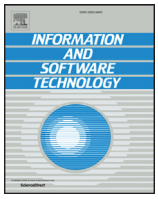
Information and Software Technology 121 (2020) 106268

Contents lists available at ScienceDirect

Information and Software Technology

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



Test Case Prioritization in Continuous Integration environments: A systematic mapping study

Jackson A. Prado Lima*, Silvia R. Vergilio

Department of Informatics, Federal University of Paraná (UFPR), CP: 19081, CEP: 81.531-980, Curitiba, Brazil

ARTICLE INFO

Keywords:
Software testing
Continuous Integration
Test Case Prioritization

ABSTRACT

Context: Continuous Integration (CI) environments allow frequent integration of software changes, making software evolution more rapid and cost-effective. In such environments, the regression test plays an important role, as well as the use of Test Case Prioritization (TCP) techniques. Such techniques attempt to identify the test case order that maximizes certain goals, such as early fault detection. This research subject has been raising interest because some new challenges are faced in the CI context, as TCP techniques need to consider time constraints of the CI environments.

Objective: This work presents the results of a systematic mapping study on Test Case Prioritization in Continuous Integration environments (TCPCI) that reports the main characteristics of TCPCI approaches and their evaluation aspects.

Method: The mapping was conducted following a plan that includes the definition of research questions, selection criteria and search string, and the selection of search engines. The search returned 35 primary studies classified based on the goal and kind of used TCP technique, addressed CI particularities and testing problems, and adopted evaluation measures.

Results: The results show a growing interest in this research subject. Most studies have been published in the last four years. 80% of the approaches are history-based, that is, are based on the failure and test execution history. The great majority of studies report evaluation results by comparing prioritization techniques. The preferred measures are Time and number/percentage of Faults Detected. Few studies address CI testing problems and characteristics, such as parallel execution and test case volatility.

Conclusions: We observed a growing number of studies in the field. Future work should explore other information sources such as models and requirements, as well as CI particularities and testing problems, such as test case volatility, time constraint, and flaky tests, to solve existing challenges and offer cost-effective approaches to the software industry.

1. Introduction

With the adoption of the agile paradigm by most software organizations, we observe a growing interest in Continuous Integration (CI) environments. Such environments allow more frequent integration of software changes, making software evolution more rapid and cost-effective [1]. This because they automatically support tasks like build process, test execution, and test results report. The results are used to resolve problems and locate faults, and rapid feedback is fundamental to reduce development costs [41].

Within an integration cycle (usually called build), regression testing is an activity that takes a significant amount of time. A test set, many times, includes thousands of test cases whose execution takes several hours or days [42].

To help in the regression testing task, we find in the literature some techniques, which are usually classified into three main categories [2]: minimization, selection, and prioritization. Techniques based on Test Case Minimization (TCM) usually remove redundant test cases, minimizing the test set according to some criterion. Test Case Selection (TCS) selects a subset of test cases, the most important ones to test the software. Test Case Prioritization (TCP) attempts to re-order a test suite to identify an "ideal" order of test cases that maximizes specific goals, such as early fault detection. TCP techniques are very popular in the industry and are the focus of our work.

Existing TCP techniques can be applied in CI environments. However, most of them require adaptations to deal with the testing budget,

* Corresponding author
E-mail addresses: japlima@inf.ufpr.br (J.A. Prado Lima), silvia@inf.ufpr.br (S.R. Vergilio).

https://doi.org/10.1016/j.infsof.2020.106268
Received 13 July 2019; Received in revised form 20 January 2020; Accepted 23 January 2020
Available online 25 January 2020
0950-5849/

limited resources, and constraints of the CI environments. For instance, the application of search-based techniques or other ones that require extensive code analysis and coverage may be unfeasible due to the time constraints and the test budget for a build.

We can see that the intersection of TCP and CI (TCPCI - *Test Case Prioritization in Continuous Integration environments*) poses some difficulties. Such a subject has been raising interest, and approaches to solve the problem have been proposed in the literature. However, we have not found works reporting the characteristics of such approaches, if they differ from existing TCP approaches, and how they have been evaluated. To provide such a general overview is important to identify challenges faced, gaps, and trends to guide research on this subject and to propose new ways to perform regression testing within integration cycles. Motivated by these facts, this work presents the results of a mapping study on TCPCI.

We adopted the process of Petersen et al. [3]. We followed a research plan including research questions, inclusion and exclusion criteria, construction of the search string, and selection of known search databases. We found 35 primary studies, and we analyzed the results according to three aspects: i) *basic information of the field*, such as evolution along the years, main authors and publication fora; ii) *characteristics of the approaches*: main goal, kind of used TCP technique, programming language, addressed CI testing problems such as flaky and time-consuming tests, as well as some CI particularities related to the testing budget and constraints, parallelism, multiple test requests, and volatility of test cases; and iii) *evaluation aspects*: such as evaluation contexts and measures used for comparison. In addition to this, some research trends and gaps were identified that allow researchers to direct future investigations.

The results show an increasing number of papers on this research subject. Most studies have been published in the last four years. 80% of the approaches are history-based, that is, are based on the failure and test execution history. The great majority of studies report evaluation results by comparing prioritization techniques, and we also found studies that use TCP after (or in combination) with TCS. The preferred measures are Time and number or percentage of Faults Detected (FD). Few studies consider CI characteristics and testing problems.

Then the contribution of this mapping is twofold: i) to present the main characteristics of the TCPCI approaches according to a classification schema generated interactively during the analysis of the studies found; and ii) to help researchers in the identification of research opportunities and encourage new works on this subject to solve existing challenges and offer to the software industry cost-effective approaches.

The remaining of this paper is organized as follows. Section 2 overviews background and related work. Section 3 describes the adopted mapping process. Section 4 analyses the results. Section 5 points out trends and identified research opportunities. Section 6 discusses threats to validity. Section 7 concludes the paper.

2. Background and related work

This section first introduces the fields explored in our mapping: Test Case Prioritization (TCP) and Continuous Integration environments, and after, reviews related work.

2.1. Test Case Prioritization

According to Rothermel et al. [4] the TCP problem can be formulated as:

Definition 2.1. Given T, a test suite, PT, the set of all possible permutations of T, and f, a function that determines the performance of a given prioritization from PT to real numbers. Find:

$$T' \in PT \text{ s.t. } (\forall T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$$

TCP problem aims to find the best T' that achieves certain specific goals [4].

This technique allows the most crucial test cases are first executed. This characteristic is interesting (and suggested), given that in a regression testing scenario, there are limited resources, and it may not be possible to execute the entire regression test suite [2,4]. TCP techniques consider all the test suite and can be, in some cases, time-consuming. But such a characteristic is also an advantage because the coverage is maintained. Besides, in the presence of time and cost constraints for the test activity that hinders the execution of all test cases, TCP allows failing test cases are executed first. This contributes to reduce resources and costs spent in the test, as well as ensures that the maximum possible fault coverage is achieved.

According to Rothermel et al. [4], some objectives of TCP techniques are: (i) to increase the fault detection rate of a test suite already in the beginning of the regression test execution; (ii) to increase the system code coverage under test; (iii) to increase high-risk fault detection rate; and (iv) to increase the probability to reveal faults related with specific code changes. To achieve such objectives, many TCP techniques were proposed in the literature. According to the information used in the prioritization, they can be classified into different categories [2] that we use to categorize the approaches found in our mapping. They are: Cost-aware, Coverage-based, Distribution-based, Human-based, History-based, Requirement-based, Model-based, and Probabilistic. An overview of these categories is presented in Section 3.5, which describes our classification schema.

2.2. Continuous Integration environments

In the past, some developers worked alone (separately) for a long time and only integrated their changes to the master branch when they completed their work. However, this practice presents some disadvantages. It is time-consuming, adds unnecessary bureaucratic cost to the projects, and results in the accumulation of uncorrected errors for long periods. These factors hampered a rapid distribution of updates to customers.

With the advent of the agile development paradigm, Continuous Software Engineering practices have become popular and adopted by most organizations, such as Continuous Integration (CI), Continuous Deployment (CD), and Continuous DElivery (CDE), allowing frequent integration, test of the changed code, deployment, and quick feedback from the customer in a very rapid cycle. The relationship between such practices is illustrated in Fig. 1.

CI is an essential practice adopted before deployment and delivery. CI environments contain automated software building and testing [6], helping to scale up headcount and delivery output of engineering teams, as well as allowing software developers to work independently on features in parallel. When they are ready to merge these features into the end product, they can do this independently and rapidly. Among the popular open-source CI servers we can mention: Buildbot [7], GoCD [8], Integrity [9], Jenkins [10], and Travis CI [11].

CDE aims at packing an artifact (the production-ready state from the application) to be delivered for Acceptance Test. In this sense, the artifact should be ready to be released to end-users (production) at any given time. On the other hand, CD is responsible for automatically packing, launching and distributing the software artifact to production. We can observe that CI ensures that the artifact used in the CD and CDE successfully passed the integration phase.

In order to provide a swift and cost-effective way to validate and launch new software updates, improving fault detection and software quality, regression testing is an essential activity in CI environments. This is because to enable rapid test feedback in CI, test cycles are restricted to a specific (short) duration. We refer to this time duration of each test cycle as a time budget. Time budgets can vary from a cycle to a cycle, and they include time to select relevant tests for running, to run the tests, and to report test results to developers. In this way, traditional TCP techniques require some adaptations to be applied in the CI context. The techniques need to consider specific particularities of CI

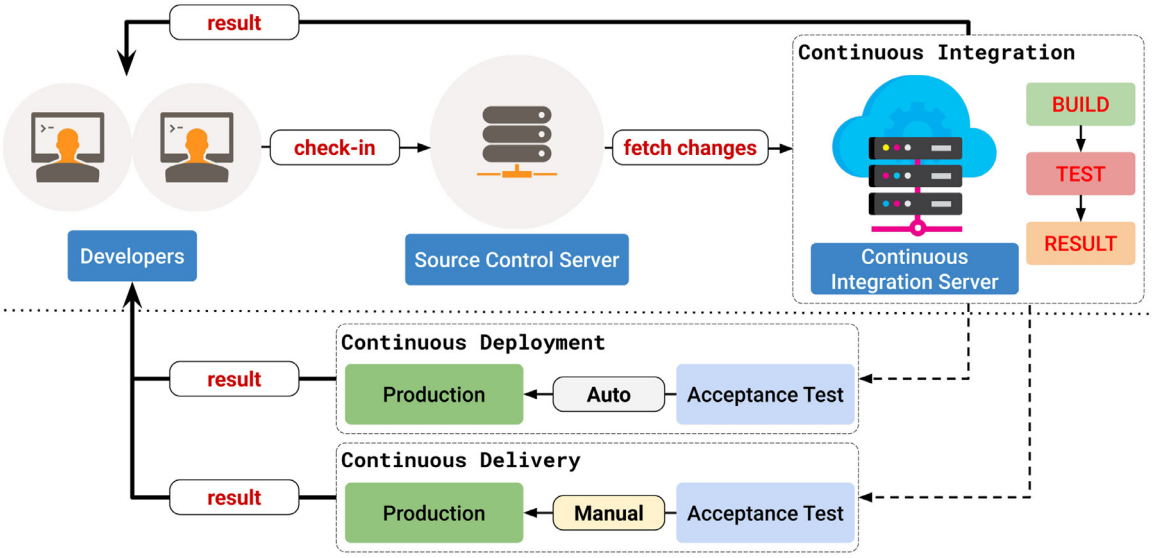


Fig. 1. Overview of the relationship between Continuous Integration, Delivery, and Deployment (adapted from Shahin et al. [5]).

Table 1
Comparison of our work with existing secondary studies.

Work	Focus	Type	NS	Pub. Year	Period covered
Hellmann et al. [17]	Agile Testing	Mapping	166	2012	2003-2011
Eck et al. [19]	CI	Review	43	2014	Not Identified
Ståhl and Bosch [20]	CI	Review	46	2014	2003-2013
Mäntylä et al. [18]	Rapid Release	Review	24	2015	2003-2013
Karvonen et al. [12]	Agile Release	Review	71	2017	2005-2015
Shahin et al. [5]	CI	Review	69	2017	2004-2016
Yoo and Harman [2]	TCM, TCS, and TCP	Survey	159	2012	1977-2009
Singh et al. [13]	TCP	Review	65	2012	1997-2011
Catal et al. [14]	TCP	Mapping	120	2013	2001-2011
Kumar and Singh [15]	TCP	Survey	11	2014	2001-2013
Khatibsyarbini et al. [16]	TCP	Review	80	2018	1999-2016
Our work	TCP and CI	Mapping	35	2019	2009-2019

environments like as: parallel execution of test cases and allocation of resources, the volatility of test cases, that is, the test cases can be added and removed in subsequent commits, the detection of as many faults as possible in a short interval of time. To address all these particularities, some approaches have emerged in the literature, being TCPCI an emergent research topic that motivates our mapping. 3

2.3. Related work

In the literature we can find some mappings, surveys and systematic reviews on continuous integration practices and environments [5,12], on regression testing [2], test case prioritization techniques in general [13–16], and on software testing in agile and continuous integration context [17–20]. Such works do not specifically address the subject test case prioritization in continuous integration environments. The work of Shahin et al. [5] mentions some studies concerning TCP that were included in our mapping, however, the focus of their work is neither TCP techniques nor regression testing techniques.

We summarize the key aspects of the above mentioned studies in Table 1, which presents, for each work, the corresponding research focus, type of study, the number of primary studies (NS) found, the publication year, and the period covered. In short, our work differs from existing ones because our focus is on TCPCI. Our mapping offers details and characteristics of the TCPCI studies, presents research gaps and trends that can help the researchers in future investigations. We can not find such results in the works most related to ours, which map the general areas in a separated way.

3. Mapping process

In this section, we describe the main steps of our mapping, conducted following the guidelines proposed by Petersen et al. [3].

First of all, and to justify our mapping, we conducted a search for related work with the following string: *test AND prioritization AND (map OR mapping OR review OR survey)*. We did not find work with the same goal of ours. Given this fact, which justifies and shows the need and relevance of our mapping, we performed the steps described next.

3.1. Definition of Research Questions

To overview existing research on TCPCI, we used three groups of research questions. The first one provides basic information about the primary studies. The second group characterizes the approaches found in the studies. The third group refers to the evaluation aspects of each proposal. The research questions from each group are presented below.

RQ1: Basic information of the field

RQ1.1: *In which fora is the research on TCPCI published?* This question allows the identification of the target public, as well as the main fora where research on TCPCI can be found and published.

RQ1.2: *How have the number and frequency of publications evolved over the years?* This question aims to analyze the interest in TCPCI over the years and to assess how relevant and active this topic is, as well as the maturity of the research being conducted.

RQ1.3: *What are the main research groups?* This question aims to identify the main research groups as well as authors and countries.

Table 2
Control group.

Year	Authors	Title	Citations
2009	Jiang et al. [43]	How Well Do Test Case Prioritization Techniques Support Statistical Fault Localization	79
2012	Jiang et al. [44]	How Well Does Test Case Prioritization Integrate with Statistical Fault Localization?	41
2013	Marijan et al. [45]	Test Case Prioritization for Continuous Regression Testing: An Industrial Case Study	73
2014	Elbaum et al. [46]	Techniques for Improving Regression Testing in Continuous Integration Development Environments	157
2016	Busjaeger and Xie [47]	Learning for Test Prioritization: An Industrial Case Study	27
2017	Spieker et al. [48]	Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration	33
2018	Liang et al. [49]	Redefining Prioritization: Continuous Prioritization for Continuous Integration	11
2018	Haghighatkhah et al. [42]	Test prioritization in continuous integration environments	4

Table 3
Inclusion and exclusion criteria.

Inclusion criteria	
I1	The paper is related to Software Engineering area;
I2	The paper is related to TCPCI.
Exclusion criteria	
E1	Out of scope, not related to TCPCI;
E2	Not available online;
E3	Not in English;
E4	Abstracts, posters, reviews, conference reviews, chapters, thesis, keynotes, and doctoral symposiums.

RQ2: Characteristics of the approach

RQ2.1: *What types of approaches are proposed?* As mentioned in Section 2, the approaches can be classified according to the information used in the prioritization. This question aims to identify the most common type of TCPCI approach and research opportunities.

RQ2.2: *What are the application contexts?* This question aims to identify application aspects related to: the CI environment particularities, addressed testing problems, programming language, assumptions, requirements, and limitations. This question aims to identify possible research gaps and to point the need for future work.

RQ3: Evaluation aspects

RQ3.1: *How have the approaches been evaluated?* This question analyses different aspects of the conducted evaluation, characterizing: used systems and datasets, threats to validity, baseline approach used in the comparison, etc. The evaluation measure is an important aspect that is treated in a separate question.

RQ3.2: *What are the used evaluation measures?* This question identifies all the measures used in the evaluation and whether their values are statistically analyzed. Most measures are based on the fault detection ability, but in the context of CI environments, the runtime is an important aspect to be evaluated.

3.2. Definition of the search string

We formulated our search string, considering the mapping goals, and posed research questions. The resulting search terms were composed of synonymous for the main terms “Test Prioritization” and “Continuous Integration” taking into account different spelling. Then, we constructed the search string using logical operators, “OR” and “AND”. After tests, we adopted the following search string that returned the greatest number of relevant articles.

(“continuous integration”) **AND** (prioritization *OR* prioritisation) **AND** (test *OR* testing)

To verify the accuracy of the keywords chosen to compose the search string, we used a control group. Such a group comprises a set of previously known studies, which are presented in Table 2 along with the number of citations extracted from Google Scholar on 1st October 2019.

3.3. Selection criteria

Table 3 presents the adopted inclusion and exclusion criteria. In summary, we screened studies written in English, available online, and verifying each one that fits the goals of our mapping.

3.4. Conducting the review

The search started and finished in October 2019. The review was conducted in a set of steps, presented in Fig. 2 following the PRISMA (Preferred Reporting Items for Systematic reviews and Meta-Analyses) statement [21]. In such a figure, we present three data sources used in our mapping: Digital Libraries, Grey Literature, and Snowballing.

First, we selected primary studies by using the search string in the digital libraries, considering the title, abstract, and keywords. The engines for performing the search are online repositories that were chosen due to their popularity and because they provide many leading software engineering publications. However, some of them required adaptations, for example, searching in title, abstract, and keywords individually.

In order to identify the state-of-the-art and -practice in the subject of this mapping, we included a search in the grey literature [22,23] (non-published, nor peer-reviewed sources of information) seeking relevant white papers, industrial (and technical) reports. However, the use of grey literature can lead to unreasonably grow in the number of studies. In addition to this, the quality assessment of the studies found in the grey literature is more difficult. Because of this, we reduced the possible noise in the search, i.e. ignoring theses and incomplete studies, once that there is a high possibility that a thesis produces articles as results. Additionally, we performed a snowballing reading (forward and backward snowballing procedures, following Wohlin’s guidelines [24]) by searching in the control group. In this step, we found five papers.

We did not define an initial publication date for the studies, then all the returned papers were included. Table 4 presents the data sources used and the period covered, with the data separated by Digital Libraries, Grey Literature, and Snowballing. In total, we performed the mapping in 17 data sources. As we can see, a total of 818 studies were found, including the period from 1979 to 2020.

In the second step, we removed repeated studies, remaining 687 ones. Then we applied the selection criteria and obtained 276 studies. The selection was conducted by the first author, who read the title, abstract, and index terms of the papers (keywords). In case of doubts, read-

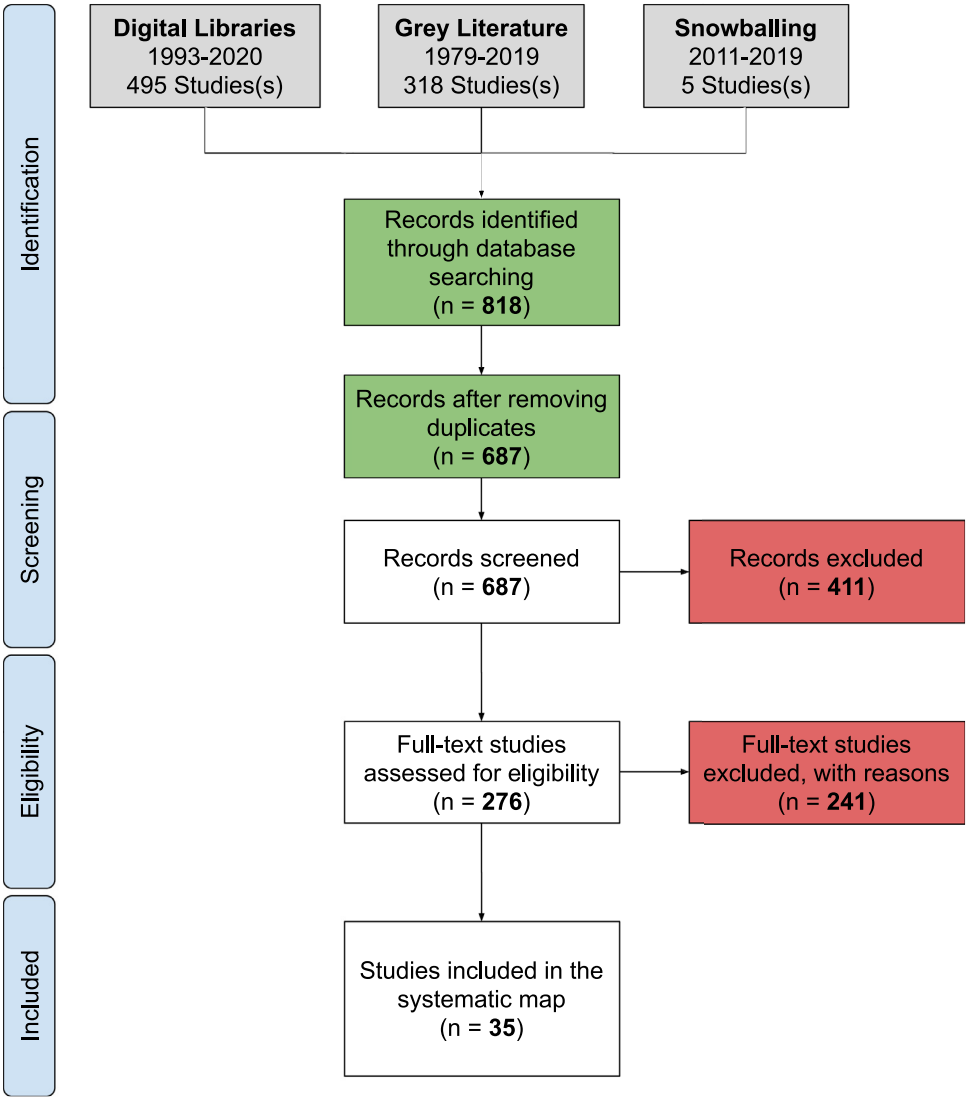


Fig. 2. PRISMA flow diagram. For each data source, the number of studies found is presented, as well as the range of publication data from these studies.

Table 4
Number of papers returned by each source.

Source	Studies	Period covered
Digital libraries		
ACM	24	2009-2019
EBSCOhost	3	2012-2018
Ei Compendex	31	2009-2019
IEEEXplore	20	2009-2019
ISI Web of Science	27	2009-2019
MIT Libraries	20	2009-2019
ScienceDirect	2	2012-2018
Scopus	35	2009-2019
Springer Link	273	1993-2020
Wiley Online Library	60	2007-2019
Grey literature		
AWS Whitepapers & Guides	0	-
Google AI	0	-
Google Cloud	2	2018-2019
Google Scholar	129	2001-2019
Microsoft Academic	2	2009-2018
Microsoft Research	0	-
Science.gov	185	1979-2019
Snowballing		
Forward and Backward procedures	5	2011-2019
Total	818	1979-2020

ing in the following order was performed: introduction, conclusion, and the entire paper. If such a doubt persists, meetings and discussions with the second author were accomplished to solve it. The second author also revised the decisions of the first one. In the end, 35 papers remain. The selected papers are presented in the end of this paper (*Primary Studies (PS)* Section).

3.5. Classification scheme, data extraction and dissemination

We collected the following information to address the research questions: title, authors, institution, publication venue, publication year, and depending on the study type (theoretical or experimental) other data were collected such as all the characteristics of the techniques used by the approach and conducted evaluation. The raw data were analyzed and disseminated in the Open Science Framework (OSF) [25].

In order to avoid the researchers' bias threat during the extraction, which can affect negatively the results, we used a classification scheme, derived interactively, which encompasses four dimensions to classify the studies found. The schema is detailed in Table 5 with a brief description of each category. Regarding the characteristics of the approach (RQ2), we use the dimensions: research goal, information source, and CI testing problems. The categories used in the first dimension were adapted from the classification proposed by Catal and Mishra [14]. In the second dimension, the categories were extracted from the work of Yoo and Harman [2], and in the third one, we used the problems reported by

Table 5
Classification schema.

Dimension/Category	Description
	RQ2
Research goal	
Prioritization	The study proposes a prioritization technique.
Measurement	The study proposes an evaluation measure to evaluate prioritization technique.
Comparison	The study compares prioritization techniques.
Minimization	The study combines prioritization and minimization techniques.
Selection	The study combines prioritization and selection techniques.
Localization	The study uses the prioritization technique for fault localization.
Time-Limit Treatment	The study considers time constraint in Continuous Integration environments.
Others	The study has a different goal not included in the other categories.
Information source	
Cost-aware	The approach prioritizes test cases based on the cost of the test cases, because their costs cannot be equal.
Coverage-based	The approach prioritizes test cases based on the code coverage.
Distribution-based	The approach prioritizes test cases based on the distribution of the test case profiles.
Human-based	The approach prioritizes test cases based on factors that (human) testers deem the most important.
History-based	The approach prioritizes test cases based on test case execution history information and code changes.
Requirement-based	The approach prioritizes test cases based on information extracted from requirements.
Model-based	The approach prioritizes test cases based on information extracted from models, such as UML (Unified Modeling Language) models.
Probabilistic	The approach prioritizes test cases based on probabilistic theories.
Others	The approach prioritizes test cases based on other kind of information not included in the other categories.
CI testing problem	
Flaky tests	Tests that randomly fail sometimes.
Time-Consuming testing	Testing takes too much time.
User Interface (UI) testing	UI testing of the application.
Complex testing	Testing is complex, e.g., setting up the environment is complex.
	RQ3
Evaluation measure	
APFD	Average Percentage of Faults Detected
APFDc	Average Percentage of Faults Detected per Cost
NAPFD	Normalized Average Percentage of Faults Detected
Expense	Minimum percentage of statements in a program that must be examined to locate the fault.
FD	Number or percentage of Faults Detected.
Time	Time spent to execute the prioritization method or the test suite.
NA	The study does not use an evaluation measure.
Others	The study uses other types of evaluation measure not mentioned above.

Laukkanen et al. [26], which are more related to test in CI environments. To answer RQ3, we considered each measure used in the evaluation as a category.

4. Outcomes

This section presents the results of our mapping and analysis to answer our research questions. To this end, we derived Table 6 containing for each study a summary of our main findings: reference and year of the study, research goal, information considered by the approach, and used evaluation measure.

4.1. Basic information of the field

This section presents answers to the questions concerning basic Information of the field: Section 4.1.1 answers RQ1.1, Section 4.1.2 answers RQ1.2, and Section 4.1.3 answers RQ1.3.

4.1.1. RQ1.1 - Publication fora

We analyzed the publication fora, and we observed that 28 papers (80%) were published in events, and 7 papers in journals. We found 24 different publication venues. Most of them (16 (≈ 67%)) published only one paper, and 7 (29%) published two papers. The preferred journal is the Journal of Systems and Software (JSS) with two papers. Among the events that published more than one paper, we can mention: Asia-Pacific Software Engineering Conference (APSEC), International Computer Software and Applications Conference (COMPSAC), International Symposium on Foundations of Software Engineering (FSE), International Conference on Software Maintenance and Evolution (ICSME), International Conference on Software Testing, Verification and Validation (ICST), International Conference on Software Quality, Reliability and Security (QRS), and International Conference on Quality Software (QSIC). Fig. 3 shows the distribution of main venues: journals and events with more than two papers. The preferred fora is the FSE with five papers. We can see that the venues are all related to the general area of software engineering and, specifically, with software quality and maintenance.

4.1.2. RQ1.2 - Publications over the years

The number of publications over the years is depicted in Fig. 4 together with a trend-line. This figure also contains bars representing the venue (Journal or Event). The trend line shows a crescent interest in the topic. The first paper appeared in 2009, and since then, the interest in TCPCI has been increasing. The most significant number of publications were found in 2018, with 9 studies out of 35 (≈ 26%). Considering the search ended in October 2019, and that some studies of this year may

Table 6
Main findings This table presents the main findings for each study. The first column contains the reference of the study, followed by the year of publication in the second column. Columns 3 to 9 show the *Research Goals* of each study, where "P" means Prioritization, "C" Comparison, "S" Selection, "M" Minimization, "FL" Fault Localization, "TL" Time-Limit Treatment, and "O" Others. Columns 10 to 16 show the *Considered Information* during prioritization, where "Cost" means Cost-aware, "Cov" Coverage-based, "Dis" Distribution-based, "Hist" History-based, "Human" Human-based, "Prob" Probabilistic, and "O" Others. The last columns show the used *Evaluation Measures* (see Table 5), where "E" means Expense and "O" Others.

Ref.	Year	Research goal							Considered information							Evaluation measure						
		P	C	S	M	FL	TL	O	Cost	Cov	Dis	Hist	Human	Prob	O	APFD	APFDc	NAPFD	E	FD	Time	O
[43]	2009	-	✓	-	-	✓	-	-	-	✓	✓	-	-	-	✓	✓	-	-	✓	-	-	✓
[50]	2010	-	✓	-	-	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-	✓
[51]	2011	-	✓	-	-	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-	✓
[52]	2011	✓	-	✓	✓	-	✓	-	-	✓	-	✓	-	-	✓	-	-	-	-	-	✓	✓
[44]	2012	-	✓	-	-	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-	✓
[45]	2013	✓	✓	-	-	-	✓	-	✓	-	-	✓	✓	-	-	-	✓	-	-	✓	✓	-
[46]	2014	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-
[53]	2015	✓	✓	-	-	-	✓	-	✓	✓	-	✓	-	-	-	-	✓	-	-	-	-	-
[54]	2015	-	✓	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓
[55]	2016	-	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
[47]	2016	✓	✓	-	-	-	-	-	-	✓	-	✓	-	-	✓	✓	-	-	-	-	-	-
[56]	2016	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-
[57]	2016	-	-	✓	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓	✓
[58]	2016	✓	-	-	-	-	✓	-	-	-	-	✓	✓	-	✓	-	-	-	-	✓	✓	✓
[41]	2016	-	✓	-	-	✓	-	-	-	✓	✓	-	-	-	✓	-	-	-	✓	-	-	-
[48]	2017	✓	✓	-	-	-	✓	-	-	-	-	✓	-	-	✓	-	-	✓	-	-	-	-
[59]	2017	-	✓	-	-	-	✓	-	-	✓	-	✓	-	-	-	-	-	-	-	✓	-	✓
[60]	2017	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-
[61]	2017	✓	✓	-	-	-	✓	-	-	✓	-	✓	-	-	-	-	-	-	-	✓	✓	✓
[62]	2017	✓	-	-	-	-	✓	-	-	-	-	✓	✓	-	✓	-	-	-	-	✓	✓	✓
[63]	2017	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-
[64]	2018	✓	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	✓	-	✓	-	-
[65]	2018	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
[49]	2018	✓	✓	-	-	-	✓	-	✓	-	-	✓	-	-	-	-	✓	-	-	-	-	-
[66]	2018	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	✓
[42]	2018	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	✓	✓	-	-	-	-	✓	-
[67]	2018	✓	✓	-	-	-	-	-	✓	✓	-	✓	-	-	✓	✓	✓	-	-	✓	✓	-
[68]	2018	-	-	-	-	-	✓	✓	-	-	-	✓	✓	-	-	-	-	-	-	-	✓	✓
[69]	2018	✓	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	✓
[70]	2018	✓	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-
[71]	2019	-	-	✓	-	-	✓	-	-	✓	-	✓	-	-	-	-	-	-	-	✓	✓	✓
[72]	2019	✓	✓	✓	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	✓
[73]	2019	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-
[74]	2019	✓	-	✓	-	-	-	-	-	✓	-	✓	-	-	✓	✓	-	-	-	✓	-	✓
[75]	2019	✓	✓	-	-	-	-	-	-	-	-	✓	-	-	✓	-	✓	-	-	-	✓	-
Total		23	26	6	1	5	18	1	4	13	2	28	4	1	15	8	6	3	5	11	16	17

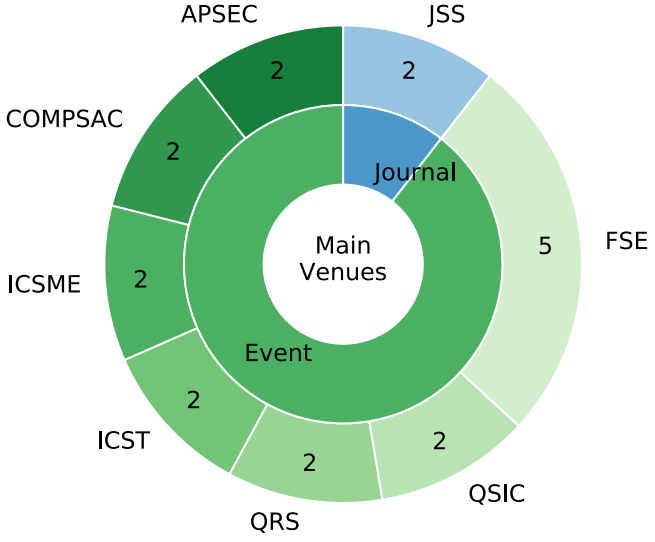


Fig. 3. Main publication fora with more than two papers.

not be included, we can see that 21 studies (≈ 60%) were published in the period 2016-2018. This corroborates the fact TCPCI is an emergent research-topic that has been arising interest in the last years.

Table 7
List of the most prominent authors.

Author	NS
Marijan, Dusica	8
Liaaen, Marius	6
Jiang, Bo	5
Chan, Wing Kwong	4
Gotlieb, Arnaud	4
Sen, Sagar	3
Tse, T. H.	3
Rothermel, Gregg	3

4.1.3. RQ1.3 - Research groups

We found 83 authors from 45 different institutions. We observe that the main research groups are in the Simula Research Laboratory that appears in 8 (23%) different studies, followed by Cisco appearing in 6 (17%) studies. The University of Hong Kong and City University of Hong Kong appearing in 4 (11%) studies, Beihang University appearing in 3 (9%) studies, and Concordia University, Sungkyunkwan University, Peking University, Mälardalen University, University of Nebraska, Westermo Research and Development, and Google in 2 studies.

Table 7 shows the main authors with more than two studies published and the number of corresponding studies (NS). Whilst, we identified 12 different countries. Table 8 presents the countries found and the corresponding number of studies (NS).

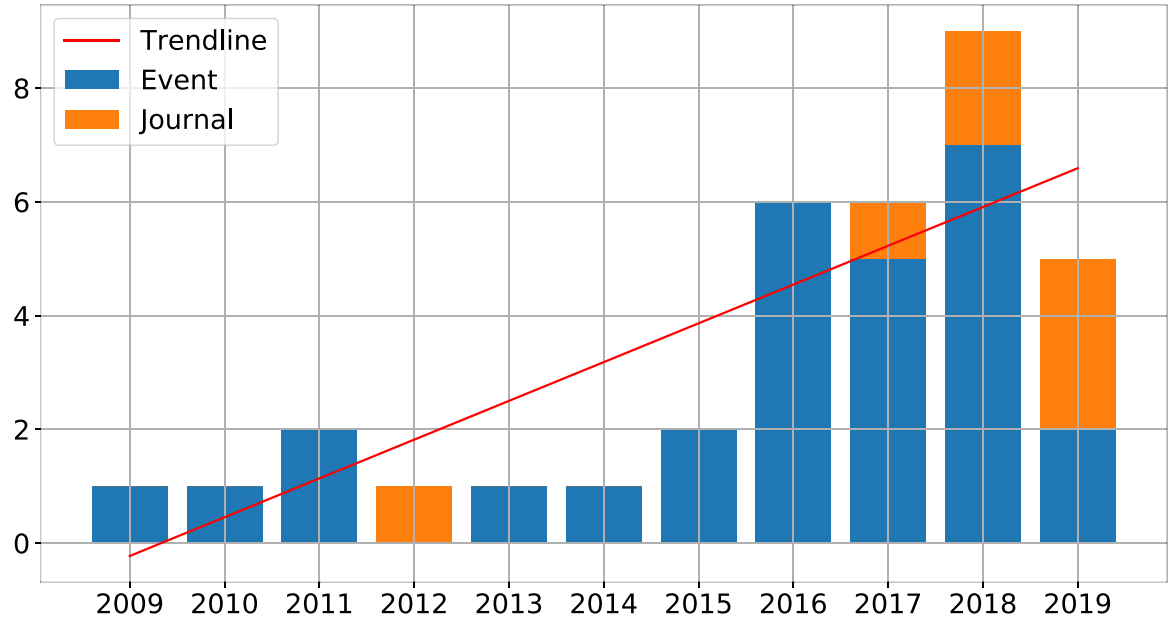


Fig. 4. Number of publications over the years.

Table 8
List of countries.

Country	NS
China	9
Norway	9
United States of America	5
Sweden	4
Republic of Korea	3
Australia	2
Canada	2
Finland	1
Germany	1
Pakistan	1
Switzerland	1
United Kingdom	1

Table 9
Number of information sources used in the prioritization process in TCPCI.

#	Perc.	Primary studies
1	9%	[57,65,66]
2	29%	[42,49,54–56,60,64,71–73]
3	31%	[45,46,48,52,61,63,67,69,70,74,75]
4	17%	[47,53,58,59,62,68]
6	14%	[41,43,44,50,51]

4.2. Characteristics of the approaches

In this section, we address the second group of questions concerning the characteristics of the approaches, answering RQ2.1 and RQ2.2, respectively, in Sections 4.2.1 and 4.2.2.

4.2.1. RQ2.1 - Types of approaches

To answer this question, we first identified the research goal of the found studies, according to the dimensions of our schema. The results can be viewed in the first columns of Table 6.

We can observe, most studies (26 out of 35, ≈ 74%) compare prioritization methods, followed by the introduction of a prioritization technique (23, ≈ 66%). 18 (≈ 51%) studies consider time constraints, 6 (≈ 17%) combine prioritization and selection methods, 5 (≈ 14%) use prioritization for fault localization, and we found 1 study that is included in the category Others and another one combining prioritization with minimization techniques. We did not find studies proposing, explicitly, new evaluation measures.

Some papers do not have a simple goal. For instance, some studies introduce a new technique and also compare it with existing ones. To provide better visualization of this intersection, we generated Fig. 5 that presents the interaction between goals and number of studies associated with each one.[1] In the figure, the bars represent the number of studies; each category is associated with one or more bars, and the bars are associated with one or more categories. When the bar is associated with more than one category, this means that there is an intersection between them, represented by a line with bullets. Consider the category `Comparison` (last row). There are 26 studies belonging to this category (horizontal bar in the left). The intersection with the other categories are represented in the right by bullets, 9 studies have intersection only with the category `Prioritization`; 8 studies with the categories `Prioritization` and `Time-Limit`; and so on.

As evidenced by Fig. 5, we observe a large number of studies comparing prioritization techniques. Besides that, we observe that all studies that aim fault localization (5 studies in the third bar) also compare techniques but without any other intersection; they do not propose any new technique, for instance. This can be considered a gap to be explored.

Regarding the information source considered by the approach and our schema (Table 5), we can see in Table 6 that we did not find works on the categories `Requirement-based` and `Model-based`. The great majority of the studies (28 out of 35 (80%)) are `History-based`, followed by the categories `Others` and `Coverage-based` with, respectively, ≈ 43% and ≈ 37% of studies, `Cost-aware` and `Human-based` with ≈ 11% each, `Distribution-based` with ≈ 6%, and `Probabilistic` with only 1 study (≈ 3%).

Tables 9 and 10 show the number of information sources used in the studies, as well as, the most commonly used.

In total, we identified a maximum of six sources used at the same time during the prioritization process and 30 different types of informa-

[1] The figure was build using UpSet [27].

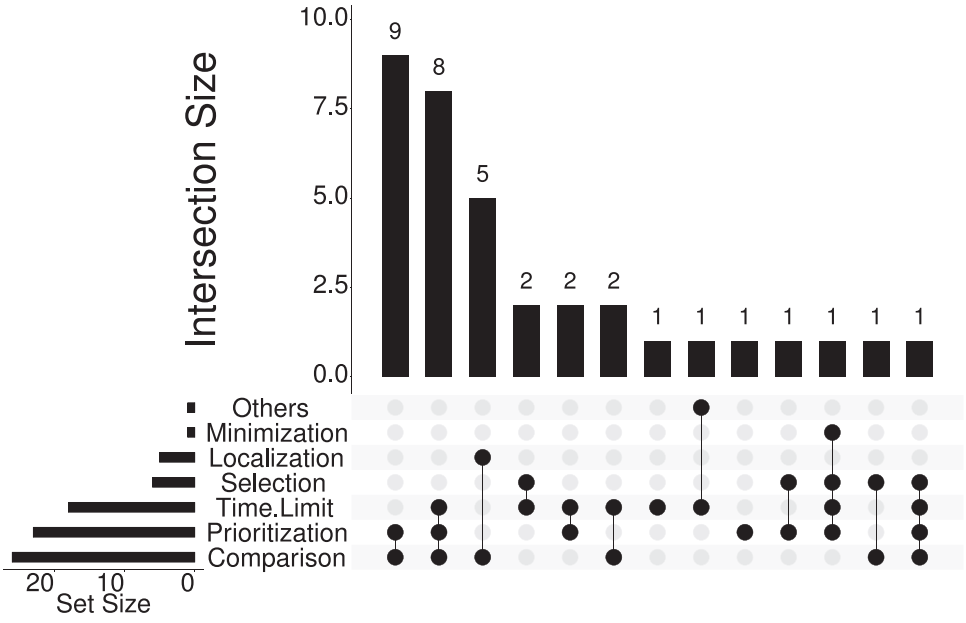


Fig. 5. Interaction between research goals.

Table 10
Most common information sources used in the prioritization process in TCPCI.

Category	Subcategory	Perc.	Primary studies
History-based	Failure History	77%	[42,45–49,52–56,58–64,66,68–75]
	Execution History	54%	[45,46,48,49,52,53,55,58,59,61–64,67–70,72,73]
	Test Age	9%	[46,47,63]
Coverage-based	Test Coverage	14%	[52,53,59,61,67]
	Functions not yet covered	14%	[41,43,44,50,51]
	Functions covered	14%	[41,43,44,50,51]
	Statements covered	14%	[41,43,44,50,51]
	Statements not yet covered	14%	[41,43,44,50,51]
	Branches not yet covered	9%	[44,50,51]
	Branch covered	9%	[44,50,51]
Human-based	Importance	9%	[58,62,68]
Others	Code Changing	9%	[54,58,62]

tion sources. Besides the information used in the prioritization shown in Table 10, we can mention other kinds of information sources used, such as prioritization order history, severity history, correlation data, dissimilarity, text similarity, and text case description. Moreover, most studies use three sources, and the most used kinds of information sources are failure and execution history.

The works in the category `History-based` consider information about which test cases failed previously. We observed that 19 of them (out of 28, 54%) also consider the Test Execution History, that is, the time to execute the test cases.

Studies in the category `Coverage-based` use the total number of covered (or not covered) statements, functions, methods, and features [41,43,44,47,50–53,59,61,67,71,74].

Works in the category `Others` use different kinds of information to prioritize the test cases. We identified 8 works that use Search-based or Machine Learning (ML) techniques [41,43,44,47,48,50–52,58,62,67,74,75]. Besides that, studies in such a category perform the prioritization considering test case dissimilarity (Diversity-based) [42] or test selection strategies [57].

In the category `Cost-aware` [45,49,53,67] the studies assume that each test case does not have the same cost, that is, some may be more costly to execute than others, maybe due to the fault severity or running time. One way to evaluate this assumption is by using the APFDc measure proposed by Elbaum et al. [28]. The test cases are usually prioritized until a maximum cost is reached that is feasible to execute.

Works in the category `Distribution-based` [41,43] prioritize test cases considering the distribution of their execution profiles via test case distances based on the dissimilarity metrics: count metric and the proportional binary metric. Then, test cases are clustered according to their similarities, allowing, in this way, the identification of redundant test cases and isolating clusters that may cause failures.

We identified only 4 studies [45,58,62,68] in the category `Human-based`. The work of Alegroth et al. [68] uses a feature priority ranked by stakeholders, whilst Strandberg et al. [58,62] use a level of priority defined in each test case by the developers. On the other hand, in the study of Marijan et al. [45], a human defines domain-specific heuristics (weights) for the prioritization according to the organization settings. Consequently, we classified this approach as a prioritization based on user preference.

In the category `Probabilistic`, Abdullah et al. [65] propose an idea of a framework to test each Internet of Things (IoT) layer in a separate Test Server (used as a CI environment). The goal is to prioritize tests based on the frequency of the features derived within an operational profile (which characterizes how a system will be used in production). Then, to predict fault location in IoT services, operational profiles (derived from interface behaviors of IoT services) are combined with Markov chain usage models. This kind of test is also known as usage-based testing.

As we can observe from the description above, most studies consider different kinds of information in the test case prioritization and belong to more than one category. This is illustrated in Fig. 6 which presents interactions among the categories and the number of studies associated to each one. Consider the category `History-based` (last row). There are 28 studies belonging to this category (horizontal bar in the left). The

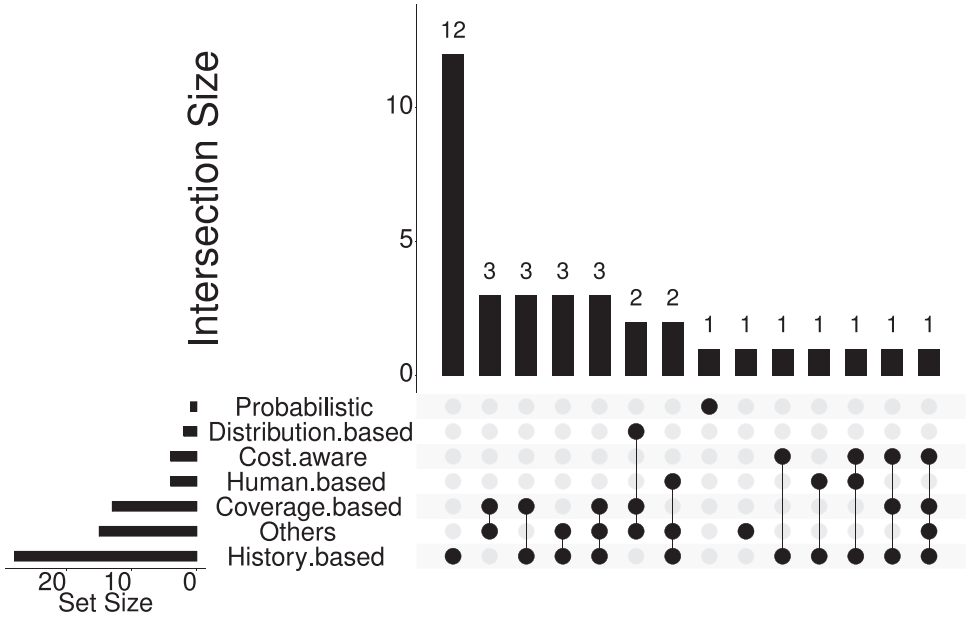


Fig. 6. Intersection between the categories regarding considered information.

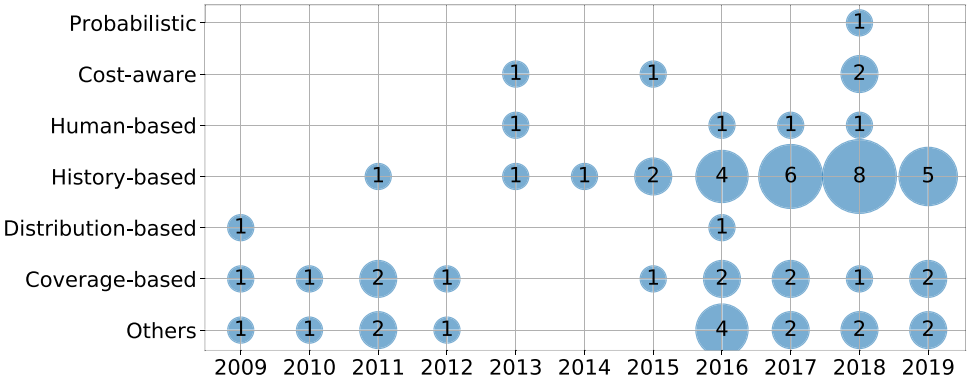


Fig. 7. Categories regarding information used over the years.

intersection with the other categories are represented in the right by bullets, 12 studies do not have any intersection with any other category (first vertical bar), 3 studies of this category also are included in the `Coverage-based` category (third bar), 3 in the category `Others`, and so on.

As mentioned before and evidenced by Fig. 6, we observe a large number of approaches based on historical information about test cases previously executed in CI environments. To obtain other kinds of information can be not possible due to the constraints in the CI environments. As a consequence, there is an effort to create lightweight approaches. Approaches that require exhaustive analysis are costly and inefficient [46], as well as the time available to run the prioritized test suite can be reduced if prioritization takes too long [29].

To better analyze this fact, we can see in Fig. 7 the categories used over the years. `Coverage-based`, `Distribution-based`, and `Others` were the first kind of categories used. The use of historical test data appeared in 2011. Since then, we observe a growing number of studies in this category, mainly in the last two years 2017 and 2018, when 14 studies in the category `History-based` (out of 28, 50%) were published. In this way, we can observe a growing number of studies in this category and a trend in the field.

4.2.2. RQ2.2 - Application contexts

To characterize the application context of the found approaches, we collected information about the programming language, requirements, environments explored, and limitations, as well as some CI particularities and testing problems.

We observed that few studies (11 out of 35, 31%) make explicit the programming language addressed. From them, we found only three programming languages: C in six studies (55%) [41,43,44,50,51,67], Java in five studies (45%) [42,56,60,67,74], and Ruby in one study (9%) [49]. Many studies use systems (datasets) from companies. Consequently, details about the used systems are limited. Furthermore, some datasets are a sample of products including many programming languages, i.e., Google Shared Dataset of Test Suite Results (GSDTSR) [30].

On the other hand, we observed that 31% of the studies [41–44,47,50–52,54,67,74] need code analysis to perform the prioritization. Most of these studies [41,43,44,47,50–52,67,74] belong to the `Coverage-based` category, whilst the other ones analyse code changing [54] and dissimilarity of the test cases [42].

In the `Coverage-based` category only 4 studies do not require code analysis [53,59,61,71]. Three studies [59,61,71] investigate TCPCI in Highly Configurable Systems (HCS) context, and one study evaluates three Android systems [53]. Such studies analyze feature coverage.

In relation to the CI environments investigated, we identified ten industrial environments and two CI frameworks (services). The most common CI environment investigated is from Google [46,48,49,52,63,64,66,73], once that the GSDTSR dataset from Google

Table 11
Continuous Integration problems and particularities in TCPCI field.

Description	Perc.	Primary studies
Continuous Integration testing problems		
User Interface (UI) testing	9%	[68,75]
Time-Consuming testing	78%	[45,48,49,52,53,55,57–59,61,62,64,68–73]
Flaky tests	13%	[46,47,52]
Complex testing	43%	[45,53,55,59,61,65,66,69,71,73]
Continuous Integration particularities		
Parallelism	14%	[46,63,65,66,73]
Volatility	34%	[42,46–49,58,61–63,66,70,74]
Multiple Commits (Test Requests)	11%	[46,49,66,72]

is available online[2] allowing easy use and comparison. This environment is followed by the Cisco environment [45,55,59,61,69,71,73], result of the cooperation between Simula Research Laboratory and Cisco company. ABB Robotics appears in the sequence [48,70,73] and it also has its datasets IOF/ROL and Paint Control available online.[3] The other CI environments are: Ericsson [54,72], Westermo [58,62], Axis Communications [54], Baidu [67], LexisNexis [75], Salesforce.com [47], and Techship [68].

Representing the CI frameworks category we found Jenkins [53] and Travis CI [42,49]. Travis CI is the most popular CI framework in the GitHub, accounting for roughly 50% of the CI market,[4] followed by Circle CI, and Jenkins. The use of CI frameworks can be considered a gap to be explored.

Table 11 presents the CI problems and particularities investigated/addressed by the studies.

We identified that 34% of the studies (12 out of 35) do not explicitly mention some CI testing problem addressed. Most studies (18 out of 26, around 78%) deal with `Time-Consuming testing` (or Time-Limit Treatment), once that this restriction is easier to deal rather than other problems.

43% (10 out of 26) of the studies address the `Complex testing` problem. In such a problem, HCS are the most investigated (7 studies), followed by Android and IoT systems (1 study each). The CI testing problems `Flaky tests` and `UI testing` are few explored, respectively, 13% and 9% of the studies address such problems. To investigate such CI testing problems is a gap for future work.

Regarding the CI particularities, we observed that 14% of the studies consider `Parallelism`, that is a particularity of the CI environment where multiple test machines are used and test cases are executed in parallel. Even with modern large-scale parallel test infrastructures, it can take a relatively long time until an engineer has received complete testing feedback for a given change [47].

Another particularity of the CI environment is the `Volatility` of the test cases, that is, that new test cases can be added or removed (discontinued) during the software life-cycle. Although the studies aim to address TCPCI, we identified few studies (12 out 35, $\approx$ 34%), which explicitly consider this particularity.

Only 11% of the studies consider `Multiple Commits` (or multiple test requests). This particularity is related to the fact that the CI environment can receive multiple requests (commits) to test at the same time, and an order to test them is required. We observed that only two studies consider prioritization of commits [49,66]. To address this particularity is a research opportunity.

We also examined the works considering another particularity about the use of resources in CI environments, such as memory consumption and the use of GPU (Graphics Processing Unit) to allow a fast prioritization process. However, we have not identified any related work.

We can see that many approaches search for code independence, due to the time constraints of a CI environment, as well as independence regarding programming language. However, few studies address CI environment particularities and problems, such as parallel execution of test suites, the impact of time constraints, and volatility of test cases.

4.3. Evaluation aspects

The evaluation phase is especially important to check the usefulness and applicability of the proposed approaches. In this section, we provide answers for RQ3, by analysing how the approaches have been evaluated (RQ3.1, Section 4.3.1) and how the results are measured and whether they are statistically analyzed (RQ3.2, Section 4.3.2).

4.3.1. RQ3.1 - Evaluation of the approaches

It is interesting to observe that almost all the found studies report evaluation results, except two studies [57,65] that are theoretical. A possible reason for this is the fact that prioritization of test cases is related to a practical issue. Then, 33 studies were considered in our analysis to characterize the evaluation contexts and answer our research questions.

We identified 99 systems (or datasets) used in the studies. Most of them were few used, $\approx$ 81% of them were used once, and $\approx$ 9% twice. In this sense, we analyzed the systems used more than twice (10%). Among them, GSDTSR [30] was used seven times. The other systems used more than twice are Video Conferencing Systems from Cisco, Paint Control from ABB Robotics, and the systems that are part of Siemens suites obtained from the Software-artifact Infrastructure Repository (SIR)[5]: tot_info, schedule, print_tokens2, replace, print_tokens, schedule2, and tcas.

A rigorous evaluation methodology should consider the techniques selected for a comparison with the proposed approach. In this sense, we identified 59 techniques commonly used as a baseline. The most common is Random TCP used in 19 studies out of 33 ($\approx$ 54%), followed by the ROCKET technique [45] and the use of untreated tests (that is the use of the same order of the original set), both used in 18% of the studies. We can also highlight the Manual prioritization, used as a baseline in $\approx$ 15% of the studies.

We also analyzed the main threats mentioned by the authors in the studies. 11 studies [41,52,54,57–59,61,62,64,65,70] do not mention any threat. Among the studies that mention threats, the most common one that appeared in 100% of the studies is related to the *Systems* used. This fact points out some concerns about the scalability and generalization of the approaches proposed. After, threats in the categories *Evaluation Measure* (58%), *Techniques Compared* and *Experimental Settings* appear with similar frequency (38% each). Other threats are related to the tool used, randomness aspects, available resources, etc.

To understand how these threats can be related to CI testing problems (discussed in Section 4.2.2) and possibly affect them, we build Fig. 8. In such a figure, we provide an illustration showing the relationship between the dimensions: CI testing problems, CI particularities, and the main limitations (threats) found in the studies.

We can also observe the maturity of the TCPCI field regarding the investigation of CI testing problems and the most involved limitations. For instance, most studies that consider the CI particularity, Volatility of test cases, also seek to address the Time-Consuming testing problem. Still, they have the main limitation concerning the systems used in the experiments. Moreover, the UI testing problem has not been investigated considering any CI particularity, and this should be explored in future work.

[2] The GSDTSR dataset can found at https://code.google.com/archive/p/google-shared-dataset-of-test-suite-results/.

[3] The datasets from ABB Robotics can found at https://bitbucket.org/HelgeS/atcs-data.

[4] Information extracted from blog: GitHub welcomes all CI tools.

[5] Available at http://sir.unl.edu/.

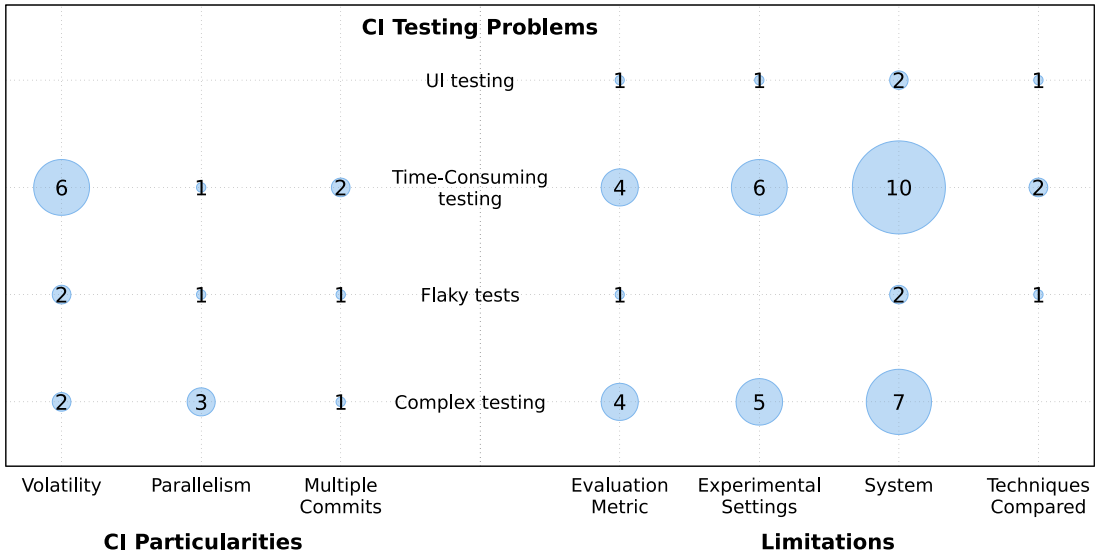


Fig. 8. Relationship of the studies concerning CI particularities, CI testing problems, and the main limitations (threats) found by authors.

The figure shows that there are more limitations than solutions in the TCPCI field. In this sense, there is a lack of studies that consider, at the same time, CI particularities and CI problems.

4.3.2. RQ3.2 - Evaluation measures

Evaluation measures play an essential role in measuring the efficacy of an approach, as well as to benchmark its effectiveness against other existing ones. In this sense, we identified 23 evaluation measures. Among them, 6 are the most common (used), and the other ones were grouped in the category `Others`.

As we can see in Table 6 the most widely used measure is Time, used in 16 out of 33 studies (48%), followed by FD (33%), APFD (24%), APFDc (18%), Expense (15%), and NAPFD (9%). In the studies, the measure Time can encompass: test case execution time, the time the approach takes to perform the prioritization, and the time reduction obtained. In this way, we can use *Time* concerning the execution time of an approach and test case execution time to measure the efficiency of the approach, whilst to measure the effectiveness we can use the time reduction obtained. This last one concerns the time spent to the first failure, which can impact the time spent in a CI cycle and consequently, the test cost.

Average Percentage of Faults Detected (APFD) [31] indicates how quickly a set of prioritized test cases (T') can detect faults present in the application being tested, and its value is calculated from the weighted average of the percentage of detected faults. Higher APFD values indicate that the faults are detected faster using fewer test cases.

Number or percentage of Faults Detected (FD) is a simple version of the APFD measure, which considers only the fault detected rate. This measure can be used to compare the faults detected in the test reduction, for instance, concerning a time constraint or a percentage of test case executed. Besides that, we can find in the literature [71] a formalization of this measure as *Fault-detection effectiveness*. This measure considers the ratio of the number of *nonrepeated* faults detected by the reduced test suite and the number of *nonrepeated* faults detected by its original (nonreduced) test suite. The values are between 0 and 1, higher values indicate better fault-detection effectiveness of a test suite. *Nonrepeated* faults are faults counted only once regardless of the number of test cases that detect that fault. However, to identify *nonrepeated* faults can be a hard task.

Expense [32–34] is a common metric to measure the effectiveness of fault localization, and it is computed by dividing the number of statements needed to be examined to find a specific fault by the total number of executable statements in the program. A technique with a smaller expense to locate a particular fault means better fault localization effectiveness.

APFDc (APFD with cost consideration) [28] and NAPFD (Normalized APFD) [35] [48] are measures adapted from APFD. APFDc was proposed to deal with an APFD limitation concerning the assumption that all faults have equal severity and the test cases have equal costs. These assumptions are not possible in practice, and therefore, APFDc takes into account the fault severity and test cost. Furthermore, if both fault severity and test case costs are identical, APFDc can be used to compute the APFD value. The NAPFD measure is an extension of APFD. To calculate NAPFD, we consider the ratio between detected and detectable faults within T. This measure is adequate to prioritize test cases when not all of them are executed, and faults can be undetected.

Besides the most common evaluation measures described before, we identified 17 measures which were grouped in the category `Others`, such as recall, precision, f-measure, the area under a curve, number of test cases run, the test case/suite size, requirement coverage, and successful fault localization percentage.

Although many studies conducted experiments and analysis of results, few of them (10 out 33 studies $\approx$ 30%) apply a statistical test. Table 12 presents the statistical tests used in the primary studies. Among them, ANOVA is the most preferred test. There is less use concerning the pairwise comparison and effect size measurement. Through this analysis, we observed a lack of statistical tests applied to evaluate the results.

Table 12
Statistical tests applied in TCPCI field.

Statistical tests	Percentage	Primary studies
Pairwise comparison		
Wilcoxon Mann–Whitney	9%	[42,71,73]
Multiple comparison		
ANOVA	18%	[41,43,44,50,51,67]
Dunn's	3%	[73]
Kruskal–Wallis	3%	[73]
Scott–Knott analysis	3%	[75]
Tukey's HSD	6%	[44,67]
Effect size measurement		
Cliff's delta	3%	[75]
Vargha–Delaney $\hat{A}$ measure	6%	[42,73]

5. Trends and research opportunities

During the analysis of the found studies, we identified some trends and research gaps. Based on them, we discuss in this section, the main research opportunities related to each research question. We also present some trends in the field.

5.1. Evolution of the field

By analysing the publications over the years, we can conclude that TCP is an emergent research topic. The great majority of the found studies (21 out of 35, 60%) have been published in the last three years (2016-2018). As a consequence, we observe that there are few research groups working on this subject, and there is space for innovation.

5.2. Type of approaches

As mentioned in Section 4.2.1, we observe a trend and preference by History-based techniques. Coverage-based techniques and search-based ones can take time to perform the prioritization. The fault-based history is the most used, but there are other kinds of information sources to be taken into account, such as execution time, relation with non-functional requirements, and so on. Moreover, we have not found studies addressing Model-based prioritization, considering, for example, behavior models and UML diagrams.

As evidenced by Table 10, most approaches use two and three kinds of information sources. In this path, it is necessary to investigate the impact of the amount of information and identify the most reliable kind of information in the prioritization process. Another research direction observed is exploiting different information sources during the prioritization process. Such a consideration falls in the use of techniques which deal properly with multi-information and dimensionality problem, a gap for future research.

We observed a trend that is to explore Artificial Intelligence (AI) techniques like Machine Learning and probabilistic ones, as well as the use of search-based algorithms with a focus on multi- and many-objectives. According to Spieker et al. [48] and Chen et al. [67], the use of Deep Learning techniques can be a promising path for future research in the TCPCI context. Moreover, search-based techniques can be time-consuming concerning the time spent to find a suitable solution. Besides that, these techniques need to consider the available resources, such as memory consumption. In this sense, we observed that the use of GPU could be considered (not explored in the TCPCI context), as well as the use of Blockchain with Machine Learning[6] which is considered a trend in the AI field.

In the Search-Based Software Engineering (SBSE) field, TCP approaches exploring user preferences to guide the search for the best test case order [36] have been proposed recently. The use of such preferences has been explored in a few studies (only 4 studies in the Human-based category). Another trend in SBSE is the use of Hyper-Heuristics [37] that can provide flexible and adaptable solutions for testing problems. These characteristics can be useful, considering the TCPCI dynamic environment.

Another research opportunity is to explore the use of user requirements and organizational constraints to prioritize the test cases. Furthermore, we identified a lack of studies proposing new techniques for TCP in combination with fault localization. This is a gap to be explored by future research that should consider the CI environment characteristics and challenges. According to Jiang et al. [43], a gap is to study how to achieve tighter integration between regression testing and debugging techniques. Additionally, Yu et al. [75] mention that the use of fault location might help to address the test failure classification problem. In such a problem, each test failure can be related to a specific fault, caused by some piece of code.

[6] Microsoft Research Blog: Leveraging blockchain to make machine learning models more accessible.

5.3. Application contexts

We identified that only three programming languages were addressed by the studies: Java, C, and Ruby. The use of TCPCI for other popular programming languages, such as C# and Python, needs to be explored [38].

We also identified only ten industrial environments and only two CI Frameworks: Travis CI and Jenkins. Considering other industrial scenarios with different kinds of systems is a gap. From such environments, only two environments made available their data used in their research: Google with the GSDTSR dataset and ABB Robotics with its datasets IOF/ROL and Paint Control. Moreover, only two studies made available the proposed approach [48,67]. This hampers the replication of the experiments and undermines a more open science. We suggest future researches to use Open Science Framework to increase the openness, integrity, accessibility, and reproducibility of scientific research, consequently contributing to the Open Science community.

We observed that among the CI testing problems, UI testing and Flaky tests have been few addressed. The most addressed problem is related to time constraints. Concerning complex testing, a gap is to investigate TCPCI techniques in emergent, new contexts, and systems that are hard to test, i.e., UI systems, Service-based systems, dynamic applications, apps phones, and HCS.

We also found few works addressing some CI particularities such as parallelism in the environment, the volatility of test cases, and multiple commits. Future works should focus on CI specific characteristics, such as the use of the available resources, for instance, memory consuming.

5.4. Evaluation of the approaches

Almost all studies (33 out of 35) are dedicated to compare and evaluate the proposed approaches. Recent studies are concerned with the approach effectiveness evaluation by using measures such as Time required to perform the prioritization, given the CI time constraints, but FD is also a measure used by a great number of studies. A gap is the use of NAPFD measure, few explored.

Few studies use a statistical test and are worried about scalability. In this way, more rigorous experiments are necessary to evaluate these aspects and perform a comparison of the approaches in practice. We identified a list of 99 systems used in TCPCI context (see Section 4.3.1, RQ3.1). However, a lack of a benchmark for the field is a gap to be addressed.

GSDTSR is the most used system which contains test suite results from a sample of Google, and it could be used in future studies. As mentioned before, Travis CI is the most popular CI framework in the GitHub, as well as it provides an API to access test results from open-source repositories. A data mining procedure could be conducted to identify different systems, such as HCS and Android. For this, the use of Travis Torrent could be considered [39]. A possible research opportunity is the construction of a repository containing a set of systems that are considered deemed TCP cases and have different characteristics regarding the number of faults by commit and the number of test cases. This could help to overcome the main threats found in the evaluations and reported in the studies.

There is a lack of studies addressing CI particularities and testing problems at the same time. This hampers the evolution of the TCPCI field, mainly due to the high number of limitations identified. The creation of a benchmark, the use of adequate evaluation measures, and the availability of techniques can accelerate research in the TCPCI context. Moreover, a qualitative analysis of the impact of the techniques in industrial settings might be considered as future research.

6. Threats to validity

In this section, we identify possible threats to the validity of our results, according to the taxonomy of Wohlin et al. [40].

Regarding construct validity, the research questions may not address all TCPCI aspects. This threat was mitigated through discussions, and we believe that the questions reflect the goals of our work. Other authors can elaborate other questions and obtain a different analysis. The databases used are well-known and related to the software engineering and software testing areas. In this way, we believe the found studies represent well the TCPCI field. Our search string does not have many elements, but we carefully created a search string capable of finding consistent results. To construct the search string, we selected terms related to our goals and synonymous used in mappings of the TCP field [14,16]. We refined our search string several times by using a control group, reducing the risk that relevant literature is omitted.

Concerning internal validity, maybe we may have extracted and misinterpreted some information. To minimize such a threat during the data extraction, we followed a rigorous plan, using the PRISMA statement, and well-defined inclusion and exclusion criteria. We had meetings and discussions to clarify any doubt arisen during the process.

Other possible threats, related to the conclusion validity, is the granularity of the information described in the studies, which may affect our conclusions. Besides, our schema can also be a threat, as well as the form we classified the papers. To mitigate them, we first documented all relevant information from the primary studies guided by the dimensions associated with the research questions and defined the categories interactively. However, other researchers may obtain another scheme and ways to group and analyze the papers.

Regarding reliability validity, our study can be easily replicated, following the steps described and using the search string or using the raw data analyzed and disseminated by the Open Science Framework (OSF).

7. Concluding remarks

In this paper, we present the results of a systematic mapping study on the TCPCI field. We investigated some aspects of the found studies: the main research goal, characteristics of the explored approach, used evaluation measures, and how the evaluation has been conducted. Furthermore, we also analyzed the main publication fora and how the field has been evolved over the years, trends, and research opportunities to guide future research.

The map found 35 papers, published in a wide range of venues, without the indication of a preferred publication vehicle. We observed an increasing number of studies in the last years and a crescent interest in the field.

The main research goal of the studies is the comparison of prioritization techniques followed by the introduction of a prioritization technique. Concerning the information considered in the prioritization, History-based approaches seem to be a trend. Probabilistic and Distribution-based approaches were explored by only one study. To explore new sources of information to prioritize the test cases such as requirements, organizational restrictions, and human aspects can be an alternative, maybe in combination with historical information. We identified few studies combining TCP and fault localization, and there is no approach based on requirements or models. These are gaps that can guide future research.

To evaluate the proposed approaches, several systems are used. We identified 99 different systems. GSDTSR is the most common system used. The main threat found in the evaluations is concerned with the systems used, followed by the evaluation measures and techniques used in the comparison. We observed that only 9 studies apply a statistical test. Regarding evaluation measures, we identified 23 measures in the primary studies, and Time and FD are the most used.

Some research opportunities are the application of TCPCI in other contexts considering languages such as C# and Python. It is necessary to address other CI testing problems and particularities such as complex and time-consuming testing, flaky tests, UI testing, parallelism, test case volatility, and multiple commits. Other limitations to be overcome are the use of different systems with different characteristics (size, faults by commit, number of commits, number of test cases) to allow generalization and scalability evaluation, as well as the construction of benchmarks.

Declaration of Competing Interest

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

Acknowledgment

This work is supported by the Brazilian agencies CAPES and CNPq. (Grant: 305968/2018-1).

References

[1] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, B. Vasilescu, The impact of continuous integration on other software development practices: a large-scale empirical study, in: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, in: ASE, 2017, pp. 60–71, doi:10.1109/ASE.2017.8115619.

[2] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a Survey, Softw. Test. Verif. Reliab. 22 (2) (2012) 67–120, doi:10.1002/stv.430.

[3] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, in: EASE'08, BCS Learning & Development Ltd., 2008, pp. 68–77.

[4] G. Rothermel, R.J. Untch, C. Chu, Prioritizing test cases for regression testing, IEEE Trans. Softw. Eng. 27 (10) (2001) 929–948, doi:10.1109/32.962562.

[5] 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, doi:10.1109/ACCESS.2017.2685629.

[6] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M.V. Mäntylä, T. Männistö, The highways and country roads to continuous deployment, IEEE Softw. 32 (2) (2015) 64–72, doi:10.1109/MS.2015.50.

[7] Buildbot, The Continuous Integration Framework, (https://buildbot.net), Accessed: 2018-01-22.

[8] GoCD, Open Source Continuous Delivery and Release Automation Server, (https://www.gocd.org), Accessed: 2018-01-22.

[9] Integrity, Continuous Integration Server, (https://integrity.github.io), Accessed: 2018-01-22.

[10] Jenkins, (https://wiki.jenkins-ci.org/display/JENKINS/Home), Accessed: 2018-01-22.

[11] Travis CI, Travis CI, (https://travis-ci.org), Accessed: 2018-01-22.

[12] T. Karvonen, W. Behutiye, M. Oivo, P. Kuvaja, Systematic literature review on the impacts of agile release engineering practices, Inf. Softw. Technol. 86 (2017) 87–100, doi:10.1016/j.infsof.2017.01.009.

[13] Y. Singh, A. Kaur, B. Suri, S. Singhal, Systematic literature review on regression test prioritization techniques, Informatica 36 (4) (2012) 379–408.

[14] C. Catal, D. Mishra, Test case prioritization: a systematic mapping study, Softw. Qual. J. 21 (3) (2013) 445–478, doi:10.1007/s11219-012-9181-z.

[15] A. Kumar, K. Singh, A literature survey on test case prioritization, Compusoft 3 (5) (2014) 793.

[16] M. Khatibsyarbini, M.A. Isa, D.N. Jawawi, R. Tumeng, Test case prioritization approaches in regression testing: a systematic literature review, Inf. Softw. Technol. 93 (2018) 74–93, doi:10.1016/j.infsof.2017.08.014.

[17] T. D. Hellmann, A. Sharma, J. Ferreira, F. Maurer, Agile Testing: Past, Present, and Future–Charting a Systematic Map of Testing in Agile Software Development, in: Proceedings of the Agile Conference, 2012, pp. 55–63, doi:10.1109/Agile.2012.8.

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

[19] A. Eck, F. Uebernickel, W. Brenner, Fit for continuous integration: how organizations assimilate an agile practice, in: Proceedings of the 20th Americas Conference on Information Systems, in: AMCIS, Association for Information Science, 2014.

[20] D. Ståhl, J. Bosch, Modeling continuous integration practice differences in industry software development, J. Syst. Softw. 87 (2014) 48–59, doi:10.1016/j.jss.2013.08.032.

[21] D. Moher, A. Liberati, J. Tetzlaff, D.G. Altman, Preferred reporting items for systematic reviews and meta-analyses: the PRISMA statement, BMJ 339 (2009), doi:10.1136/bmj.b2535.

[22] V. Garousi, M. Felderer, M.V. Mäntylä, The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, in: EASE, ACM, 2016, pp. 26:1–26:6, doi:10.1145/2915970.2916008.

[23] V. Garousi, M. Felderer, M.V. Mäntylä, Guidelines for including grey literature and conducting multivocal literature reviews in software engineering, Inf. Softw. Technol. 106 (2019) 101–121, doi:10.1016/j.infsof.2018.09.006.

[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, in: EASE'14, ACM, 2014, pp. 38:1–38:10, doi:10.1145/2601248.2601268.

[25] J.A. Prado Lima, S.R. Vergilio, Supplementary Material - Test Case Prioritization in Continuous Integration Environments: a Mapping Study, 2020, 10.17605/OSF.IO/ZFE64.

[26] 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, doi:10.1016/j.infsof.2016.10.001.

[27] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, H. Pfister, UpSet: visualization of intersecting sets, IEEE Trans. Visualiz. Comput.Graph. (InfoVis'14) 20 (12) (2014) 1983–1992.

[28] S. Elbaum, A. Malishevsky, G. Rothermel, Incorporating varying test costs and fault severities into test case prioritization, in: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, 2001, pp. 329–338, doi:10.1109/ICSE.2001.919106.

[29] C. Henard, M. Papadakis, M. Harman, Y. Jia, Y. Le Traon, Comparing white-box and black-box test prioritization, in: Proceedings of the 38th International Conference on Software Engineering, in: ICSE'16, ACM, 2016, pp. 523–534, doi:10.1145/2884781.2884791.

[30] S. Elbaum, A. McLaughlin, J. Penix, The Google Dataset of Testing Results, 2014.

[31] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Test case prioritization: an empirical study, in: Proceedings of the IEEE International Conference on Software Maintenance, in: ICSM'99, IEEE Computer Society, 1999, pp. 179–188, doi:10.1109/ICSM.1999.792604.

[32] D. Jeffrey, N. Gupta, R. Gupta, Fault localization using value replacement, in: Proceedings of the 2008 International Symposium on Software Testing and Analysis, in: ISSTA'08, ACM, 2008, pp. 167–178, doi:10.1145/1390630.1390652.

[33] J.A. Jones, M.J. Harrold, Empirical evaluation of the tarantula automatic fault-localization technique, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, in: ASE'05, ACM, 2005, pp. 273–282, doi:10.1145/1101908.1101949.

[34] Y. Yu, J.A. Jones, M.J. Harrold, An empirical study of the effects of test-suite reduction on fault localization, in: Proceedings of the 30th International Conference on Software Engineering, in: ICSE'08, ACM, 2008, pp. 201–210, doi:10.1145/1368088.1368116.

[35] X. Qu, M.B. Cohen, K.M. Woolf, Combinatorial interaction regression testing: a study of test case generation and prioritization, in: IEEE International Conference on Software Maintenance, 2007, pp. 255–264, doi:10.1109/ICSM.2007.4362638.

[36] T.N. Ferreira, S.R. Vergilio, J.T. de Souza, Incorporating user preferences in search-based software engineering: a systematic mapping study, Inf. Softw. Technol. 90 (2017) 55–69, doi:10.1016/j.infsof.2017.05.003.

[37] T.N. Ferreira, J.A.P. Lima, A. Strickler, J.N. Kuk, S.R. Vergilio, A. Pozo, Hyper-heuristic based product selection for software product line testing, IEEE Comput. Intell. Mag. 12 (2) (2017) 34–45, doi:10.1109/MCI.2017.2670461.

[38] The RedMonk programming language rankings: January 2018s, (https://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/), Accessed: 2018-03-20.

[39] M. Beller, G. Gousios, A. Zaidman, TravisTorrent: Synthesizing Travis CI and GitHub for full-stack research on continuous integration, in: Proceedings of the 14th Working Conference on Mining Software Repositories, 2017.

[40] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: an Introduction, Kluwer Academic Publishers, 2000.

[41] B. Jiang, W.K. Chan, Testing and debugging in continuous integration with budget quotas on test executions, in: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, in: QRS, IEEE, 2016, pp. 439–447, doi:10.1109/QRS.2016.66.

[42] A. Haghighatkhah, M. Mäntylä, M. Oivo, P. Kuvaja, Test prioritization in continuous integration environments, J. Syst. Softw. 146 (2018) 80–98, doi:10.1016/j.jss.2018.08.061.

[43] B. Jiang, Z. Zhang, T.H. Tse, T.Y. Chen, How well do test case prioritization techniques support statistical fault localization, in: Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, in: COMPSAC, 1, IEEE, 2009, pp. 99–106, doi:10.1109/COMPSAC.2009.23.

[44] B. Jiang, Z. Zhang, W.K. Chan, T.H. Tse, T.Y. Chen, How well does test case prioritization integrate with statistical fault localization? Inf. Softw. Technol. 54 (7) (2012) 739–758, doi:10.1016/j.infsof.2012.01.006.

[45] D. Marijan, A. Gotlieb, S. Sen, Test case prioritization for continuous regression testing: an industrial case study, in: Proceedings of the IEEE International Conference on Software Maintenance, in: ICMS, IEEE, 2013, pp. 540–543, doi:10.1109/ICSM.2013.91.

[46] S. Elbaum, G. Rothermel, J. Penix, Techniques for improving regression testing in continuous integration development environments, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, in: FSE, ACM, New York, NY, USA, 2014, pp. 235–245, doi:10.1145/2635868.2635910.

[47] B. Busjaeger, T. Xie, Learning for test prioritization: an industrial case study, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, in: FSE, ACM, 2016, pp. 975–980, doi:10.1145/2950290.2983954.

[48] H. Spieker, A. Gotlieb, D. Marijan, M. Mossige, Reinforcement learning for automatic test case prioritization and selection in continuous integration, in: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, in: ISSTA, ACM, New York, NY, USA, 2017, pp. 12–22, doi:10.1145/3092703.3092709.

[49] J. Liang, S. Elbaum, G. Rothermel, Redefining prioritization: continuous prioritization for continuous integration, in: Proceedings of the 40th International Conference on Software Engineering, in: ICSE, ACM, New York, NY, USA, 2018, pp. 688–698, doi:10.1145/3180155.3180213.

[50] B. Jiang, W.K. Chan, On the integration of test adequacy, test case prioritization, and statistical fault localization, in: Proceedings of the 10th International Conference on Quality Software, in: QSIC, IEEE, 2010, pp. 377–384, doi:10.1109/QSIC.2010.64.

[51] B. Jiang, W.K. Chan, T.H. Tse, On practical adequate test suites for integrated test case prioritization and fault localization, in: Proceedings of the 11th International Conference on Quality Software, in: QSIC, IEEE, 2011, pp. 21–30, doi:10.1109/QSIC.2011.37.

[52] S. Yoo, R. Nilsson, M. Harman, Faster fault finding at Google using multi objective regression test optimisation, in: Proceedings of the 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, in: ESEC/FSE'11, 2011. Industry Track

[53] D. Marijan, Multi-perspective regression test prioritization for time-constrained environments, in: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, in: QRS, IEEE Computer Society, 2015, pp. 157–162, doi:10.1109/QRS.2015.31.

[54] E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, M. Castell, Supporting continuous integration by code-churn based test selection, in: Proceedings of the IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering, in: RCoSE, IEEE, 2015, pp. 19–25, doi:10.1109/RCoSE.2015.11.

[55] D. Marijan, M. Liaaen, Effect of time window on the performance of continuous regression testing, in: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, in: ICSME, IEEE, 2016, pp. 568–571, doi:10.1109/ICSME.2016.77.

[56] Y. Cho, J. Kim, E. Lee, History-based test case prioritization for failure information, in: Proceedings of the 23rd Asia-Pacific Software Engineering Conference, in: APSEC, IEEE, 2016, pp. 385–388, doi:10.1109/APSEC.2016.066.

[57] M. Eyl, C. Reichmann, K. Müller-Glaser, Fast feedback from automated tests executed with the product build, in: Proceedings of the 8th Internation Conference on Software Quality, in: SWQD, Springer, 2016, pp. 199–210, doi:10.1007/978-3-319-27033-3_14.

[58] P.E. Strandberg, D. Sundmark, W. Afzal, T.J. Ostrand, E.J. Weyuker, Experience report: automated system level regression test prioritization using multiple factors, in: Proceedings of the 27th International Symposium on Software Reliability Engineering, in: ISSRE, IEEE, 2016, pp. 12–23, doi:10.1109/ISSRE.2016.23.

[59] D. Marijan, M. Liaaen, A. Gotlieb, S. Sen, C. Ieva, TITAN: test suite optimization for highly configurable software, in: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation, in: ICST, IEEE, 2017, pp. 524–531, doi:10.1109/ICST.2017.60.

[60] J. Kim, H. Jeong, E. Lee, Failure history data-based test case prioritization for effective regression test, in: Proceedings of the Symposium on Applied Computing, in: SAC, ACM, New York, NY, USA, 2017, pp. 1409–1415, doi:10.1145/3019612.3019831.

[61] D. Marijan, M. Liaaen, Test prioritization with optimally balanced configuration coverage, in: Proceedings of the IEEE 18th International Symposium on High Assurance Systems Engineering, in: HASE, IEEE, 2017, pp. 100–103, doi:10.1109/HASE.2017.26.

[62] P.E. Strandberg, W. Afzal, T.J. Ostrand, E.J. Weyuker, D. Sundmark, Automated system-level regression test prioritization in a nutshell, IEEE Softw. 34 (4) (2017) 30–37, doi:10.1109/MS.2017.92.

[63] J.-H. Kwon, I.-Y. Ko, Cost-effective regression testing using bloom filters in continuous integration development environments, in: Proceedings of the 24th Asia-Pacific Software Engineering Conference, in: APSEC, IEEE, 2017, pp. 160–168, doi:10.1109/APSEC.2017.22.

[64] L. Xiao, H. Miao, Y. Zhong, Test case prioritization and selection technique in continuous integration development environments: a case study, Int. J. Eng. Technol. 7 (2.28) (2018) 332–336, doi:10.14419/ijet.v7i2.28.13207.

[65] A. Abdullah, H.W. Schmidt, M. Spichkova, H. Liu, Monitoring informed testing for IoT, in: Proceedings of the 25th Australasian Software Engineering Conference, in: ASWEC, IEEE, 2018, pp. 91–95, doi:10.1109/ASWEC.2018.00020.

[66] Y. Zhu, E. Shihab, P.C. Rigby, Test re-prioritization in continuous testing environments, in: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, in: ICSME, IEEE, 2018, pp. 69–79, doi:10.1109/ICSME.2018.00016.

[67] J. Chen, Y. Lou, L. Zhang, J. Zhou, X. Wang, D. Hao, L. Zhang, Optimizing test prioritization via test distribution analysis, in: Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, in: ESEC/FSE, ACM, 2018, pp. 656–667, doi:10.1145/3236024.3236053.

[68] E. Alégroth, A. Karlsson, A. Radway, Continuous integration and visual gui testing: Benefits and drawbacks in industrial practice, in: Proceedings of the 11th International Conference on Software Testing, Verification and Validation, in: ICST, IEEE, 2018, pp. 172–181, doi:10.1109/ICST.2018.00026.

[69] D. Marijan, M. Liaaen, S. Sen, DevOps improvements for reduced cycle times with integrated test optimizations for continuous integration, in: Proceedings of the IEEE 42nd Annual Computer Software and Applications Conference, in: COMPSAC, 01, IEEE, 2018, pp. 22–27, doi:10.1109/COMPSAC.2018.00012.

[70] W. Wen, Z. Yuan, Y. Yuan, Improving RETECS method using FP-Growth in continuous integration, in: Proceedings of the 5th IEEE International Conference on Cloud Computing and Intelligence Systems, in: CCIS, IEEE, 2018, pp. 636–639, doi:10.1109/CCIS.2018.8691385.

[71] D. Marijan, A. Gotlieb, M. Liaaen, A learning algorithm for optimizing continuous integration development and testing practice, Software 49 (2) (2019) 192–213, doi:10.1002/spe.2661.

[72] A. Najafi, W. Shang, P.C. Rigby, Improving test effectiveness using test executions history: an industrial experience report, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, in: ICSE-SEIP, IEEE Press, Piscataway, NJ, USA, 2019, pp. 213–222, doi:10.1109/ICSE–SEIP.2019.00031.

[73] D. Pradhan, S. Wang, S. Ali, T. Yue, M. Liaaen, Employing rule mining and multi-objective search for dynamic test case prioritization, J. Syst. Softw. 153 (2019) 86–104, doi:10.1016/j.jss.2019.03.064.

[74] S. Ali, Y. Hafeez, S. Hussain, S. Yang, Enhanced regression testing technique for agile software development and continuous integration strategies, Softw. Qual. J. (2019), doi:10.1007/s11219-019-09463-4.

[75] Z. Yu, F. Fahid, T. Menzies, G. Rothermel, K. Patrick, S. Cherian, TERMINATOR: better automated UI test case prioritization, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, in: FSE, ACM, 2019, pp. 883–894, doi:10.1145/3338906.3340448.