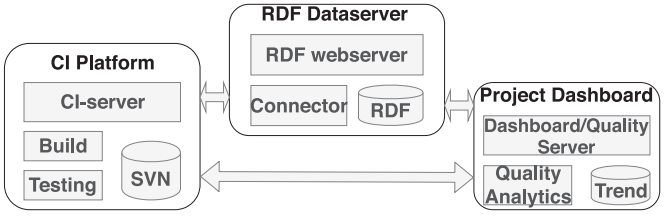

Fig. 12. The cloud-based code-analysis framework [44].

of predefined test cases to check regression issues by the latest code changes in order to improve the efficiency and quality [44].

HARNESS [45]: It is a DevOps approach based on version control system, automated software deployment and continuous integration. The workflow of the approach is shown in Fig. 13.

Stillwell et al [45] introduced the workflow as developers firstly made a commit and pushed to the source code repository, then the

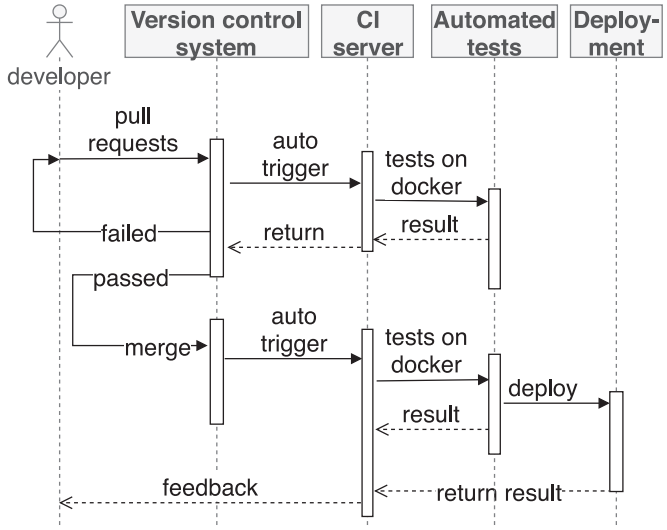

Fig. 13. The HARNESS workflow[45].

CI tool automatically triggered unit and integration tests executed in a docker container which was installed and deployed by using Ansible and Vagrant tools in an automated way. They also mentioned that the automation of environment installation and configuration improved the speed of software deployment and also boosted the efficiency of using the hardware resources, as a result, developers received fast feedback and the total time of fixing bugs was reduced as well [45].

In summary, we identified 8 CI approaches to answer RQ1.1. However, we found a bias of interpretation of these CI approaches for NFR testing by using CI environments. The CI approach "HARNESS" [45], focussed on evaluating the efficiency of the CI system itself rather than the efficiency of a software product and is thereby out of scope for this SLR. However, this paper also reported that the speed of deploying software improved by using the same CI approach. Therefore, we included this CI approach for RQ1.1.

4.1.2. RQ1.2: What tools have been proposed or used in a CI environment reported in the SOTA?

A list of tools used in CI environments for NFR testing was extracted from the selected papers. The tools' names and referred papers' index is shown in Table 10.

Table 10
A list of tools used in CI environments.

Tool name	Index of referred paper
Gerrit	P15, P35
Bitbucket	P37
Github	P37, P41[46], P19
Git	P37
SVN	P12
StarTeam	P28
SonarQube	P6, P38, P28
Findbug	P4, P32[47], P28
Cobertura	P4
Checkstyle	P6[48]
Jenkins	P17, P20, P37
Hudson	P15
TravisCI	P37
TeamCity	P12
CruiseControl	P1
Homu	P35
Buildbot	P17
Bamboo	P37
Docker	P17
vSphere	P12
Azure	P28
Artifactory	P28
Maven	P28
Ant	P28
Ansible	P37, P17
Vagrant	P17
Chef	P28
Zuul	P35
Junit	P28
Jmeter	P15
Robot Framework	P12
Selenium	P15
Github Issue	P37
JIRA	P37, P12
MantisBT	P1[49]
BmPAD	P25

The tools included both open-source tools (like Jenkins, SonarQube) and commercial tools (like Bamboo, vSphere). As shown in Table 10, open source tools were more commonly used by researchers comparing to commercial tools, for instance, Gerrit, Github, SonarQube, Findbugs, Jenkins, Ansible and JURA are all open source tools, and have been reported in multiple publications.

4.2. RQ2: What is state-of-the-practice (SOTP) in the research of utilising CI environments to test NFRs?

We identified only 2 industrial-practice studies that presented NFR testing by utilising CI environments. The focused NFRs (maintainability and performance), CI approaches (Quality Model [50] and the HANA's CI process [11]) and CI tools (Jenkins[3], SVN[4], BmPAD [11]) were illustrated in Table 11.

To answer this research question, we defined three steps to illustrate each industrial practice as follows:

1. *Introduction*. Describe the basic information for each industrial case, e.g. the purpose of the industrial case, which year, and in which company.
2. *Approach*. Introduce the CI approach and tools utilised in each industrial practice.
3. *Usage*. Explain the way of evaluating NFRs by using the CI approach.
4. *Result*. Show the result after applying the approach.

The two identified industrial cases are: **Case 1** comes from P13[50] and **Case 2** is from P25[11].

[3] https://jenkins.io.
[4] https://subversion.apache.org.

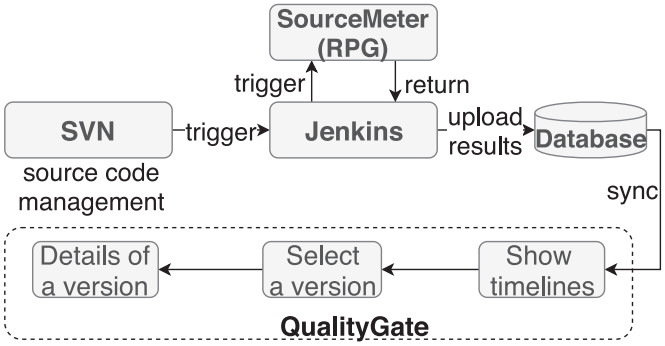

Fig. 14. The quality model workflow[50].

Case 1: Introduction: Ladnyi et al. [50] introduced a quality model for RPG programming language to evaluate the maintainability of RPG software systems and one successful validation of the quality model conducted in the R&R software development company in 2015.

Case 1: Approach: The quality model contains a CI tool (Jenkins and SVN) and a QualityGate [50] (continuous quality monitoring tool). The quality model workflow is shown in Fig. 14.

Case 1: Usage: A few steps used to improve maintainability of the RPG system [50].

1. Define software metrics, rule violations
2. Code refactoring for an RPG system component
3. Push source codes to SVN repository
4. Auto trigger static code analysis with tool SourceMeter[5] running on CI Jenkins.
5. Upload quality benchmark data to a database.
6. Use QualityGate [50] to monitor and evaluate maintainability of the system based on the time-line

These steps helped developers in R&R company to avoid duplicated classes or functions by using the static code analysis tool (SourceMeter) presented by Ladnyi et al. [50]. Jenkins and SourceMeter tools automatically validate code quality continuously which makes source code more maintainable.

Case 1: Result: The application of this CI approach was a big success, and the number of critical and major rule violations was reduced by 50% [50]. As a result, the maintainability of the project was improved and crossed the baseline value defined in the company [50].

Case 2: Introduction: Rehmann et al. [11] presented a method to monitor the database performance early at different stages of CI processes, and the method conducted in SAP HANA database platform in 2016 [11].

Case 2: Approach: The HANA's CI development workflow is visualised in Fig. 15.

Firstly, developers push the code changes to feature branches or hotfix branches, and then CI server triggers "barrier tests" and "regression tests." If all tests are passed, developers are able to merge the code challenges into release branch, otherwise, developers have to improve their source codes.

Case 2: Usage: In the study of Rehmann et al., this CI approach was used to keep track of performance regressions for the development teams [11] by following below steps:

1. Introduce performance metrics.
2. Integrate performance tests into the pre-commit tests of the CI pipeline.
3. Collect testing data and benchmark.
4. Detect and report performance deviations by using CI pipeline.

[5] https://www.sourcemeter.com.

Table 11
State-of-practice of using CI approach to test NFRs.

Index	Publish year	Paper type	Research type	NFR focused	CI approach	CI tool(s)
P13	2015	Conference	Industrial practice	Maintainability	Quality Model	Jenkins SVN
P25	2016	Journal	Industrial practice	Performance	HANA's CI process	BmPAD

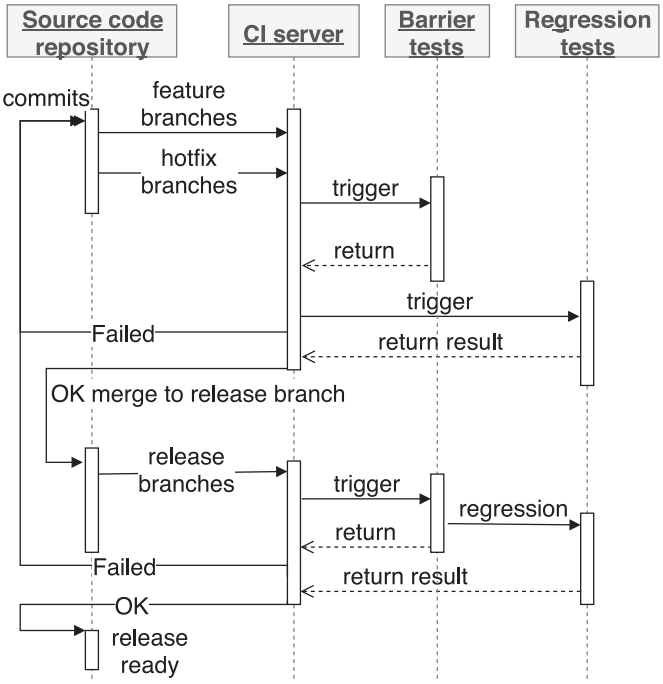

Fig. 15. The HANA's CI workflow[11].

Case 2: Result: The result of this industrial practice is listed as follows:

1. The performance regression tests were established in the very early development process of HANA DB [11].
2. The overall number of performance bugs decreased along with the lifetime of bugs reduced as well [11].
3. Developers of HANA DB teams became more aware of the performance implications of code changes [11].

In summary, these industrial studies clearly show that utilising a CI environment has a positive impact on the system's NFRs in industrial studies, as shown by the above two cases which improved maintainability and performance. Furthermore, the studies present practical steps to adopt CI approaches and associated tools for NFR testing in two companies, which in practice, could be applicable to other companies.

4.2.1. RQ2.1: what NFRs have been reported as testable by utilising a CI environment?

The RQ1 and RQ2 answered the NFR testing by utilising CI environments from both state-of-the-art and state-of-practice aspects in this section. After combining the results of both RQ1 and RQ2, nine NFRs (reliability, productivity, efficiency, latency, availability, performance, maintainability, scalability, and stability) have been reported and evaluated by using CI approaches and tools. The NFRs and connected papers are presented in Table 12.

As shown in Table 12, in total, 10 studies reported nine NFRs in three types of research: theory, case study, and industrial study. The theory studies (2/10) are P28 and P35; the case study studies (7/10) that were based on theory are P12, P17, P19, P20, P38, P4, and P28; the industrial studies (2/10) are P25 and P13. The mapping between NFRs

Table 12
Testable NFRs reported in both SOTA and SOTP studies.

NFR name	Index of selected paper	Type of research paper
Reliability	P12, P17	Theory + case study
Productivity	P19, P20	Theory + case study
Efficiency	P17, P19, P38	Theory + case study
Latency	P19	Theory + case study
Availability	P28	Theory
Performance	P28	Theory
	P25	Industrial practice
Maintainability	P4	Theory + case study
	P13	Industrial practice
Scalability	P28, P38	Theory + case study
Stability	P35	Theory

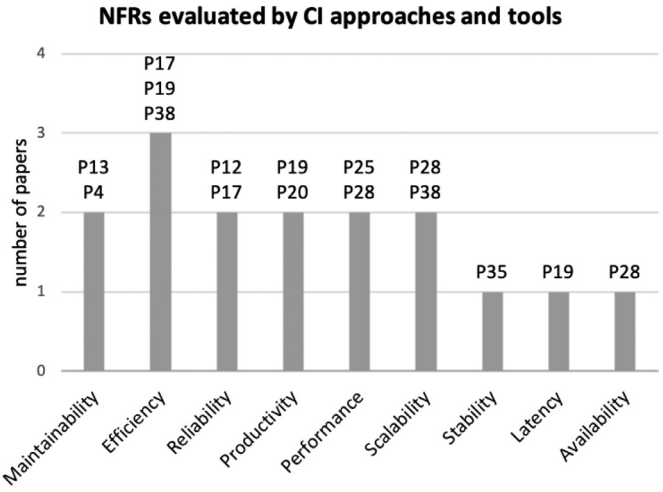

Fig. 16. The reported NFRs and mapped papers.

and connected papers is visualised in Fig. 16. Three papers (P17, P19 and P38) reported efficiency; maintainability in P4 and P13; reliability in P12 and P17; productivity in P19 and P20; performance in P25 and P28; scalability in P28 and P38; stability in P35; availability in P28; latency in P19.

As we can see here, *only 2 papers performed in actual industrial practice which are underrepresented, implying a need for more industrial research on the investigation of NFRs testing by utilising CI environments*.

4.3. Challenges of testing NFRs by using CI environment

A list of challenges were found for testing NFRs by using CI environments in this SLR, which are presented as follows:

– **A set of tools cause an unstable CI environment**. For a CI environment, it is required to combine a number of tools to be well integrated and configured. One tool does not fit all needs, but more tools increase the complexity for use and maintenance. However, NFR testing utilises software tools running on a CI environment in many cases. Many existing NFR testing tools are hard to integrate in a CI environment due to an appropriate visualised UI interface of the tool that is missing, and this is a major contributor for unstable CI environments. To overcome the UI interface problem, a number

of homemade tools in different companies are implemented for NFR testing, which cause the CI environment to be unstable P31[5].

- **Regression feedback time**. The common issue is that the regression testing feedback takes too much time, including NFRs regression tests. Some regression testing takes from six hours to two days after integrating the source code, and this long-time feedback loop could cause the result to be already out-of-date already P14[51].
- **Integration queue**. In some projects, hundreds of developers are working on different components at the same time from different locations - meaning source codes, including both functional and non-functional pull requests, are constantly added to the integration queue constantly. As a result, it is hard to keep track of all changes, and this increases the chance of blocking the integration queue and delivery branch being blocked for days. In this SLR we focused on NFRs aspects, but Laukkanen et al. [17] presented this problem from the functionality perspective when continuous delivery was adopted.
- **Unstable test cases**. NFR test cases are sometimes not stable, and it is difficult to evaluate the test result, due to the fact that the failure might lead to network failure, environment issues, tool downgrade/upgrades, hardware resources, etc. [52].
- **Preserving quality**. Maintaining the right level of quality while adopting CI is a challenge. By utilising CI environment, developers can get fast feedback for NFR testing. Comparing the NFR testing without using a CI environment, the integration frequency is much higher, and higher integration frequencies increase more pressure to preserve quality [52].
- **Process suitability**. The same frequent integration does not fit all different components or projects. The integration frequency depends on the feature demands, for example, some components are depending on many other sub-components or projects, therefore, it should be more frequent integration [52]. Especially for NFR testing, it exists and affects almost all components in a project, like a system performance and security.
- **Integration coordination**. It is not easy to handle integration dependencies by using existing CI processes [52], for example:
  - Component interfaces need to be clearly defined, but NFR interfaces are less considered or planned in many projects.
  - More failures during integration, and debugging is time-consuming.

  Imagine a situation when an NFR feature is committed, but the integration test keeps failing over a long period due to a dependent component is not being ready yet - sometimes a long waiting period for a feature or a component causes people to forget their failed commits, especially when the priority of a task is not very high. This leads to “dead code” [52].
- **Hard to test non-functional requirements**. Functional requirements can usually be broken down to basic functions and fulfilled by a dedicated module or component in a product. However, many NFRs cannot be fulfilled by a sub-module or sub-component. By planning NFRs early, expensive design changes can be avoided in the later software lifecycle P23[53].

5. Discussion

In this section, we interpret and reflect on the results in the previous section.

5.1. NFRs evaluated by utilising CI approaches and tools

In Section 4, 21% (10 out of 47 papers) investigated NFRs testing by utilising CI approaches and tools. The number of papers is a very low ratio, and it tells us that there is a lack of research about NFR testing by using CI approaches and tools. One reason could be that there is a lack of CI tools for NFR testing and some CI tools are hard to integrate into CI environments. Another possible reason is that the NFR testing increases the CI tool-chain complexity and maintenance work as well.

Moreover, we compared our extracted NFRs with the international standard ISO/IEC 25010:2011 [54], and the quality attribute “portability,” which is defined as the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one environment to another [54], which does not exist in our results. The reason could be that the portability is closely connected to functional requirements (e.g. installation and upgrade features) in many projects.

Besides, we only found 2 industrial studies in Table 11, which related to NFRs testing by using CI approaches and tools - the others were theoretical studies. The low number of industrial studies indicates that CI approaches and tools are not commonly utilised for NFR testing in industrial cases. The reason could be that, in production, it requires a stable and reliable CI environment for NFR testing, but it takes time to verify and improve theoretical or empirical research continuously to be stable in the end as well as apply it to real industrial projects.

In addition, some NFRs have been investigated and reported more frequently than the others. For instance, in our SLR efficiency is reported by researchers to have a higher ratio compared to the other NFRs as shown in Fig. 16. A few identified reasons are:

1. Efficiency is associated with several clearly defined metrics, e.g. response time, CPU/Memory usage, data accuracy, etc.
2. A number of CI tools exist to collect efficiency data.
3. Developers have awareness of the efficiency of a project.

5.1.1. CI components

As we can see in Table 9, the eight CI approaches and CI tools were presented to answer RQ1, and in Table 11, the two CI approaches and associated tools were used to answer RQ2.1. These 10 CI approaches and corresponding tools have been used for NFR testing in previous studies. We extracted eight CI components, which are described as follows:

- **Source code management**. *Purpose*: This is the source code repository where developers push their changes of the source code. The purpose is to handle source code pull requests and reviews. *Example tools*: (Gerrit,[6] GitHub,[7] Bitbucket[8]) reported in P19[42] P35[43] P37[55].
- **Version control system**. *Purpose*: The purpose of a version control system is to record changes of a file or a set of files over time so that specific versions can be recalled at any time. *Example tools*: Git,[9] SVN, StarTeam[10] used in P8[56] P28[28] P37[55].
- **Static code analysis**. *Purpose*: The purpose of static code analysis is to analyse source codes against a set or multiple sets of coding rules when a source code pull request is committed to the source code repository. *Example tools*: SonarQube,[11] FindBug,[12] CheckStyle[13] presented in P4[40] P28[28] P38[44].
- **Continuous integration**. *Purpose*: The purpose of continuous integration is to integrate code into a shared repository very frequently and then automatically build and test source codes in order to help developers detect problems. *Example tools*: Jenkins, Travis-CI[14], TeamCity[15] reported in P17[45] P37[55] P35[43].
- **Cloud/virtualisation platform**. *Purpose*: The purpose of Cloud/virtualisation platform is to build, ship, and run any service or application that allows you to virtualise the platform such that it can be run in a cloud without an in-house hardware

[6] https://www.gerritcodereview.com.
[7] https://github.com.
[8] https://bitbucket.org.
[9] https://git-scm.com.
[10] https://www.microfocus.com/products/change-management/starteam/.
[11] https://www.sonarqube.org.
[12] http://findbugs.sourceforge.net.
[13] http://checkstyle.sourceforge.net.
[14] https://travis-ci.org.
[15] https://www.jetbrains.com/teamcity/.

platform. *Example tools*: Docker,[16] Vsphere,[17] Azure,[18] OpenStack[19] presented in P17[45] P8[56] P28[28].

- **Artifacts and dependencies management**. *Purpose*: The purpose of artifact and dependency management is to store and manage software artifacts or dependencies. *Example tools*: Artifactory-Jfrog[20] used in P28[28].
- **Test automation**. *Purpose*: The purpose of test automation is to automatically run unit testing, integration testing, performance testing, E2E testing and user acceptance testing in CI environments. *Example tools*: Junit, Jmeter,[21] selenium,[22] robot-framework[23] reported in P8[56] P28[28] P15[57].
- **Issue tracking**. *Purpose*: The purpose of issue tracking is to record and follow the progress of an issue until it is resolved. An issue can be anything from a bug report to a customer question to development inquiry. *Example tools*: Github issue P19[42], JIRA P12[41].

The extracted CI components have their own purposes in the CI environment, and each of them represents the feature of one or many software tools. Different projects or organisations build and configure CI environments based on their needs. For instance, small start-up companies might only need "source code management," "version control system," and "continuous integration" server. Some large companies might require all these extracted components to ensure software quality. A unified CI components design could be beneficial for building a robust CI environment with appropriate software tools.

5.1.2. CI tools and components

To figure out where the tools can be used in a CI environment, we went through all tools presented in Table 10, and we mapped the tools to the CI components mentioned in the previous sub-section.

As shown in Table 13, each CI component is associated with one or several software tools. The CI environment consists of a number of components, and every component connects to tools which shows that tool-chain is very important to set up a CI environment.

Some components are mapped with many alternative tools, for instance, "source code management" is associated with "Gerrit," "Bitbucket," "Github." Each tool has its own strengths and limitations. For instance, "Gerrit" is an open-source tool and requires some configuration and maintenance work to set it up; "Bitbucket" needs less maintenance compared to Gerrit; "Github" is normally used by open-source projects. Suitable tools organised in a CI environment will cut project costs and reduce the configuration and maintainability of the CI environment.

5.2. Validity threats

In our SLR, four researchers actively participated in this review actively, and three of them have well experience in conducting SLRs validated the research processes. Nonetheless, there are potential threats to the validity of this SLR.

Internal validity: The analysis of qualitative data requires interpretations, which is subjective to potential bias. However, we have taken actions to triangulate all results with several sources. The result of the bias may be that some challenges, CI processes, or NFRs might be missed in data extraction. Another possible threat is that the categorised tool-chain processes from the tooling perspective might be biased because in all chosen papers we did not find rigorous information about how to group existing tools into proper CI processes.

[16] https://www.docker.com.
[17] https://www.vmware.com/se/products/vsphere.html.
[18] https://azure.microsoft.com/en-us/.
[19] https://www.openstack.org.
[20] https://jfrog.com/artifactory/.
[21] https://jmeter.apache.org.
[22] https://www.seleniumhq.org.
[23] http://robotframework.org.

Table 13
CI tools and components.

Tool name	CI component mapping
Gerrit	Source code management
Bitbucket	
Github	
Git	Version control system
SVN	
StarTeam	
SonarQube	Static code analysis
Findbug	
Cobertura	
Checkstyle	
Jenkins	Continuous integration tool
Hudson	
TravisCI	
TeamCity	
CruiseControl	
Homu	
Buildbot	
Bamboo	
Docker	Cloud/Virtualisation platform
vSphere	
Azure	
Artifactory	Artifacts/Dependencies management
Maven	
Ant	
Ansible	Workflow automation
Vagrant	
Chef	
Zuul	Monitoring
Junit	Unit testing
Jmeter	Performance testing
Robot framework	E2E testing
Selenium	
Github issue	Issue tracking
JIRA	
MantisBT	
BmPAD	

External validity: The selected literature studies were from the four selected bibliographic sources. There may be some missing articles in other bibliographic sources that present different cases related to our research. We have applied a snowballing analysis based on the chosen papers, which searches over other bibliographic sources. Additionally, we used inter-rater reliability analysis to validate the selected papers in order to make sure we included all relevant primary studies.

Construct validity: The SLR processes in this paper were well planned and validated by experienced researchers who conducted it step by step. The primary data was collected with clearly defined steps, and we utilised the thematic coding approach to extract evidence-chain of codes in order to answer the research questions. The conclusions we have made should be replicated by following the exact steps we have applied.

6. Proposed CI framework

We proposed a baseline CI framework based on the results we found through this SLR which aims to facilitate effective and efficient NFR testing. Moreover, we mapped all extracted CI tools and testable NFRs to each component of the proposed CI framework.

The proposed CI framework, including optional tools and what components of the CI chain that focus on what NFRs, is illustrated in Fig. 17.

6.1. Motivations

An unified E2E CI framework could be a solution to make test results less ambiguous, manage version-less documentation changes, and help make software continuously releasable at higher quality [17]. This makes a CI framework an important part for agile ways-of-working and

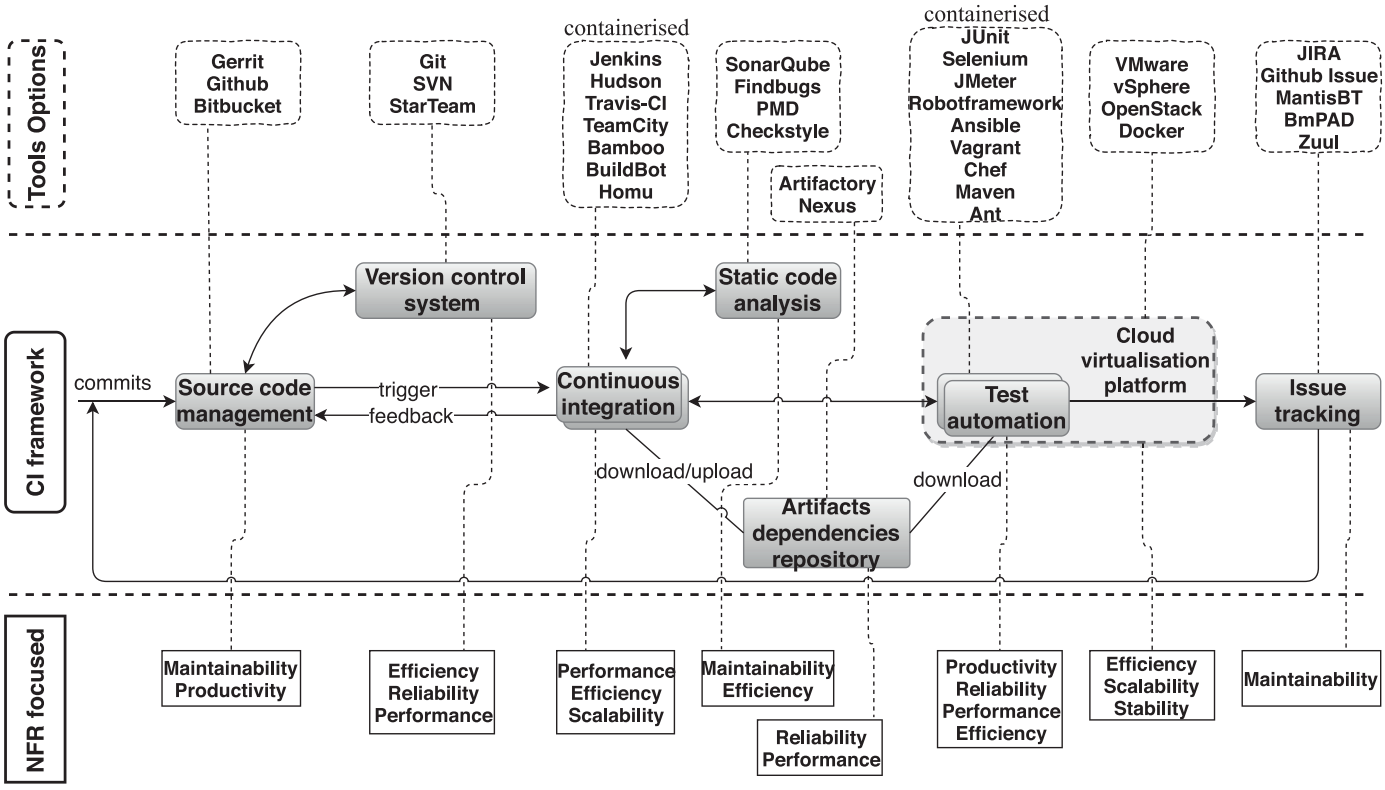

Fig. 17. Proposed CI framework for NFRs testing.

knowledge of how to set up such environments requested information from industry.

However, none of the ten CI approaches we found in this study present an E2E CI environment which could be used as a baseline for NFR testing. For example, the HANA CI approach covers only 3 components, which are ”source code management”, ”CI server” and ”automation test”, compared to the proposed CI framework which contains eight components supporting the E2E pipeline flow (as defined in Section 3.2).

Moreover, this proposed CI framework is a baseline design which could be used for different NFR tests customised with optional tools based on the needs. As such, our baseline model provides support for how to design such an environment on a finer level of detail compared to previously published works.

In addition, the proposed CI framework is also helpful to address some challenges we found in this SLR, for example, to improve the CI environment stability, to preserve quality, the CI process suitability, and NFR testing. In particular, NFR testing through utilisation of the CI environment is underrepresented in previous work, providing a contribution by itself.

6.2. Steps to get the framework

We used the following extracted data as input data for the proposed CI framework.

- Ten CI approaches extracted in Section 4.
- The CI tools shown in Table 10.
- The CI tools and components mapping in Table 13.
- NFRs presented in Table 9 and Table 11.

The general steps we applied to create the proposed CI framework are listed below.

1. Draw all CI components shown in the 10 CI approaches from Figs. 6 to 15.
2. Add connections between each CI components extracted from the 10 existing CI approaches.
3. Refine the workflow of CI components.
4. Get the proposed CI framework.
5. Use the connection between CI tools and components presented in Table 13.
6. Connect the CI components and optional tools.
7. Get the connection between CI framework and optional tools.
8. List NFRs with the components shown in the 10 CI approaches.
9. Migrate the same connection to the CI framework.
10. Get the connection between CI components and NFRs.

As shown in Fig. 17, the CI framework contains eight components, which are “source code management,” “version control system,” “continuous integration,” “static code analysis,” “test automation,” “cloud/virtualisation platform,” “artifacts/dependencies management,” and “issue tracking.”

Each CI component of the CI framework is mapped to alternative tools and associated NFRs. These components were through analysis of previously published approaches and finding commonalities between them. For instance, the “source code management” component connects to alternative tools e.g. Gerrit, Github, Bitbucket etc. This CI component is used to review code changes and handle pull requests, and code review enhanced the quality of the source code. The component was then aligned with suitable NFR’s using the sources found in the SLR as well as logical reasoning based on these sources. For instance, higher source code quality reduced the maintenance work of the system which implies system maintainability is improved [11]. As such, we can conclude that better “source code management” has a positive effect on system maintainability [10,40].

6.3. Perceived benefits

As shown in Fig. 17, the proposed CI framework provides insights which components that are the most suitable to use for NFR testing

within the CI environment. This information is valuable to guide industrial practitioners to establish, or improve their existing, CI environments and to improve their NFR testing.

The *perceived benefits* of the proposed CI framework are summarised below:

- An overview of the components and the design of a suitable CI environment.
- An extensive list of suggested tools that can be used to design a CI environment.
- Visualisation of which component could potentially impact what NFRs of a product.
- Help practitioners design, improve, and transform their CI environments to make NFR testing more effective and efficient.

6.4. Framework usage

The proposed CI framework, as we predicted, can be applied in both industrial cases and academic studies. From industrial aspect, developers use this baseline CI framework to design their own CI pipeline flow and select suitable software tools for the purpose of improving NFR testing. Managers use it to track the trend of specific NFRs (e.g. security, performance), and collect real-time quality data by using the CI framework for decision making.

From academic perspective, we propose a baseline design for a CI environment focussed on the quality aspects. On top of this baseline design, researchers could add various metrics to measure quality attributes in CI components, e.g. correlation between lines of code and maintainability. Furthermore, some monitoring probes can be added in CI components to precisely measure NFRs, for example, if a critical security vulnerability is detected by NFR testing, a CI tool automatically creates a new defect ticket and put high priority.

In summary, the proposed framework is designed to support NFR testing and could for instance be used following these proposed steps:

- define a list NFRs to test in a prioritised order.
- select appropriate tools for each CI component based on the requirements, financial budget, tool's features etc. for the given context.
- integrate all selected tools.
- configure the tools to work as a complete environment.
- automate NFR testing using the chosen components and given the ordered list defined in step 1.
- monitor the NFRs using conventional metrics.

Note, at the time of publication, these steps have not yet been verified or validated based on efficiency. The perceived benefits and proposed steps are planned to be verified and validated in the future work.

7. Conclusion and future work

This section presents the conclusions of this SLR, the CI framework proposal, and the future work.

7.1. SLR findings

We have concluded three main findings through this SLR:

- *The CI environment is underutilised for NFR testing.* The possible causes come from two parts: (1) from a tooling perspective - CI tools are lacking to support the testing of NFRs, and some tools are not easy to integrate and maintain in the CI environment. (2) from NFR aspect, some NFR, e.g. usability, is just hard to test and automate, as we have mentioned before.
- Nine NFRs (maintainability, efficiency, reliability, productivity, performance, scalability, stability, availability and latency), which are testable by using CI environments, have been reported by previous studies, but *there are many other NFRs (e.g. portability and compatibility) are still unknown, which requires more research work.* In the nine NFRs, some NFR (e.g. efficiency) is more popular than the others. One of the reasons we found is that there are more CI tools existing for that specific NFR which facilitates the test automation.
- *There is a very low ratio of industrial studies in CI-NFR testing compared to the academic studies.* We found eight challenges, but the most frequently reported one is that the CI environment is not stable, due to the fact that the integrated tools increase the complexity and maintainability of the environment. Further, some NFRs are evaluated by homemade tools or scripts which prevent the CI environment to be applicable in other companies on different projects. The same CI environment does not fit all different projects, due to there being different NFR priorities and demands. A unified and high-level CI framework for NFRs testing could help industrial practitioners to evolve their CI environments for various projects.

7.2. CI framework proposal

We have proposed a descriptive CI framework based on the findings through the SLR. The identified NFRs are mapped to the components of the CI framework. This CI framework proposal provides the big picture concerning the correlations between CI and NFRs, in addition to the information as to which software tools can be used in which component of the CI framework. Furthermore, the components in this CI framework gives a level of understanding where an organisation or team designs their CI environment in practice. The tools in this CI framework are helpful for teams or organisations while choosing tools to build a CI environment, and the NFRs in the framework visualise which NFRs are affected in specific CI components by using the appropriate tools.

7.3. Future work

Software qualities, or NFRs, are as important as the functionality of a software for it to provide value. This value can be external towards the customer, e.g. performance or security, and internal, e.g. maintainability or complexity. Given the importance of the qualities, more research on the topic, and how to utilise the CI environment to achieve quality, is thereby warranted. As such, we have identified open questions that have not been answered in this paper but that are of interest for future work.

As we mentioned, a lack of tools for NFR testing and the addition of more tools increases the CI tool-chain complexity and maintainability. These are the two major roadblocks for testing NFRs by using CI environments. The open questions should be addressed in the future. These include how to address CI tool-chain complexity for NFR testing, and which other NFRs can be evaluated by using CI environments except the nine identified NFRs.

We have already identified many tools to build a CI environment. A few optional tools have been provided for each component in the proposed CI framework, but the questions concerning which tool to select when designing a CI environment, and which strategy or principle should be considered while selecting tools are not answered. These questions place requirements on any extensions of the CI environment to be easily adaptable to a myriad of tools.

Additionally, we have already proposed the CI framework for NFRs testing, but it has not been verified and validated by any empirical study or industrial study. Additionally, there is a lack of data to support how many NFRs can be evaluated by utilising this CI framework for a system.

Acknowledgments

We would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

Appendix A. Selected papers in the systematic literature review

ID	Author(s)	Title	Paper Type	Year	Research Type
P1	Y. Ki and M. Song	An Open Source-Based Approach to Software Development Infrastructures	Conference	2009	Theoretical
P2	G. D. S. Pereira Moreira et al.	Software product measurement and analysis in a continuous integration environment	Journal	2010	Theoretical, Case study
P3	J. Humble and D. Farley	Continuous delivery: Huge benefits, but challenges too	Journal	2010	Theoretical
P4	A. Janus et al.	The 3C Approach for Agile Quality Assurance	Conference	2012	Theoretical, Case study
P5	J. Yu et al.	A Virtual Deployment Testing Environment for Enterprise Software Systems	Conference	2012	Theoretical, Case study
P6	P. Merson	Ultimate Architecture Enforcement: Custom Checks Enforced at Code-commit Time	Conference	2013	Theoretical
P7	C. A. Cois et al.	Modern DevOps: Optimizing software development through effective system interactions	Conference	2014	Theoretical
P8	J. J. Hu	The Verification and Validation of a Large-Scale System: Equipment TaaS as an Example	Conference	2014	Theoretical, Case study
P9	R. Ramler et al.	Automated Testing of Industrial Automation Software: Practical Receipts and Lessons Learned	Conference	2014	Industry report
P10	S. Krusche et al.	Rugby: an agile process model based on continuous delivery	Journal	2014	Theoretical, Case study
P11	A. Wahaballa et al.	Toward unified DevOps model	Conference	2015	Theoretical
P12	R. Nouacer et al.	EQUITAS: A tool-chain for functional safety and reliability improvement in automotive systems	Journal	2016	Theoretical, Case study
P13	G. LadÃ!‘nyi et al.	A software quality model for RPG	Conference	2015	Industry report
P14	J. Waller et al.	Including Performance Benchmarks into Continuous Integration to Enable DevOps	Journal	2015	Theoretical, Case study
P15	P. Dowling and K. McGrath	Using free and open source tools to manage software quality	Journal	2015	Theoretical
P16	M. Dlugi et al.	Model-based Performance Evaluations in Continuous Delivery Pipelines	Conference	2015	Theoretical
P17	M. Stillwell and J.G.F. Coutinho	A DevOps Approach to Integration of Software Components in an EU Research Project	Conference	2015	Theoretical, Case study
P18	N. Ferry et al.	Continuous deployment of multi-cloud systems	Journal	2015	Theoretical
P19	B. Vasilescu et al.	Quality and productivity outcomes relating to continuous integration in GitHub	Journal	2015	Industry report
P20	R. Abreu, H. Erdogmus, and A. Perez	CodeAware: Sensor-based Fine-grained Monitoring and Management of Software Artifacts	Conference	2015	Theoretical
P21	M. Du, S. Versteeg et al.	Interaction Traces Mining for Efficient System Responses Generation	Journal	2015	Theoretical, Case study
P22	N. Rathod, A. Surve	Test orchestration a framework for Continuous Integration and Continuous deployment	Conference	2015	Theoretical
P23	B. Vogel-Heuser et al.	Evolution of software in automated production systems: Challenges and research directions	Journal	2015	Theoretical
P24	A. Karapantelakis et al	DevOps for IoT Applications Using Cellular Networks and Cloud	Conference	2016	Theoretical
P25	K.-T. Rehmann et al.	Performance Monitoring in SAP HANAâs Continuous Integration Process	Journal	2016	Industry report
P26	A. Chandrasekaran and R. Jain	Patterns and commonalities in rapid system development methodologies	Journal	2016	Theoretical
P27	D. Preuveneers et al.	Systematic scalability assessment for feature oriented multi-tenant services	Journal	2016	Theoretical, Case study
P28	M. Soni	E2E Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery	Journal	2016	Theoretical
P29	J. Itkonen et al.	Perceived Benefits of Adopting Continuous Delivery Practices	Conference	2016	Theoretical, Case study
P30	E. Laukkanen et al.	Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization	Conference	2016	Theoretical, Case study
P31	E. Knauss et al.	Continuous Integration Beyond the Team: A Tooling Perspective on Challenges in the Automotive Industry	Conference	2016	Theoretical, Case study
P32	M. Ohtsuki et al.	Software Engineer Education Support System ALECSS Utilizing DevOps Tools	Conference	2016	Theoretical
P33	C. Rossi et al.	Continuous Deployment of Mobile Software at Facebook	Conference	2016	Industry report
P34	M. Arta et al.	Model-driven continuous deployment for quality DevOps	Journal	2016	Theoretical
P35	R. Ben Rayana et al.	GitWaterFlow: A Successful Branching Model and Tooling, for Achieving Continuous Delivery with Multiple Version Branches	Conference	2016	Theoretical
P36	P. KÃ Aa et al.	Exploring Peopleware in Continuous Delivery	Conference	2016	Theoretical
P37	S. Taheritanjani et al.	A Comparison between Commercial and Open Source Reference Implementations for the Rugby Process Model	Journal	2016	Theoretical
P38	S. Bougouffa et al.	Scalable cloud based semantic code analysis to support continuous integration of industrial PLC code	Conference	2017	Theoretical

(continued on next page)

P39	C. H. Kao	Continuous evaluation for application development on cloud computing environments	Journal	2017	Theoretical
P40	F. Deissenboeck et al.	The quamoco tool chain for quality modeling and assessment	Conference	2017	Theoretical, Case study
P41	A. Taryana et al	Pioneering the automation of lnternal quality assurance system of higher education (IQAS-HE) using DevOps approach	Journal	2017	Theoretical, Case study
P42	D. Perez-Palacin et al.	Quality assessment in DevOps: Automated analysis of a tax fraud detection system	Conference	2017	Industry report
P43	S. Vost and S. Wagner	Keeping continuous deliveries safe	Journal	2017	Theoretical
P44	M. Zolfagharinia et al.	Do Not Trust Build Results at Face Value: An Empirical Study of 30 Million CPAN Builds	Conference	2017	Industry report
P45	K. V. R. Paixao et al.	On the Interplay Between Non-functional Requirements and Builds on Continuous Integration	Conference	2017	Theoretical, Case study
P46	A. Janes, V. Lenarduzzi, and A. C. Stan	A Continuous Software Quality Monitoring Approach for Small and Medium Enterprises	Conference	2017	Theoretical, Case study
P47	V. Ferme and C. Pautasso	A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments	Conference	2017	Theoretical

References

[1] L. Chen, Continuous delivery: huge benefits, but challenges too, IEEE Softw. 32 (2) (2015) 50–54.
[2] J. Bosch, Continuous software engineering: an introduction, in: Continuous software engineering, Springer, 2014, pp. 3–13.
[3] B. Fitzgerald, K.-J. Stol, Continuous software engineering: a roadmap and agenda, J. Syst. Softw. 123 (2017) 176–189.
[4] N. Rathod, A. Surve, Test orchestration a framework for continuous integration and continuous deployment, in: Pervasive Computing (ICPC), 2015 International Conference on, IEEE, 2015, pp. 1–5.
[5] E. Knauss, P. Pelliccione, R. Heldal, M. Ågren, S. Hellman, D. Maniette, 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, ACM, 2016, p. 43.
[6] A. Miller, A hundred days of continuous integration, in: Agile, 2008. AGILE'08. Conference, IEEE, 2008, pp. 289–293.
[7] S. Dösinger, R. Mordinyi, S. Biffl, Communicating continuous integration servers for increasing effectiveness of automated testing, in: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ACM, 2012, pp. 374–377.
[8] A. Janes, V. Lenarduzzi, A.C. Stan, A continuous software quality monitoring approach for small and medium enterprises, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ACM, 2017, pp. 97–100.
[9] M. Shahin, M.A. Babar, L. Zhu, Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices, IEEE Access 5 (2017) 3909–3943.
[10] R. Abreu, H. Erdogmus, A. Perez, Codeaware: sensor-based fine-grained monitoring and management of software artifacts, in: Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, 2, IEEE, 2015, pp. 551–554.
[11] K.-T. Rehmann, C. Seo, D. Hwang, B.T. Truong, A. Boehm, D.H. Lee, Performance monitoring in SAP HANA's continuous integration process, ACM SIGMETRICS Perform. Eval. Rev. 43 (4) (2016) 43–52.
[12] L. Chung, J.C.S. do Prado Leite, On non-functional requirements in software engineering, in: Conceptual Modeling: Foundations and Applications, Springer, 2009, pp. 363–379.
[13] K.V.R. Paixão, C.Z. Felício, F.M. Delfim, M. de A. Maia, On the interplay between non-functional requirements and builds on continuous integration, in: Proceedings of the 14th International Conference on Mining Software Repositories, IEEE Press, 2017, pp. 479–482.
[14] D. Mairiza, D. Zowghi, N. Nurmuliani, An investigation into the notion of non-functional requirements, in: Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, 2010, pp. 311–317.
[15] D. Ståhl, J. Bosch, Modeling continuous integration practice differences in industry software development, J. Syst. Softw. 87 (2014) 48–59.
[16] P. Rodríguez, A. Haghighatkhah, L.E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J.M. Verner, M. Oivo, Continuous deployment of software intensive products and services: a systematic mapping study, J. Syst. Softw. 123 (2017) 263–291.
[17] E. Laukkanen, J. Itkonen, C. Lassenius, Problems, causes and solutions when adopting continuous delivery a systematic literature review, Inf. Softw. Technol. 82 (2017) 55–79.
[18] K.L. Gwet, Computing inter-rater reliability and its variance in the presence of high agreement, Br. J. Math. Stat.Psychol. 61 (1) (2008) 29–48.
[19] A. Agresti, Categorical data analysis. 2013, hoboken.
[20] J.L. Fleiss, Measuring nominal scale agreement among many raters, Psychol. Bull. 76 (5) (1971) 378.
[21] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Meas. 20 (1) (1960) 37–46.
[22] D.G. Altman, Practical Statistics for Medical Research, CRC press, 1990.
[23] J.R. Landis, G.G. Koch, The measurement of observer agreement for categorical data, Biometrics (1977) 159–174.
[24] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2014, p. 38.
[25] D.S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: 2011 International Symposium on Empirical Software Engineering and Measurement, IEEE, 2011, pp. 275–284.
[26] C. Robson, Real World Research, 3, Wiley, Chichester, 2011.
[27] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, S. Wagner, The quamoco tool chain for quality modeling and assessment, in: Software Engineering (ICSE), 2011 33rd International Conference on, IEEE, 2011, pp. 1007–1009.
[28] M. Soni, End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery, in: Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on, IEEE, 2015, pp. 85–89.
[29] V. Ferme, C. Pautasso, A declarative approach for performance tests execution in continuous software development environments, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ACM, 2018, pp. 261–272.
[30] R. Ramler, W. Putschögl, D. Winkler, Automated testing of industrial automation software: practical receipts and lessons learned, in: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation, ACM, 2014, pp. 7–16.
[31] C. Rossi, E. Shibley, S. Su, K. Beck, T. Savor, M. Stumm, Continuous deployment of mobile software at facebook (showcase), in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2016, pp. 12–23.
[32] C.A. Cois, J. Yankel, A. Connell, Modern DevOps: Optimizing software development through effective system interactions, in: Professional Communication Conference (IPCC), 2014 IEEE International, IEEE, 2014, pp. 1–7.
[33] N. Ferry, F. Chauvel, H. Song, A. Solberg, Continous deployment of multi-cloud systems, in: Proceedings of the 1st International Workshop on Quality-Aware DevOps, ACM, 2015, pp. 27–28.
[34] C.H. Kao, Continuous evaluation for application development on cloud computing environments, in: Applied System Innovation (ICASI), 2017 International Conference on, IEEE, 2017, pp. 1457–1460.
[35] M. Dlugi, A. Brunnert, H. Krcmar, Model-based performance evaluations in continuous delivery pipelines, in: Proceedings of the 1st International Workshop on Quality-Aware DevOps, ACM, 2015, pp. 25–26.
[36] E. Laukkanen, T.O.A. Lehtinen, J. Itkonen, M. Paasivaara, C. Lassenius, Bottom-up adoption of continuous delivery in a stage-gate managed software organization, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2016, p. 45.
[37] J. Itkonen, R. Udd, C. Lassenius, T. Lehtonen, Perceived benefits of adopting continuous delivery practices, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2016, p. 42.
[38] D. Preuveneers, T. Heyman, Y. Berbers, W. Joosen, Systematic scalability assessment for feature oriented multi-tenant services, J. Syst. Softw. 116 (2016) 162–176.
[39] P. Kärpänoja, A. Virtanen, T. Lehtonen, T. Mikkonen, Exploring peopleware in continuous delivery, in: Proceedings of the Scientific Workshop Proceedings of XP2016, ACM, 2016, p. 13.
[40] A. Janus, A. Schmietendorf, R. Dumke, J. Jäger, The 3c approach for agile quality assurance, in: Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics, IEEE Press, 2012, pp. 9–13.
[41] R. Nouacer, M. Djemal, S. Niar, G. Mouchard, N. Rapin, J.-P. Gallois, P. Fiani, F. Chastrette, A. Lapitre, T. Adriano, et al., Equitas: a tool-chain for functional safety and reliability improvement in automotive systems, Microprocess. Microsyst. 47 (2016) 252–261.
[42] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015, pp. 805–816.
[43] R.B. Rayana, S. Killian, N. Trangez, A. Calmettes, Gitwaterflow: a successful branching model and tooling, for achieving continuous delivery with multiple version branches, in: Proceedings of the 4th International Workshop on Release Engineering, ACM, 2016, pp. 17–20.

[44] S. Bougouffa, S. Diehm, M. Schwarz, B. Vogel-Heuser, Scalable cloud based semantic code analysis to support continuous integration of industrial PLC code, in: Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on, IEEE, 2017, pp. 621–627.
[45] M. Stillwell, J.G.F. Coutinho, A devops approach to integration of software components in an EU research project, in: Proceedings of the 1st International Workshop on Quality-Aware DevOps, ACM, 2015, pp. 1–6.
[46] A. Taryana, I. Setiawan, A. Fadli, E. Murdyantoro, Pioneering the automation of lnternal quality assurance system of higher education (IQAS-HE) using devops approach, in: Sustainable Information Engineering and Technology (SIET), 2017 International Conference on, IEEE, 2017, pp. 259–264.
[47] M. Ohtsuki, K. Ohta, T. Kakeshita, Software engineer education support system ALECSS utilizing DevOps tools, in: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, ACM, 2016, pp. 209–213.
[48] P. Merson, Ultimate architecture enforcement: custom checks enforced at code–commit time, in: Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity, ACM, 2013, pp. 153–160.
[49] Y. Ki, M. Song, An open source-based approach to software development infrastructures, in: Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on, IEEE, 2009, pp. 525–529.
[50] G. Ladányi, Z. Tóth, R. Ferenc, T. Keresztesi, A software quality model for RPG, in: Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, IEEE, 2015, pp. 91–100.
[51] J. Waller, N.C. Ehmke, W. Hasselbring, Including performance benchmarks into continuous integration to enable DevOps, ACM SIGSOFT Softw. Eng. Notes 40 (2) (2015) 1–4.
[52] A. Debbiche, M. Dienér, R.B. Svensson, Challenges when adopting continuous integration: acase study, in: International Conference on Product-Focused Software Process Improvement, Springer, 2014, pp. 17–32.
[53] B. Vogel-Heuser, A. Fay, I. Schaefer, M. Tichy, Evolution of software in automated production systems: challenges and research directions, J. Syst. Softw. 110 (2015) 54–84.
[54] Q. Requirements, International Standard Iso/Iec 25021 (2012). 10.1109/IEEESTD.2015.7106438.
[55] S. Taheritanjani, S. Krusche, B. Bruegge, A comparison between commercial and open source reference implementations for the rugby process model (2016).
[56] J.-J. Hu, The verification and validation of a large-scale system: equipment TaaS as an example, in: Computer, Consumer and Control (IS3C), 2014 International Symposium on, IEEE, 2014, pp. 13–18.
[57] P. Dowling, K. McGrath, Using free and open source tools to manage software quality, Queue 13 (4) (2015) 20.