Performance regression testing initiatives: a systematic mapping

Luciana Brasil Rebelo dos Santos[a,b], Érica Ferreira de Souza[c], André Takeshi Endo[d], Catia Trubiani[a], Riccardo Pinciroli[a], Nandamudi Lankalapalli Vijaykumar[e]

[a] *Gran Sasso Science Institute (GSSI), L'Aquila, Italy*
[b] *Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), Jacareí, Brazil*
[c] *Universidade Tecnológica Federal do Paraná (UTFPR), Cornélio Procópio, Brazil*
[d] *Universidade Federal de São Carlos (UFSCar), São Carlos, Brazil*
[e] *Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, Brazil*

Abstract

Context. Issues related to the performance of software systems are crucial, as they have the potential to impede the effective utilization of products, compromise user satisfaction, escalate costs, and lead to failures. Performance regression testing has been identified as a prominent research domain, since it aims to prevent anomalies and substantial slowdowns.
Objective. The objective of this paper is to examine recent approaches proposed in the literature concerning performance regression testing. Our interest lies in contributing insights that offer a forward-looking perspective on what is essential in this promising research domain.
Method. We carried out a systematic mapping study with the objective of gathering information on various initiatives related to performance regression testing. Our methodology follows the state-of-the-art guidelines for systematic mappings comprising planning, conducting, and reporting activities, thus obtaining a comprehensive set of selected studies.

[1]luciana.rebelo@gssi.it
[2]ericasouza@utfpr.edu.br
[3]andreendo@ufscar.br
[4]catia.trubiani@gssi.it
[5]riccardo.pinciroli@gssi.it
[6]vijay.nl@inpe.br

Results. Our selection includes 68 papers, and our analysis focuses on four key research questions, delving into (i) publication trends, (ii) developed approaches, (iii) conducted evaluations, and (iv) challenges. As a result of this investigation, we present a roadmap highlighting research opportunities. *Conclusion.* This flourishing research field entails a broad set of challenges, such as deciding the granularity of tests and the frequency of launching the performance regression process. Consequently, there is still much work to be undertaken to trade-off between the accuracy and the efficiency of capturing complex performance issues across diverse application domains and/or execution environments.

Keywords: Performance regression, Software testing, Systematic mapping

---

1. Introduction

The monitoring and verification of performance requirements have been recently assessed as key activities in the software development process. In fact there exists an emerging trend in research considering the system performance characteristics as fundamental to guarantee the system's correctness [1]. Detecting and fixing performance issues is indeed valuable to prevent critical consequences, such as customers' dissatisfaction, and abandonment of software projects with large economic losses [2].

Performance regression testing represents the spotlight activity to evaluate whether a new build of a software system has some sort of performance deterioration when compared to its previous version. In this respect, software testing plays the major role of validating the new version and making the necessary adjustments to prevent it from degrading after the addition of new functionalities w.r.t. its previous build [3, 4, 5].

The high-level objective of this study is to support researchers and practitioners in gaining comprehensive and in-depth understanding of how the performance regression testing is recently evolving in the literature. More specifically we intend to focus on four Research Questions (RQ). In particular: $RQ1$ is meant to extract the venues that attract this type of research contribution, along with analytical data across years of publications; $RQ2$ is aimed to point out the main approaches characteristics, together with most common application domains; $RQ3$ focuses on investigating the evaluation of the selected approaches, e.g., most frequent subject systems, type of conducted experimentation, etc.; and $RQ4$ collects the challenges highlighted in the literature with the goal of deriving which directions have been explored so far, and which research opportunities are still valid.

We address these research questions through a systematic mapping study, with the goal of providing an overview of the performance regression testing field [6]. It is worth remarking that systematic mapping studies build upon the trade-off between effort and the reliability of the outcome, as outlined in [6]. However, we think that our effort is compensated by the selection of such a field of investigation, given that industrial perspectives recently recognized the importance of handling performance issues as the new system correctness [1, 7]. In particular, we identified 68 studies addressing software testing with performance regressions. The selected studies were retrieved, reviewed, and analyzed following the well-established systematic mapping study process [8, 9]. In summary, the key contributions of this study include:

- A comprehensive overview of the current state of research that integrates software testing and performance regression. This includes a characterization of the developed techniques, and their evaluation;
- A reflection on the evaluation methodologies from the literature, in terms of the characteristics of the subject systems, as well as the evaluation aspects, such as the most common programming languages, the granularity and the frequency of testing, etc.;
- A summary of findings and limitations extracted from the prior work, which triggers future research opportunities.

The remainder of this manuscript is structured as follows. Section 2 presents the background knowledge, briefly explaining software testing and performance regression testing; it includes Section 2.4 that discusses the related work. Sections 3 and 4 argue on the research method and the selection process applied to perform the mapping. Data extraction and synthesis are explained in Section 5, whereas the analysis of the obtained results according to the research questions is reported in Section 6. Research opportunities, along with their implications and challenges are discussed in Section 7. Section 8 discusses the threats to validity. Conclusion and future research are outlined in Section 9. Supplementary material to review the selected studies and enable future replications can be found at the following link: *https://doi.org/10.5281/zenodo.14236847*.

2. Background

In this section, the main concepts of this study are discussed.

2.1. Software testing

Software testing activities are crucial for ensuring the quality of developed software systems. Verification and Validation (V&V) play a significant role in this regard, as highlighted by Mathur [10]. V&V activities encompass both static elements, such as technical reviews and inspections, and dynamic elements, which involve testing. The primary goal of V&V is to ensure that the software model and the implemented product adhere to the specified requirements [11].

Conducting exhaustive testing on software may not always be practical, particularly when dealing with a large input domain. In such cases, it becomes essential to employ techniques that focus on selecting subsets of the input domain while still addressing the coverage aspect. These techniques can be categorized as either "Black-box Techniques", where test cases are generated based on the input/output behavior without delving into the code, or "White-box Techniques", which require the use of code to generate test cases and assess their outcomes [10].

2.2. Performance regression

A software regression is identified as a bug, typically denoting a lapse in functionality or behaviour compared to a previous version. Essentially, any alteration in a version, a function, or a bug fix may inadvertently introduce new issues. The purpose of regression testing is to ensure that software modifications do not affect features of the software that should not change [12]. While a new version might appear functionally sound, it could suffer from degraded performance, such as increased resource utilization or other efficiency drawbacks. Performance glitches may also be introduced by changes to the running environment, such as system upgrades, or even changes due to external conditions, e.g., to adapt to daylight saving time [13].

Software quality, encompassing various facets, hinges on addressing performance issues to prevent failures in software systems. Performance regression, highlighted by Chen et al. [14], stands out as a crucial concern. Examples cited by the authors include deteriorating response times and heightened resource usage. Although not all performance issues equate to bugs, they can significantly impact users, leading to delays in responding to requests. These issues can result in reputation damage for companies and increased costs [3]. Notably, the detection of performance regression only occurs after the software is developed and in use. Consequently, it becomes imperative to explore available options for mitigating performance regression, with one viable solution being the incorporation of Performance Regression Testing in the Software Test Plan.

2.3. Performance regression testing

Performance regression testing involves comparing the performance of different versions of software systems, with the goal of identifying differences that may indicate performance regressions. Software systems are continually modified, and their upkeep is facilitated by regression testing, despite its inherent expense. The primary objective is to validate modified software systems, ensuring that changes in new versions do not degrade or negatively impact software behavior, as emphasized by [15].

In contrast to unit testing, performance tests come with specific constraints, requiring the utilization of numerous resources, extended run times, and realistic scenarios for effective functioning. While it would be ideal to conduct tests for every version, the practicality of such an approach diminishes due to excessive costs and time constraints [16].

Therefore, it becomes imperative to define this type of testing in the test plan. Even after the system is ready and deployed with the client, ongoing changes are often necessary to accommodate new functionalities or correct errors reported by the client. Care must be taken to avoid deterioration in the new version compared to the previous one. Performance testing falls under the category of nonfunctional testing, focusing on verifying speed, stability, reliability, and scalability of the software [17]. Neglecting this type of testing may have adverse effects on operational profiles and should thus be diligently conducted.

2.4. Related work

In this paper, we present a systematic mapping, i.e., a secondary study that is based on analyzing a collection of research papers (primary studies) with the intention of characterizing a field of research [8]. Our mapping identifies and classifies all research relating to performance regression testing. Hence, before accomplishing the secondary study presented in this paper, we performed a tertiary study looking for other secondary studies investigating the same research topic. Tertiary studies are considered as a review that focuses only on other secondary studies (Systematic Literature Reviews or Systematic Mappings) [8].

To conduct the tertiary study, we used the search string presented in Table 1. For the construction of this string, we adopted a suitable approach to retrieve secondary studies, as proposed in Napoleão et al. [18]. We used Scopus and Google Scholar as search engines. Scopus shows the possibility to select metadata fields, and we considered title, abstract and keywords as target fields of interest. Searching for the given string does not return any study. Google Scholar does not report metadata fields, the search string is executed considering the entire document, and a considerable amount of studies can be returned. Our search string approximately produced 2890 results on Google Scholar We decided to scan the first result pages until saturation, as discussed in Garousi et al. [19]. We stopped our extraction when no relevant study emerged from the results, i.e., 50 papers have been selected for a further round of analysis. To avoid some personal search bias, we performed our search in an incognito mode.

Table 1: Tertiary study - Areas and Keywords

Areas	**Keywords**
Software	“Software” OR “System*”
Performance Regression	“Performance Regression*”
Software Testing	“Test*” OR “suite”
Secondary Study	“systematic review” OR “literature review” OR “systematic mapping” OR “mapping study” OR “systematic map”
Search String: (“Software” OR “System*”) AND (“Performance Regression*”) AND (“Test*” OR “suite”) AND (“systematic review” OR “literature review” OR “systematic mapping” OR “mapping study” OR “systematic map”)	

When analyzing the 50 studies returned by Google Scholar, none of them corresponded to a secondary study directly related to performance regression testing. However, two studies drew our attention since they address regression testing and software performance, respectively. A brief description of these studies is presented below.

Arora and Bhatia [20] conducted a systematic literature review about existing test case generation approaches for regression testing and agent-based software testing systems. The authors identified 115 studies as potential research on the investigated topic, of which 59 studies are on regression test case generation, and 56 are on agent-based software testing. The study shows a quantitative review of existing regression test case generation and agent-based testing studies, in order to identify existing frameworks, techniques, methods and platforms. Some results presented by the study are: the use of dominant agents in Web testing and object-oriented testing; most of the studies recommend the use of agents to reduce the testing time and effort; the reliability and scalability of most of the agent-based software testing techniques are still an issue; there is an increase in use of agents in test case generation using structural-based and model-based approaches; there is a need to develop more sophisticated techniques with orientation toward industry requirements. Considering industrial applications, none of the studies have used real-world cases as subject systems; in relation to platforms used, JADE is the favorite platform for the development of agent-based software testing systems; and mobile agent-based regression testing is still in its infancy, that is, there are still open areas for research.

Han et al. [21] performed a systematic mapping to provide an overview of the latest research literature available in software performance (from 2011 to 2020). The study reviewed 222 primary studies, focusing on identifying the types and characteristics of software performance research that have been conducted. Some findings are: 25% of papers are about performance issue detection; selected studies are heavily concentrated on the validation research techniques type (85%); most studies are concentrating on testing and maintenance and most techniques investigated (87%) are meant for developers; most studies use database servers, web servers, and APIs as study subjects. Regarding databases, about 92% of the primary studies choose MySQL (or its variations such as the MariaDB server); and only about 48% of those studies design experiments to evaluate the methods' effectiveness.

In addition to the two aforementioned secondary studies, our systematic mapping identified the related work of Kazmi et al. [22]; this study was not included in our mapping as it does not meet the selection criteria. In fact, Kazmi et al. [22] investigate regression testing, but do not address performance regression testing. The authors examined 47 primary studies with the objective of identifying the effective regression test case selection techniques focusing on cost, coverage, or fault based effectiveness. Some main results of the study are: among the regression test case selection techniques, the most mentioned are Mining and Learning, Model-Based Testing (MBT), Program Slicing and Control Flow Graph (CFG); unit testing was the most used test level in the selected studies; Java is the most used and accepted environment and object-oriented paradigm is more popular than structured solutions; most studies employed case studies with small-sized datasets; the use of industrial artifacts are still uncommon in the experiments; the most common cost measure was execution time of the test suite; the studies also consider in the test suite evaluation comparing their results with previous versions of the same test suite execution.

Besides the studies discussed in this section, it is possible to find further secondary studies focused on regression testing in general [23, 24, 25, 26]. However, based on the results of our investigation, we were unable to identify secondary studies whose main scope was performance regression testing. This motivated us to conduct this study.

3. Research method

The systematic mapping conducted in this study was based on the guidelines given in [8, 9]. This research method involves three phases: (i) **Planning**: refers to identifying a need for conducting the review, and aims at establishing a review protocol. The protocol defines the research questions, inclusion and exclusion criteria, sources of studies, and search string; (ii) **Conducting**: searches and selects the studies, in order to extract and synthesize data from them. In a systematic mapping, the data extraction activity is broad and the analysis is usually presented through graphic representations and/or tables summarizing and categorizing the data; and (iii) **Reporting**: in this phase the findings of the mapping study are used to answer the research questions. Then the results must be written and circulate them to potentially interested parties.

In addition to the searches in the databases, Kitchenham and Charters [8] suggest conducting a backward snowballing from reference lists of initially selected studies, in order to identify additional relevant studies. Forward snowballing can also be used to search for studies that cited the papers that are part of the selected studies [27, 28].

Next, we discuss the main protocol steps we performed for this systematic mapping.

Research questions. Based on our objectives, four main Research Questions (RQs) were defined. Table 2 presents the RQs as well as the rationale for considering them.

Search string. The search string considers three areas - Software, Performance Regression, and Software Testing. The areas and the search string adopted in this systematic mapping can be seen in Table 3. We

Table 2: Research questions and their rationales

N^o	Research Question	Rationale
RQ1	When and where have the studies been published?	This RQ aims at giving an understanding on whether there are specific publication sources for these studies, and when they have been published.
RQ2	What are the characteristics of the performance regression testing approaches?	This RQ investigates the approaches proposed and applied in the selected studies. This information is useful for researchers and practitioners who intend to carry out new initiatives on performance regression testing, as well as to guide future research towards new technologies in order to fill the existing gaps. Therefore, we analyze the techniques, application domain, test granularity, test frequency, target programming language, and supporting tools.
RQ3	How have the approaches been evaluated?	This RQ looks at how these approaches were evaluated in each study. This is an important question, since it can be used to evaluate the current maturity stage of the study and the area. For this, we analyze the subject systems evaluated in the studies, empirical standards used (e.g. experiments, case studies), comparisons with other techniques, and the presence of threats to validity.
RQ4	What are the main challenges reported regarding performance regression testing?	This RQ provides an overview of the main challenges reported on the studies regarding performance regression testing. The goal of this question is to point out the specific open issues or future directions that may motivate further research and contributions in the area.

executed the search string in three metadata fields: title, abstract and keywords.

Table 3: Systematic mapping - Areas and Keywords

Areas	Keywords
Software	"Software" OR "System*"
Performance Regression	"Performance Regression*"
Software Testing	"Test*" OR "Suite"
Search String: ("Software" OR "System*") AND ("Performance Regression*") AND ("Test*" OR "Suite")	

Sources. We decided to use the Scopus database, since it is considered the largest abstract and citation database of peer-reviewed literature. In addition, Scopus is the most commonly used database in Computer Science [29, 30], with more than 60 million records. It is important to emphasize that Scopus indexes papers of other international publishers, including Cambridge University Press, Association for Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE), Nature Publishing Group, Springer, Wiley-Blackwell, and Elsevier.

In order to add other studies and minimize the threat of missing relevant papers, we also conducted backward and forward snowballing. Regarding forward snowballing, we used Google Scholar. Some research on which databases are most appropriate for conducting forward snowballing (e.g., for updates) has shown that Google Scholar is most appropriate. Furthermore, the guidelines suggest that one iteration is enough [27, 28, 31, 32, 33].

Selection criteria. The selection criteria are organized in two inclusion criteria (IC) and ten exclusion criteria (EC).

The inclusion critera are:

(IC1) Study must target performance regression testing; and

(IC2) Studies published from 2012.

The exclusion criteria are:

(EC1) Study has no abstract;

(EC2) Abstract or extended abstract without full text;

(EC3) Study is not a Primary Study. The studies considered editorials, summaries of keynotes, tutorials, posters, conference Review, systematic mapping/review, survey of literature were excluded;

(EC4) The study is not related to Computer Science;

(EC5) Study is not written in English;

(EC6) Study is a copy or an older version of another publication already considered. In these cases, the most current version is considered;

(EC7) No access to the full paper;

(EC8) Study mentions performance testing, but it is not about performance *regression* testing;

(EC9) Study does not contain any systematic evaluation; and

(EC10) The study mentions testing, but it does not include performance regression testing.

Data storage. The publications returned in the searching phase were cataloged and stored appropriately. A data extraction worksheet was developed to catalog all relevant data from the identified studies. In this catalog, unique identifiers were assigned to each selected study, which facilitated data extraction at each selection phase. Each selection phase is detailed in the catalog as well as all necessary information to answer later the research questions and mainly to maintain management of systematic mapping activities. This catalog helped us in the classification and analysis procedures.

Assessment. Before conducting the mapping, the protocol created was tested. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to our investigation, called control group. We defined two studies in the control group, namely [34, 35]. In addition, in order to elaborate the search string, the set of search terms was devised in an iterative manner. We started with an initial set of search terms and iteratively calibrated this set until all pre-selected studies were found. The mapping process was conducted by all authors. We equally divided the initial set of studies to be analyzed. At each new activity and evolution of each phase (select, extract, and synthesize data), we exchanged some studies to obtain the perception and evaluation of another author in the group. This allowed avoiding biases throughout conducting the process. In addition, we held periodic meetings, usually fortnightly, so that the authors could follow the evolution of the activities, clarify issues, and align certain planning points, as well as the next activities.

4. Selection process

The main steps of our selection process are presented in Figure 1. We performed the search string, shown in Table 3, on the Scopus database. We considered the studies published from 2012 (IC2) until December 2022. As an initial result, 99 publications were returned. In the 1st stage, we applied the selection criteria (i.e., inclusion and exclusion criteria) over title, abstract and keywords, resulting in 79 papers (reduction of approximately 20%). In the 2nd stage, the selection criteria were applied considering the full text, resulting in 41 studies (reduction of approximately 48%).

In the 3rd stage, we performed a backward snowballing on those 41 studies. At this stage, 1744 studies were analyzed, that is, we conducted

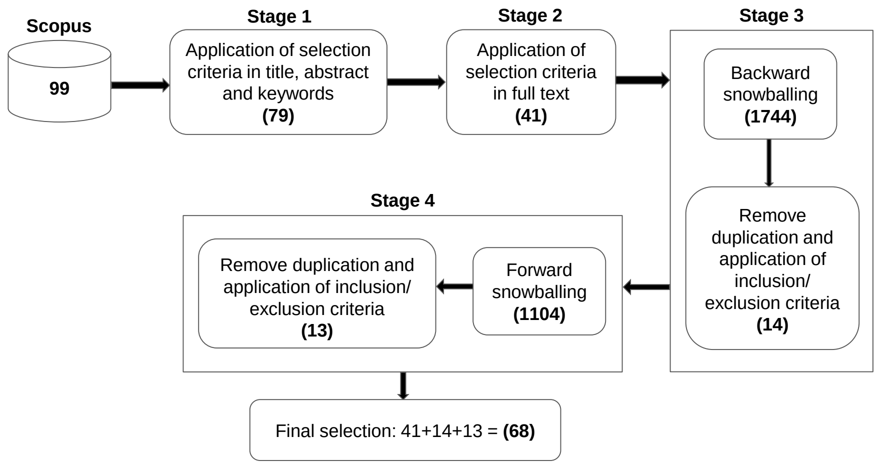

Figure 1: Search and selection process.

the entire selection process in 1744 studies. After removing duplicates and applying the selection criteria, 14 studies remained (99% reduction over the 1744 studies selected by backward snowballing). So far, 55 studies have been selected (41 from Scopus and 14 from backward snowballing).

Finally, in the 4th stage, we applied forward snowballing using those 41 studies as a starting set. We implemented a script to collect Google Scholar citations of the start set; we ran it in May 2023 and led to the identification of 1104 studies. After removing duplicates and applying the selection criteria, 13 studies remained (98% reduction over the 1104 studies selected by forward snowballing). As the final result, we obtained 68 studies to be analyzed (41 from Scopus, 14 from backward snowballing, and 13 from forward snowballing). Table 4 summarizes the stages and their results. Appendix A presents the bibliographic reference of the 68 selected studies.

Table 4: Results from the selection stages

Stage	**Applied Criteria**	**Analyzed Content**	**Initial Number of Studies**	**Number of Excluded Studies**	**Final Number of Studies**
1^{st}	IC1-IC2, EC1-EC5	Title, abstract, keywords	99	20	79

Continues

Stage	Applied Criteria	Analyzed Content	Initial Number of Studies	Number of Excluded Studies	Final Number of Studies
2^{nd}	IC1-IC2, EC6-EC10	Full Text	79	38	41
3^{rd}	Backward Snowballing, IC1-IC2, EC1-EC5	Title, abstract, keywords	(1744) References from 41 studies	1693	51
3^{rd}(b)	Backward Snowballing, IC1-IC2, EC6-EC10	Full Text	51	37	14
4^{th}(a)	Forward Snowballing, IC1-IC2, EC1-EC5	Title, abstract, keywords	1104	1060	44
4^{th}(b)	Forward Snowballing, IC1-IC2, EC6-EC10	Full Text	44	31	13
Final Result: 41 (electronic databases) + 14 (backward snowballing) + 13 (forward snowballing)= 68 selected studies					

5. Data extraction and synthesis

In data selection and synthesis, it is necessary to follow a classification scheme [9]. After applying the inclusion and exclusion criteria, 68 studies represent the approaches to be analyzed (see Table 4). We read the full text of 68 selected studies looking for contributions (e.g., techniques and tools) that reflect RQ2 and RQ3, thus identifying facets and their categories. These contributions can be presented as keywords. Once identified, the keywords can be grouped. Each set of keywords can form what we call facets, and the items in each set are called categories. In other words, when we identify a final set of keywords, we group them to form the categories.

This process of identifying keywords was done iteratively. We organized meetings between the authors to reach an agreement and validate the classification of the keywords. The resulting facets and their categories are presented below.

Approaches characteristics (RQ2): this facet aims to understand the approaches depicted in the selected studies, when applying performance regression testing. We have identified six main categories:

- **Techniques:** it is related to which techniques have been proposed and used in each study, for instance, Profiling or Machine Learning. These techniques can be classified according to the context in which performance regression testing is inserted.

- **Application domain:** this category is related to the different application domains targeted by the studies, for instance, Web-based systems and Database Management Systems (DBMS).

- **Granularity of tests:** granularity is related to the level of testing aimed at by the proposed approach, for example, unit tests, integration tests, and system tests.

- **Frequency of testing:** this category means when (frequency) the performance regression testing approach proposed in the study is performed. For example, for every commit, for every system versions (release), after each test run, and so on.

- **Target programming languages:** this category is used to classify, if applied, the target programming languages used by the approaches proposed in each study.

- **Tools:** this category is used to indicate whether the research developed a tool to support the proposed approach. Additionally, we report if they adopted some external tool that could be interesting for performance regression testing.

Evaluation (RQ3): this facet discusses the characteristics of the evaluations conducted in the analyzed studies. We have identified four main categories:

- **Subject systems:** it is related to the subject systems used in the evaluation. We identified three groups: open source projects, industrial (closed source) projects, and benchmark suites. For open source and industrial projects, we counted the number of projects evaluated in the study. We also analyzed the most used open source projects and their domain.

- **Experiments:** in this category, our intention was to understand the trend of investigations in the area of performance regression testing.

To achieve this, we investigated the empirical methods used in each study.

- **Comparison:** in this category we analyzed whether in the selected studies any comparison was made with other approaches, techniques or tools.

- **Threats to validity:** this classification is designed to identify whether the study reports the threats to validity or not.

For RQ1, we analyzed some statistics about the papers, such as publication venues, trend of studies over the years (e.g., number of journal, conference, and workshop papers per year), main authors and their countries of affiliation. Regarding RQ4, we extracted and summarized the main challenges that have been explicitly stated in the studies.

6. Results of the mapping

The results of the mapping study are presented in this section. We used the facets of the classification schema aforementioned to answer the RQs. Appendix B presents the mapping between the categories and the primary studies.

6.1. (RQ1) When and where have the studies been published?

Figure 6.1 shows the distribution of studies per venue where performance regression testing studies have been published. Almost three quarters come from events: 45 studies (66.2%) were published in conferences and five in workshops (7.4%). Over one quarter (18 studies - 26.5%) are journal articles.

Figure 3 depicts how the number of studies evolved over the years; the bars are divided by publication venue. The number of papers published over the years shows some variation. From 2013 to 2017, there is a growing trend with up to 7 studies published in 2017. Nevertheless, 2018 had a decrease to 5 studies and only 2 papers in 2019. On the other hand, several papers have been published in the last three years (27 studies $\approx$ 40%). Years 2020 and 2022 have more studies, with 11 and 12 studies, respectively. Proportionally, more journal articles were published recently, but over the years conferences have been the preferred forum. Workshop papers showed up only in years 2013, 2014, 2018, and 2020.

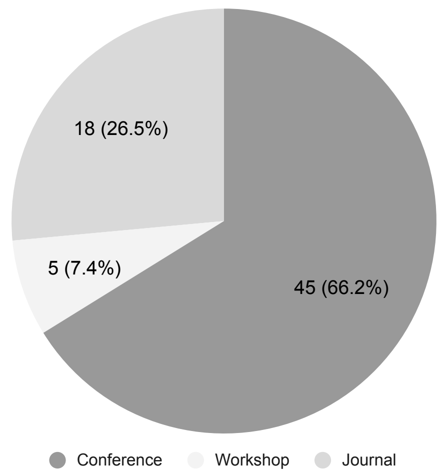

Figure 2: Publication venue.

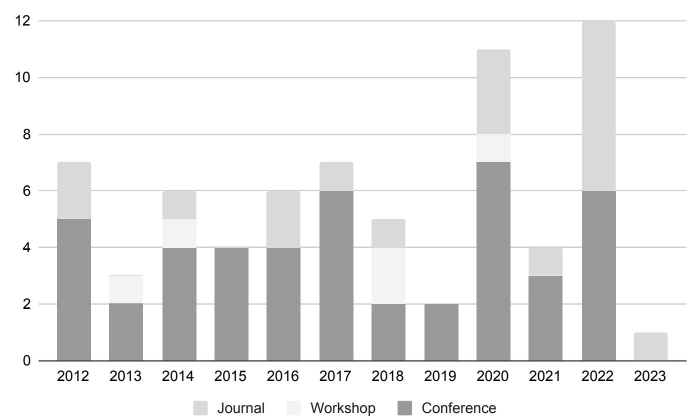

Figure 3: Studies per year, per publication type.

Table 5 brings the journals and the number of studies. The preferred venues are Empirical Software Engineering (4 studies), followed by IEEE Transactions on Software Engineering (3 studies). Most of the journals are related to Software Engineering, though there are journals from other Computer Science areas. Only one paper was published in a journal focused on performance (namely, Performance Evaluation Review).

Table 6 brings the conferences and the number of studies. ICSE has more

Table 5: Journals.

Name	**#Studies**
Empirical Software Engineering (EMSE)	4
IEEE Transactions on Software Engineering (TSE)	3
Advanced Robotics	1
Egyptian Informatics Journal	1
IEEE Transactions on Cloud Computing	1
IEEE Transactions on Knowledge and Data Engineering	1
International Journal on Parallel Programming	1
Journal of Software: Evolution and Process	1
Journal of Systems and Software (JSS)	1
Performance Evaluation Review	1
Science of Computer Programming	1
Software Testing Verification and Reliability	1
Swarm and Evolutionary Computation	1

studies (10), followed by ICPE with 7 studies. Other conferences with at least 2 studies are all related to Software Engineering (namely, MSR, ISSTA, ASE, and ICSME). The workshops with selected studies are: DBTest, LTB, RDSS, EPEW, and WOSP-C, each with one study.

Table 6: Conferences.

Name	**#Studies**
International Conference on Software Engineering (ICSE)	10
International Conference on Performance Engineering (ICPE)	7
Mining Software Repositories conference (MSR)	4
International Symposium on Software Testing and Analysis (ISSTA)	3
International Conference on Automated Software Engineering (ASE)	2
International Conference on Software Maintenance and Evolution (ICSME)	2
Symposium On Applied Computing (SAC)	1
Australasian Software Engineering Conference (ASWEC)	1
International Conference on Big Data (BigData)	1
International Conference on High Performance Computing (HiPC)	1
International Conference on IT Convergence and Security (ICITCS)	1
International Conference on Software Architecture (ICSA)	1
International Symposium on High-Performance Computer Architecture (HPCA)	1
Conference on Programming Language, Design and Implementation (PLDI)	1
International Conference on Cloud Computing Technology and Science (CloudCom)	1
International Conference on Software Testing, Verification and Validation (ICST)	1
International Conference on Neural Information Processing Systems (NIPS)	1
International Conference on Very Large Data Bases (VLDB)	1
International Conference on Cloud Computing (CLOUD)	1
SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)	1
International Conference on Web Engineering (ICWE)	1
International Conference on Intelligent Robots and Systems (IROS)	1
International Conference on Software Analysis, Evolution, and Reengineering (SANER)	1

For the 68 studies, there are 204 different researchers involved; each study has on average four authors (minimum: 1 author, maximum: 9 authors). We also look at how many studies researchers are involved with; Table 7 lists the most prolific researchers with at least 2 studies, for researchers with at least 3 papers, it also shows their affiliation. The top 4 authors are W. Shang - Concordia University (10 studies), A.E. Hassan - Queen's University (6), C. Bezemer - University of Alberta (5) and J. Chen (5) - Concordia University. Notice that there exist top researchers that come from companies like BlackBerry, SAP, and MongoDB Inc; this indicates the interest of industry in performance regression testing. We observed that 39 researchers conducted at least 2 studies, while 165 were involved in only one study.

Table 7: List of the most prominent researchers, ordered by the number of studies they contributed to. Authors of 2 studies are grouped, their affiliation is omitted for readability.

Name	**Affiliation**	**#Studies**
Shang, W.	Concordia University	10
Hassan, A. E.	Queen's University	6
Bezemer, C.	University of Alberta	5
Chen, J.	Concordia University	5
Flora, P.	BlackBerry	4
Jiang, Z. M.	York University	4
Lee, D.	SAP	4
Adams, B.	Polytechnique Montreal	3
Daly, D.	MongoDB Inc	3
Leitner, P.	University of Gothenburg	3
Liao, L.	Concordia University	3
Nasser, M.	BlackBerry	3
Alshoaibia, D.; Apel, S.; Arulraj, J.; Boehm, A.; Desella, T.; Eismann, S.; Grechanik, M.; Hoorn, A. V.; Ingo, H.; Kühne, S.; Laaber, C.; Li, H.; Liu, Y.; Mkaouera, M. W.; Mühlbauer, S.; Nguyen, T. H. D.; Pouwelse, J.; Pradel, M.; Rehmann, K.; Reichelt, D. G.; Sajedi, S.; Siegmund, N.; Souic, M.; Sporea, C.; Toma, A.; Wang, X.; Zeng, Y.		2

Figure 4 illustrates the collaboration network of researchers with two or more papers. In this network, nodes represent authors, and edges mean collaborations between them in at least one study. The largest cluster, located in the top left, contains most of the researchers, with 9 of the top-12 listed in Table 7. Many of them are affiliated with Canadian universities. A. E. Hassan is involved in the earlier studies, while W. Shang connects this group to another set of researchers working on more recent studies. The other three authors in the top-12 (D. Lee, D. Daly, and P. Leitner) are positioned in smaller clusters. The remaining researchers are spread across seven clusters, ranging in size from 1 to 4 nodes.

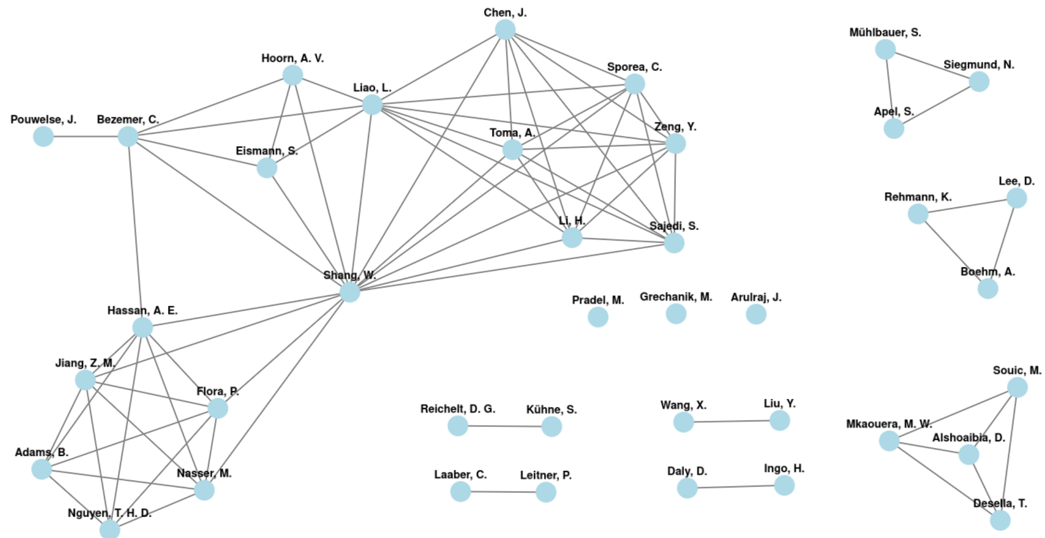

Figure 4: Collaboration between researchers with at least 2 studies.

Table 8 displays the identified countries along with the number of studies in which researchers from specific countries participate. Authors are affiliated with universities and companies of 27 different countries. On average, each study involves researchers from 1.4 countries. USA (26), Canada (20), and Germany (11) have shown substantial involvement in a multitude of studies. Next, there are Switzerland (5), China (4), Sweden (4), and South Korea (3).

Table 8: List of countries.

Country	**#Studies**
USA	26
Canada	20
Germany	11
Switzerland	5
China	4
Sweden	4
South Korea	3
Czech Republic, India, Saudi Arabia, Singapore, The Netherlands	2
Austria, Bolivia, Brazil, Chile, Egypt, Finland, France, Georgia, Ireland, Japan, New Zealand, Norway, Tunisia, UK, Vietnam	1

Summary of RQ1. The key findings concerning the main trends of the analyzed studies are:

- *The studies mostly come from events:* Around 74% were published in conferences or workshops. ICSE has the highest number of studies (10), followed by ICPE (7).
- *Studies published in journals:* The preferred venues are Empirical Software Engineering (4 studies), followed by IEEE Transactions on Software Engineering (3 studies). Most of the journals are related to Software Engineering.
- *Variation in trending:* The number of published papers shows some fluctuating variation across years. From 2013 to 2017, there was a growing trend; 2018 and 2019 presented a decrease; and the last four years (2020-2023) concentrate around 41% of the papers (upward trend again).
- *Researchers and country affiliations:* The most prolific researchers are Shang, W. from Concordia University (10 studies), Hassan, A. E. from Queen's University (6), Bezemer, C. from University of Alberta (5), and Chen, J. from Concordia University (5). Also, the studies involve researchers affiliated mainly in USA (26 studies), Canada (20), and Germany (11).

6.2. (RQ2) What are the characteristics of the performance regression testing approaches?

In this section we aim at characterizing the approaches presented in the selected studies. For this, we have identified six main categories, discussed next.

6.2.1. Classification of techniques

In order to understand which are and how the approaches characteristics have been established, initially we analyzed the techniques outlined for each one of the studies. As there are a huge number of studies, we group those techniques according to the main context, obtaining 12 clusters: Modeling, Profiling, Machine Learning, Statistics, Logic, Simulation, Code Analysis, Code Mutation, Evolutionary Algorithm, Scraping, Test Case Prioritization, and Interview. The three most considered techniques were Profiling with 19 studies, followed by Statistics and Machine Learning, with 12 studies each. Figure 5 shows all the identified clusters along with the number of studies for each one.

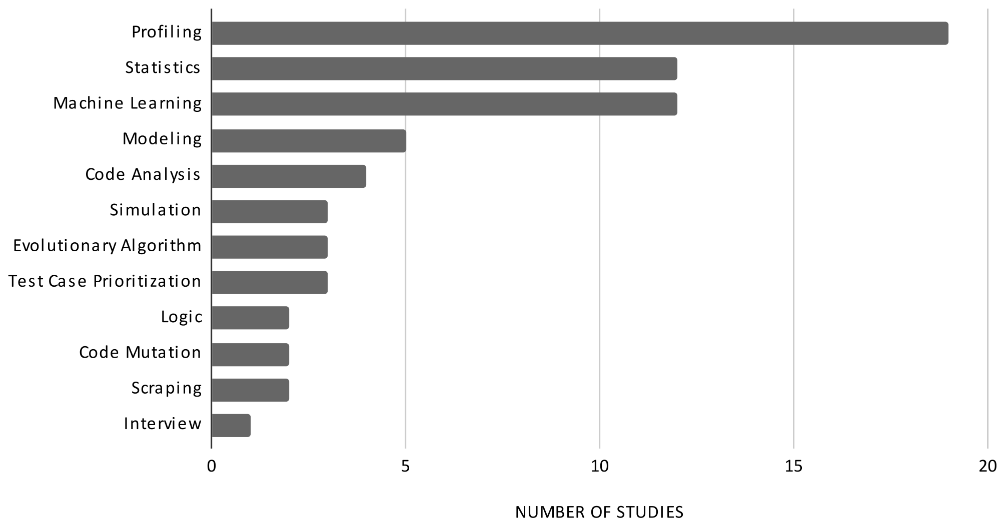

Figure 5: Classification of Techniques

The cluster Profiling refers to approaches that help comprehend the distribution of computational resources throughout program execution. Once having this information, one can use it to diagnose performance regression issues. As profiling is *de facto* technique to aid program optimization and performance engineering, we assume that the researchers would first try to leverage such a technique. In these studies, profiling is primarily accompanied by performance metrics at system level. Some of them are linked to special-purpose CPU registers that count hardware-related operations, also known as performance counters. They measure resource utilization related to the CPU (11 studies), memory (7 studies), disk I/O (8 studies), network I/O (4 studies), and threads (2 studies). Performance metrics are also collected at software level, where execution times (e.g., the duration of a function call, or response time of an HTTP request) appear in 9 studies. System-level metrics are usually gathered using OS tools like Perf (see Section 6.2.6), while software-level metrics are collected through tracing mechanisms. This involves instrumenting the system under test to capture pieces of information at runtime. Binary instrumentation is mentioned in 5 studies, and 2 studies use source code instrumentation. Finally, 12 studies adopt visualization of profiling data to support tasks, such as monitoring, reporting, detection and diagnosis.

As an example of a study using Profiling, Jalan and Kejariwal [36] introduce a Pin-based dynamic call graph extraction framework. The tool, called Trin-Trin, captures the run time spent in the kernel space. For this, it measures processor cycles (using the `rdtsc` instruction) for each thread. The time spent by a thread in the kernel space can be approximated by summing the run times of routines that are known to context switch into the kernel, which can help to characterize the multithreaded execution behavior of industry-standard benchmarks. On the other hand, Lee et al. [37] adopt Statistical Process Control (SPC) charts to detect performance anomalies and differential profiling to identify their root causes in DBMS development. Using the proposed framework, they removed most of the manual overhead in detecting anomalies and reduced the analysis time for identifying the root causes in regression testing. Also, Ocariza Jr [38] addresses bisection, which attempts to find the bug-introducing commit using binary search. The approach consists of analyzing the effectiveness of bisection for performance regressions. First, a metric that quantifies the probability of a successful bisection is formulated, and a list of input parameters that potentially impact its value is extracted. A sensitivity analysis is then conducted on these parameters to understand the extent of their impact.

Concerning Statistics, the studies that belong to this cluster present some kind of statistical analysis as their core technique, that is, by collecting and analyzing data it is possible to distinguish patterns and trends. Statistical analysis can show valuable trends and its significance is corroborated by being the second most used cluster. Most approaches adopt descriptive statistics in some form, such as proportions, means, thresholds, standard deviations, and moving averages. Time series analysis is also popular, showing up in 7 studies. Since performance regressions may be viewed as significant changes within a time series, 3 studies employed change point detection techniques. Additionally, differences between datasets collected from different software versions may suggest performance regressions, leading 4 studies to apply statistical hypothesis tests (2 of them specifically using the Student's t-test). Outlier detection, i.e., a technique to pinpoint data points that differ significantly from other observations and may signal a performance regression, is utilized in 2 studies. Other statistical methods appear individually. For instance, Jimenez et al. [39] apply sensitivity analysis, particularly statistical regression analysis (SRA), to application-independent performance feature vectors that characterize the performance of machines. This feature is used to automatically validate performance behavior across multiple revisions of an application's code base without having to instrument code or obtain performance counters. Another example is the exploratory study of Chen and Shang [14] about source code changes that introduce performance regressions. They conduct a statistically rigorous performance evaluation on commits from releases of Hadoop and RxJava. They repetitively run tests and performance micro-benchmarks for each commit while measuring response time, CPU usage, memory usage and I/O traffic.

Machine Learning (ML) has been widely adopted to support performance regression testing due to its enormous growth and importance in recent years. ML encompasses techniques that in some way carry out learning from either labeled datasets (supervised ML), or from unlabeled data without human intervention (unsupervised ML). We found 8 studies applying supervised ML, 2 studies using unsupervised ML, and 2 studies combining both. Among supervised ML, there are 7 studies with classification models, while 3 use regression models. Examples of classical algorithms adopted are: Logistic Regression, SVM, XGBoost, Random Forest, Linear Regression, Naives Bayes, Decision Tree, KNN, and Neural Networks. Within unsupervised ML, the main goal is to group unlabeled data using clustering algorithms like K-means, Hierarchical Clustering, Outlying Cluster Detection, and JRip. Deep learning algorithms appear in 3 studies: 2 adopted supervised algorithms like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory Network (LSTM); and 1 study used the unsupervised algorithm Autoencoder. As an example of a study using ML, Chen et al. [34] propose an approach that automatically predicts whether a test would manifest performance regressions given a code commit. They build random forest classifiers (a classifier based on random forests - a combination of tree predictors [40]) that are trained from all prior commits to predict in a given commit whether some test would manifest a performance regression or not. The results show that the approach can predict tests that manifest performance regressions in a commit with high AUC values (0.86 on average). In turn, Alam et al. [41] present AutoPerf – an approach to automate regression testing that utilizes three core techniques: (i) zero-positive learning, (ii) autoencoders, and (iii) hardware telemetry. First, they leverage hardware performance counters (HWPCs) to collect fine-grained information about run-time executions of parallel programs in a lightweight manner. Then, they utilize zero-positive learning (ZPL), autoencoder neural networks, and k-means clustering to build a general and practical tool based on this data. On average, AutoPerf accurately diagnoses more performance bugs than prior state-of-the-art approaches.

Additionally, we have also identified other important technique clusters, such as Modeling, Code Analysis, Simulation, among others. Each study related to all the clusters is reported in Table B.13 of Appendix B.

6.2.2. Application domain

Figure 6 shows the application domain targeted by the selected studies. Notice that a considerable amount of studies (17 studies - 25%) do not focus on a specific application domain, i.e., the studies discuss initiatives for the performance regression testing in general. On the other hand, looking at the studies that target an specific domain, three categories have greater representation: Web-based system (10 studies - 15%), DBMS (9 studies - 13%) and Libraries (8 studies - 12%).

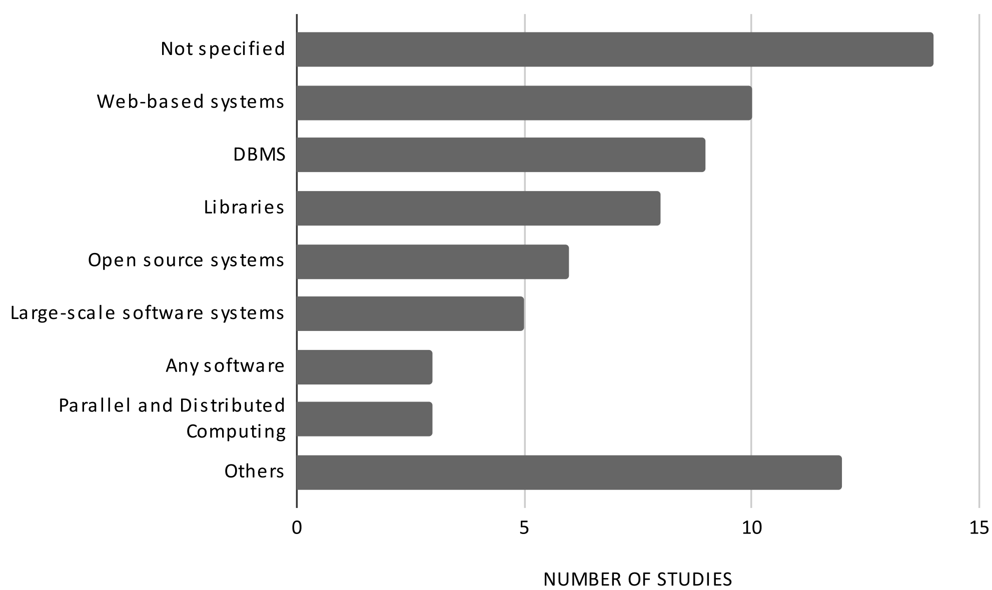

Figure 6: Application Domain

Web-based systems, such as search engines or social media, are becoming imperative in people's daily lives. This application domain needs to meet stringent performance requirements, since providing uninterrupted services to millions or even billions of users is paramount. For example, Liao et al. [35] present an approach to locate performance regression root causes in the field operations of Web-based Systems. A large-scale industrial software system was used as study. Ahmed et al. [42] investigate Application Performance Management (APM) tools to identify performance regressions. In the study, the authors took into account three open source web applications (PetClinic, CloudStore, and OpenMRS). The study focuses on web applications because some of the APM tools studied (namely, New Relic and Pinpoint) do not support standalone applications and these tools are more commonly used for Web-based systems.

Similarly, DBMSs also have stringent performance requirements, as they are considered critical components of data-intensive applications. Building DBMSs involves various aspects of performance analysis, e.g., query execution speed, query optimization speed, standards compliance, achieving feature parity, modularity, and portability. Therefore, several studies can be found proposing strategies to automatically detect performance regressions in DBMSs. For example, Jung et al. [43] propose a toolchain, called APOLLO, for automatically detecting, reporting, and diagnosing performance regressions in DBMSs. The study of Lee et al. [37] presents a framework that allows you to eliminate almost all manual overhead in performance testing to detect performance anomalies for a real-world DBMS and find their root causes during regression testing. Finally, Liu et al. [44] introduce AMOEBA: its main idea is to build two semantically equivalent queries, and then compare the time it takes the DBMS under test to execute the two queries in order to identify a possible performance bug.

Libraries are related to a set of functions (or classes) implemented in a language and can be imported by the developer. Studies focused on performance regression testing in software libraries are also identified. Chen and Shang [14] perform an exploratory study on source code changes that introduce performance regressions. The authors analyzed commits from RxJava, a Java library for composing asynchronous and event-based programs. In another study [45], the performance evolution of four Python libraries was analyzed with respect to change points.

In addition to the three most investigated application domains presented above, it is important to emphasize the diversity of other application domains investigated to detect performance regression problems, for example. open source system, parallel and distributed computing, microservices, robotic systems and mobile applications. Some application domains identified in just one or two studies were grouped into a classification called “Others” (see Figure 6). The studies in each one of these application domains are presented in Table B.14 of Appendix B.

6.2.3. *Granularity of tests*

We consider the test level (granularity of tests) targeted by the proposed approaches. For instance, unit testing, integration testing, system testing, that is, single components, their communication (integration), or the system as a whole. Figure 7 shows the distribution of the studies considering the addressed granularity level.

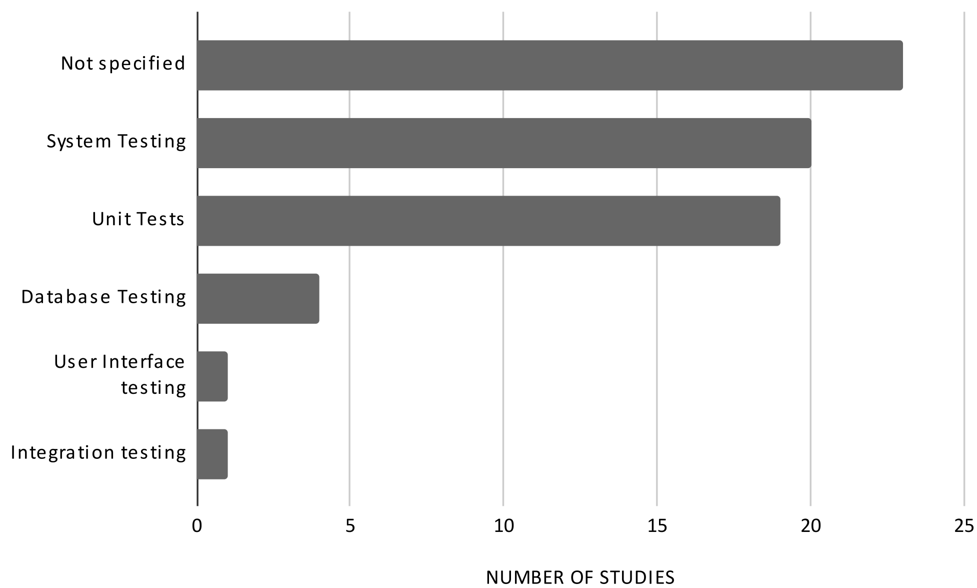

Figure 7: Granularity of tests

Notice that a high number of papers did not specify the granularity applied. Considering the ones that did, the majority reports system testing (20 studies) and unit tests (19 studies). For example, Laaber et al. [46] bring the analysis of a specific type of performance test, i.e., software microbenchmarks. They typically measure execution runtime of small software components, such as methods or statements (i.e., unit tests). In turn, Hindle [47] presents a case study where over 500 versions of Firefox 3.6 are tested, showing variation of power consumption between versions (at system testing level). Database, User Interface, and Integration testing show up with the fewest number of studies.

6.2.4. Frequency of testing

Figure 8 shows the number of studies by testing frequency. Observe that "Continuous time" (25 studies), "Commits" (16 studies) and "System versions" (15 studies) are the most considered moments in which the approach proposed in the study on performance regression testing occurs.

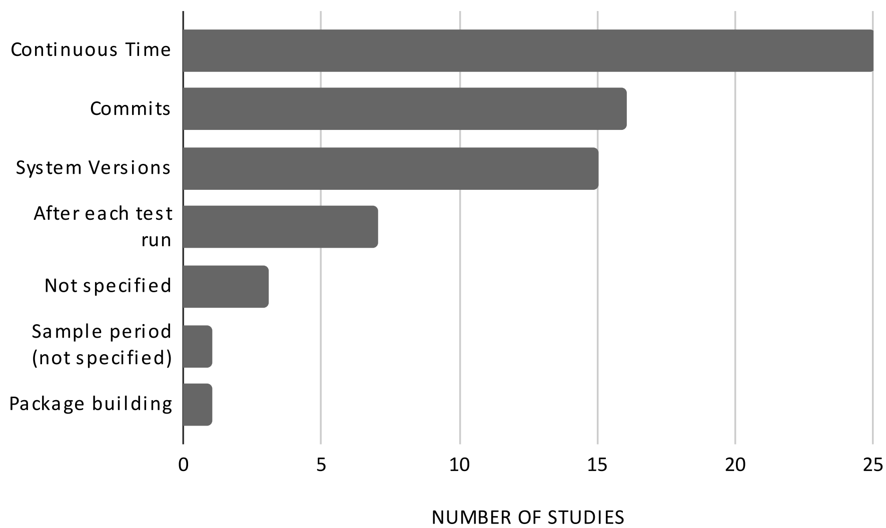

Figure 8: Frequency of testing

Yoshimura et al. [48] focus on containerized applications, and a continuous-time regression analysis is proposed. According to the authors, small updates to containers, without changing their versions, also affect application performance. Therefore, it is important to determine the performance impact on different container images along with continuous monitoring of small changes. In relation to commits, De Oliveira et al. [49] propose a strategy to predict which commits will cause performance changes on the specific benchmarks under analysis. Regarding system versions (or releases), some studies conduct performance analyses to verify whether a system version presents performance regression in relation to a previous version. This is the case for Liao et al. [50], in fact the proposed approach compares black-box models derived from the current version of the system with a previous version to detect performance regressions between these two versions. Other categories are presented in Table B.16 of Appendix B.

6.2.5. Target programming languages

We investigated which target programming languages were used in each study, that is, which languages were aimed by the approaches proposed in each study. Figure 9 shows the number of studies considering the target programming languages. Out of the 68 studies, 41 (60%) do not specify a target programming language in the study. For those that did, most studies target Java (14 studies) and C/C++ (6 studies).

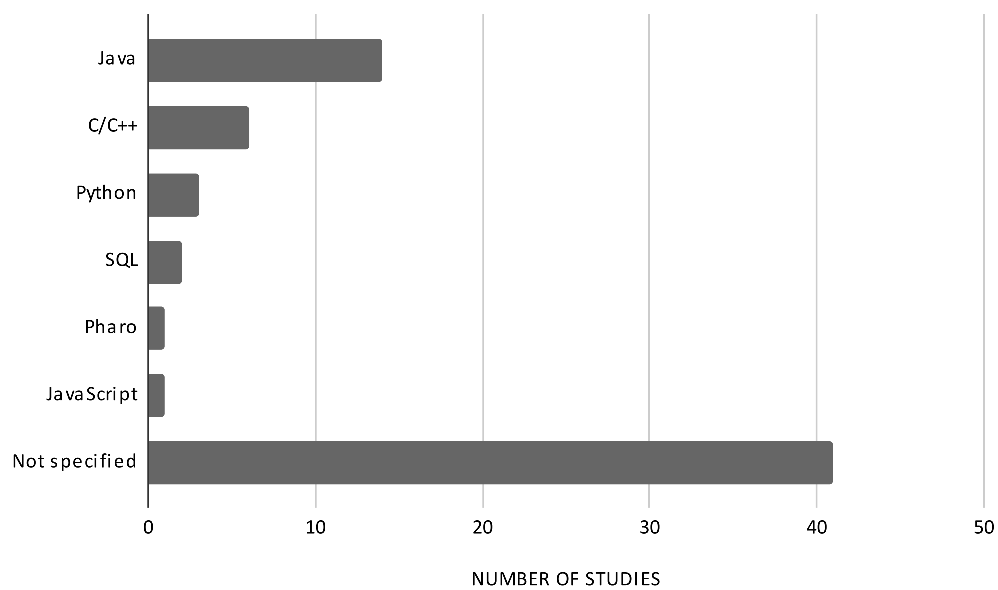

Figure 9: Target programming languages

Various approaches proposed by the selected studies were evaluated considering different applications and written in Java language. This is the case of study [51], the authors proposed a recommendation system, PerfImpact, which aims to recommend code changes responsible for performance regressions. To evaluate PerfImpact, two open source web applications were used. Both applications were developed in Java language. Another example is the study presented in [52]. In this study, a tool, called PReT, was developed to perform automatic performance regression tests on software. PReT was developed considering Java applications. The tool has a performance monitoring module that continuously monitors Java applications running on the system. A method for specifying and performing performance tests for individual components of component-based robotic systems was proposed in [53]. Java language was chosen as the target language. According to the Wienke and Wrede [53], Java provides the performance necessary to generate heavy loads, while also providing a relatively easy-to-use.

Similarly, the C/C++ language was chosen as the target language in several other studies. In [54], a new lightweight and white-box approach, performance risk analysis (PRA) was proposed, to improve performance regression testing efficiency via testing target prioritization. PRA works on regular performance-critical software written in standard C/C++. In another example, presented in [55], the PERUN tool is proposed. This tool allows profile-based performance analysis. PERUN has C++ as the target programming language. The other target languages identified in each study are presented in Table B.17 from Appendix B.

6.2.6. Tools

The purpose for the tools category is to identify if the studies developed frameworks, platforms, and/or algorithms to support the approaches presented, and also if they adopted some tool of interest to conduct the research. As for the development of new tools, 39 out of 68 studies (57%) describe the implementation of a new one to perform the experiments. We found that 15 tools only have an active website[7]. For example, AMOEBA [44] constructs semantically equivalent query pairs, run both queries on the DBMS under test, and compares their response time. If the queries exhibit significantly different response times, that indicates a potential performance bug in the DBMS. Also, Bhattacharyya and Amza [52] present PReT, a tool which does non-intrusive profiling based on application snapshots to learn behaviour for performance regression tests and can identify any changes in

[7]The tools' websites are listed per study in the supplementary material.

the testing behaviour by comparing the current behaviour against a learned model. PReT annotates resource usage profiles with application stacktraces and uses a variation of k-means to learn the models for regression test online.

Considering the studies that adopted some tool of interest, it is worth mentioning JMeter[8] (quoted by 5 studies) and Perf[9]/Perfmon[10] (the performance analyzer for Linux and Windows, respectively), likewise quoted by 5 studies; and also JUnit[11], quoted by 3 studies. For instance, Liao et al. [50] extensively applied JMeter in order to accomplish their results: for simulating a more realistic workload in the field, adding random controllers and random order controllers. Also, they designed a total of five JMeter-based performance tests. Moreover, they used JMeter to create performance tests that exercise Apache James. Alshoaibi et al. [56] used Perf to collect dynamic information in order to perform the analysis of commits having performance change. Still, Hewson et al. [57] proposes Buto, a framework that was integrated into the popular JUnit testing tool. The idea is to have a Java runtime ecosystem unification, which reduces the barriers to adaptation. In total, 40 studies stated the adoption of some tool to achieve the results, that is, 59% of the studies used an external tool to support their research.

[8] `https://jmeter.apache.org`

[9] `https://perf.wiki.kernel.org/index.php/Main_Page`

[10] `https://en.wikipedia.org/wiki/Performance_Monitor`

[11] `https://junit.org/junit5/`

Summary of RQ2. To understand the developed approaches, six main categories were identified:

- *Technique clusters.* The methodologies outlined for each study were classified according to their main contexts. We have identified twelve different clusters of techniques, the most used are Profiling (19 studies), Statistics (12 studies), and Machine Learning (12 studies).
- *Application domain.* A considerable amount of studies (14 studies) did not specify the domain of application. Other studies presented the following domains: web-based system has the highest number (19 studies), followed by DBMS (9 studies), and libraries (8 studies). The remaining studies show different domains (e.g., mobile apps, or robotic systems) that are not representative as a cumulative indicator.
- *Granularity of tests.* We mean the test level targeted by the proposed approach. Twenty three studies did not specify the level tested. For those that did, the preferred levels were system testing (20 studies) and unit tests (19 studies).
- *Frequency of testing.* Concerning the moments in which the testing methodology proposed in the study occurs, that is, the frequency that the tests were executed, continuous time was the most considered (25 studies), followed by commits (16 studies), and system versions (15 studies).
- *Target programming languages.* We investigate the target programming language proposed in each study, and we get Java (14) and C/C++ (6) as the most preferred languages. However, the majority of the studies (41) did not mention a specific programming language, as they present a more general purpose and can be applied to different implementations.
- *Tools.* Relating to development of new tools, 57% of the studies produced a new one to perform its experiments. Considering the studies that adopted some other tool of interest as part of their methodology, we can mention JMeter (5 studies), Perf/Perfmon (5 studies), and also JUnit (3 studies).

6.3. (RQ3) How the approaches have been evaluated?

To study the evaluation of the selected approaches, we investigated the following aspects: (i) the subject systems (i.e., the systems used to perform the experiments), (ii) the types of experiments that have been performed (according to the Empirical Standards[12]), (iii) the comparison with other techniques/tools, if any; and (iv) the threats to validity of the approaches if outlined by the authors. Hereafter are the results of our investigation.

6.3.1. Subject systems

When evaluating the proposed approaches, 54 studies used at least one open source project, 19 studies adopted at least one industrial (closed source) project, and 9 studies employed some benchmark suite. The selected studies may have employed different types of subject systems, either individually or in combination.

Table 9 shows the combinations of subject system types, in how many studies these combinations were utilized, and their corresponding percentages. Several studies (58%) revolve their evaluation around open source projects only. Industrial projects (11.8%) and their combination with open source projects (14.7%) appear next. Benchmark suites are less used, alone (7.4%) or combined with open source projects (4.4%). Just the study of Jalan and Kejariwal [36] used the three types of subject systems in their evaluation. Another study (None) used neither software projects nor benchmark suites in its evaluation; instead, Lee and Park [58] use simulation data, generated using the fuzzy logic toolbox of Matlab, to evaluate the proposed approach.

As for the studies with open source and industrial projects (62 out of 68 studies), we counted the number of different software projects used in their evaluation. On average, the studies evaluated 3.1 projects. Most studies present empirical evaluations with up to 3 projects (48 studies). The remaining studies (14) adopted 4 or more projects; only 5 studies adopted 7 or more different projects. The studies of Sandoval Alcocer et al. [59] and Ocariza Jr [38] used more projects in total, both evaluated 17 projects.

There is no project that is widely evaluated in the performance regression testing studies, as many of them showed up in 1 or 2 studies. Table 10 lists

[12] https://sigsoft.org/EmpiricalStandards/docs

Table 9: Types of subject systems used.

Type	#Studies	Perc.
Open source projects only	40	58.8%
Industrial projects only	8	11.8%
Open source and industrial projects	10	14.7%
Benchmark suites only	5	7.4%
Open source projects and Benchmark suites	3	4.4%
Open source and industrial projects, and Benchmark suites	1	1.5%
Industrial projects and Benchmark suites	0	0.0%
None	1	1.5%

the projects featured in 3 or more studies. Among them, there are 3 Web-based systems (namely, Dell DVD Store, JPetStore, and OpenMRS), and 4 DBMSs (namely, MongoDB, MySQL, Cassandra, PostgreSQL).

Table 10: Most used open source projects.

Project Name	Description	#Studies
Dell DVD Store	Web-based system	4
Hadoop	Distributed processing of data sets	4
JPetStore	Web-based system	4
MongoDB	NoSQL DBMS	4
MySQL	Relational DBMS	4
Apache Commons-IO	Java library for IO	3
Cassandra	NoSQL DBMS	3
Firefox	Popular web browser	3
Git	Distributed version control system	3
OpenMRS	Web-based system	3
PostgreSQL	Relational DBMS	3

Whenever possible, we identified the project domain and how many studies evaluated at least one project in the given domain. Notice that, in RQ2, we look at the application domain targeted by the proposed approach/tool; here we analyzed the domain of projects used in the evaluation. Table 11 shows the domains of subject systems used in the evaluation. Software libraries are particularly present, followed close by DBMSs. Web apps, software engineering tools, and distributed systems have also drawn some attention.

6.3.2. Empirical standards

Our interest in these standards is motivated by the need to learn the trend of investigations in the field of performance regression testing. As expected, a large portion of studies (55%, i.e., 38 over 68) refers to *Engineering*

Table 11: Project domains of subject systems.

Project Domain	**#Studies**
Libraries	20
DBMS	19
Web-based system	16
Software engineering tool	11
Parallel and distributed computing	6
Utility	5
Browser	3
Web server	3
Desktop app	2
Game	1
Mobile app	1

Research that deals with promoting novel technological artefacts that are also evaluated. The second top standard is represented by *Case Study* adopted by 29% (i.e., 20 over 68) of the considered papers that take into account the application of techniques in a real-world context. The remaining papers belong to the following standards with very small percentage values of occurrence: 0.06% *Longitudinal*, 0.03% relates to *Experiments* and *Benchmarking*, and finally 0.01% adopts *Repository Mining* and *Quantitative Simulation*. We also check whether the studies make experimental data available and/or provide replication packages. Of the studies reviewed, 19 (28%) include links to online repositories with experimental data. Overall, we found it quite relevant that engineering research and case study were widely spread in the selected approaches, thus confirming that performance regression testing is a practical field of research, i.e., techniques are applied in real-world scenarios.

6.3.3. Comparison

When classifying the approaches, we found that 68% (i.e., 46 over 68) do not perform any comparison, whereas the 32% of the papers (i.e., 22 over 68) make an effort to assess the added value of their technique/tool with respect to a reference baseline. In some cases, the baseline is artificially built (e.g., the proposed approach is compared with random testing, see [60]). State-of-the-art approaches are also used to run a comparison, e.g., Perphecy [49] seems quite popular as a baseline, it has been used in [56, 61, 3]. To facilitate the comparison among different approaches, one important aspect is to make artifacts (e.g., data, implementation code, test cases) publicly available, thus fostering the replicability of the results.

6.3.4. Threats to validity

We verified if the studies discuss the threats to validity with respect to the evaluation performed. Figure 10 summarizes whether or not the threats to validity are described in the analyzed studies. Almost half of them (31 studies – 45.6%) follow known guidelines for empirical research and have a specific section in the paper to identify the threats and discuss actions for mitigation. Moreover, 11 studies (16.2%) present them partially, so some threats to validity are discussed in other sections like future work and limitations. For 26 studies (38.2%), there is no explicit mention to threats to validity.

Summary of RQ3. To study the evaluation of the selected approaches, we investigated the following aspects:

- *Subject systems.* The selected studies may have employed different types of subject systems, either individually or in combination. 58% revolve their evaluation around open source projects only. Industrial projects alone appear next, in 11.8% of the studies. Benchmark suites are less used alone, with 7.4%. 62 out of 68 studies used either open source and industrial projects. Projects that were most featured in the studies (4 studies each): Dell DVD Store, Hadoop, JPetStore, MongoDB, and MySQL.

- *Experiments, according to the empirical standards.* A large portion of studies (55%) refers to Engineering Research that deals with promoting novel technological artifacts that are also evaluated. The second top standard is represented by Case Study adopted by 29% of the considered papers that take into account the application of techniques in a real-world context. Overall, this leads one to believe that performance regression testing is a practical field of research, i.e., techniques are applied in real-world scenarios.

- *Comparison with other techniques.* We found that 68% of the studies do not perform any comparison, whereas the 32% make an effort to assess the added value of their technique/tool with respect to a reference baseline. Besides, this baseline can be artificially built and Perphecy [49] seems to be a popular baseline, in fact it has been used in three studies.

- *Threats to validity outlined.* 45.6% of the analyzed studies have a specific section in the paper to identify the threats and discuss actions for mitigation. Moreover, 16.2% present them partially, thus some threats to validity are discussed in other sections like future work and limitations. For 38.2%, there is no explicit mention of threats to validity.

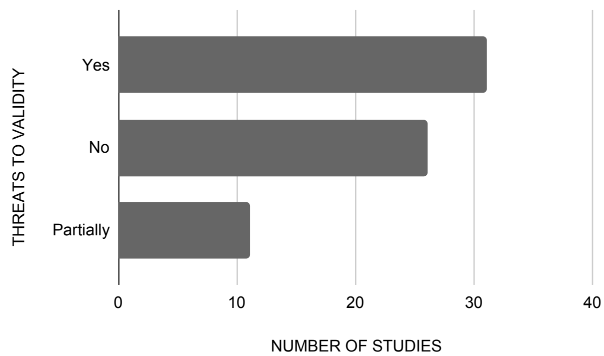

Figure 10: Threats to validity.

6.4. (RQ4) What are the main challenges reported regarding performance regression testing?

Regarding *challenges*, the aim is to enumerate unresolved issues that have yet to be addressed, alongside an analysis of future research directions. A substantial portion of the selected studies (82%, or 56 out of 68) explicitly mentioned these challenges. Only a minor fraction, 18% (12 out of 68), did not discuss challenges. Upon examination of those studies that did mention challenges, it becomes evident that categorizing these challenges is not straightforward, as a diverse array of challenges are presented. They were grouped in the following clusters: Strategies for Effective Analysis, Root Cause Analysis, Application Code and Metrics, and others named as Miscellaneous. A summary of the related challenges is provided hereafter.

Strategies for effective analysis.

- Determine sampling frequency for characterizing metrics [62, 35, 50, 59], and evaluating different sampling strategies thus reducing time to detect performance regressions. Besides, the sampling can be based on functions to improve the overhead and to detect the bottlenecks [36].

- Replicate experiments across different programming languages [59] and investigate different statistical debugging models [43], as well as support several database systems.

- Avoid software contention for resources among different processes or devices besides testing software evaluation [63]. Receive feedback from practitioners to develop more realistic testing scenarios [64].

- Deal with analysis of data from open repositories (e.g., GitHub) and track bugs or other issues in a timeline, so that one may compare data collected overtime. The challenge refers to detecting patterns and performance trend regardless of changes in tests or even how they were conducted [57].

- Identify performance thresholds and conduct statistical tests [65]. Explore more Statistical Process Control (SPC) techniques to be employed in software testing [66]. Remove tests that do not cover realistic scenarios and focus on test coverage that reveal only specific performance issues [4]. Recommend only those cases that can expose software versions that are risky [54]. Large studies on industrial scenarios to evaluate accuracy of the approaches to accurately identify the location and the type of regression [67].

Root cause analysis.

- Explore lack of standards and tackle the continued integration issues as well as root causes dealing with requirements not well specified [68].

- Explore more powerful indicators (besides response time, utilization, throughput, system Logs, etc.) as well as employing machine learning techniques to predict such parameters [49].

- Test real-case scenarios to calibrate indicators and define a threshold to accept/not accept the resulted values. More emphasis has been explored on Android platforms and there could be more research towards other platforms [69]. The same authors also suggest to improve web applications with more effective root cause analysis besides the need to handle multi-thread codes.

Techniques for automation and analysis.

- There is a concern in assuring stability in defining performance metrics as well as some environments for benchmarks to be explored so that tools can embed these aspects. Another concern is with respect to give more stress to automatically identify root causes as well as studying to come up with new applications of analyses [70, 37].

- There are some applications using Application Performance Monitoring (APM) tools for performance regression. However, some studies point out that the output may not be consistent with applications to identify and detect performance regression issues, leading to the need for research to minimize the gap between APM and performance regression detection [42].

- Analysis must be improved by combining work load monitoring with static checking and dynamic analysis [71].

Application code and metrics.

- Use code-level metrics to improve the accuracy of performance regression testing [56, 72]. Such metrics can be used to prioritize test cases, thus enabling the detection of faults at the early stages in the testing process.

- Deadlocks can pose challenges to detect performance bugs in application codes. One possible way to overcome these challenges is to extract dynamic information from executing tests with additional metrics. Further investigations are required into the granularity of code commits as well as the accuracy of the metrics [34].

- Investigate the differences between Database Transaction Systems (DBT) and regular software systems as they pose challenges [73]. The main issue is the limited availability of both test codes and hardware platforms targeted by DBT systems making the test process more complicated.

- Explore reward functions to improve detection of software performance regression [74]. This study suggests to improve mechanisms to compare several versions using better statistical tests, e.g., Welch's t-test, Mann-Whitnet U-test.

Miscellaneous.

- Analyze client-server systems [45] and evaluate the performance degradation, if any, with different programming languages besides looking for variety of bugs beyond performance [52].

- Enhance performance regression by considering peak resource usage as it might be sensitive in being detected [53]. Another recommendation is to develop tools that can automatically evaluate change impacts [14]. Reichelt et al. [75] suggests to isolate root causes of performance changes, thus enabling a clearer analysis. This study and Muhlbauer et al. [76] also recommend to establish some standards, thus defining classes of performance changes, besides exploring the prioritization of manual measurements with configuration constraints.

- Consider both functional and non-functional scenarios while creating test suites [77, 78]. Mostafa et al. [77] also suggest to investigate what would be the ideal number of test executions to make the performance evaluation more effective.

- Monitor at a finer level of granularity to accurately capture the performance changes besides exploring a broader historical commits (not just against the last commit) to better evaluate performance [79].

Summary of RQ4. Several unresolved challenges in performance regression testing are outlined stressing the importance and need for future research. 82% of the selected studies identified such challenges that are grouped into 5 clusters. **Strategies for effective analysis**: key issues include determining optimal sampling strategies, replicating experiments across different programming languages, addressing resource contention in software, and analyzing data from open repositories. Also suggested to look for statistical methods for identifying performance thresholds. **Root cause analysis**: challenges involve the lack of standards, enhancing indicators for performance metrics, and the importance of real-case testing, especially on diverse platforms. **Techniques for automation and analysis**: concerns include ensuring stability in performance metrics, the integration of Application performance monitoring (APM) tools with performance regression detection, thus improving analysis through workload monitoring. **Application code and metrics**: utilizing code-level metrics for early fault detection, addressing challenges posed by deadlocks, and recognizing differences between database transaction systems and regular software systems. **Miscellaneous**: various additional challenges include evaluating client-server systems, considering peak resource usage, establishing standards for performance changes, and ensuring a comprehensive approach to test suite development. Overall, a more nuanced exploration and solutions are recommended to face these challenges in software performance regression testing.

7. Research opportunities

This section argues on the main research directions that raised from the selected studies as part of future challenges, and leading to research opportunities. In the following, we distinguish aspects that relate to software engineering practices, complemented with more specific directions in the software quality domain.

7.1. Opportunities for software engineers

Verifying the system requirements at the earliest. The assessment of requirements is recognized as of key relevance in software engineering [80], and it has been noted that within the software development process, late fixes are much more expensive [81]. The early validation of non-functional requirements is indeed valuable [82], and the development of strategies to interpret the performance regression testing campaign is much appreciated. This constitutes a strong opportunity for software engineers since they can get informed about the most critical system components or interactions [83], and this information can be precious in the later development stages [84].

Supporting open source development of analysis tools. An examination of tool development reveals a prevalence of proprietary tools, often tailored for specific purposes within individual studies. In some selected studies, it proves challenging to discern whether a tool was developed or if the code was simply programmed to validate performance regression evaluations. A notable gap exists in the availability of open-source or freely accessible systems for performance regression, and little has been discussed regarding the automation of performance regression testing. This is indeed an enticing opportunity for exploration, especially if software engineers make an effort in developing open source and modular tools that guarantee flexibility in the specification of different analysis modules [85].

Integrating testing in continuous integration and deployment pipelines. Continuous Integration and Continuous Deployment (CI/CD) are known software development practices to efficiently integrate code changes while assuring reliable releases at any time. However, CI/CD pipelines may lead to inefficient use of system resources, and consequently a delayed feedback to software teams throughout the development process [86]. Software engineers can play the key role of adopting tools for performance regression testing in the CI/CD pipelines to (i) enable automated, frequent, and regular testing, and (ii) save time when compared to manual testing [87].

7.2. Opportunities for software quality engineers

Exploiting the target application domain. A notable gap in the existing literature pertains to comprehensive investigations in certain areas, including concurrent processes, embedded software, cyber-physical systems, robotics, computer graphics, image processing applications, mobile applications, and artificial intelligence (AI) applications. For instance, embedded software deserves particular attention due to its criticality depending on the target application. Consequently, software quality engineers do have the opportunity to assess performance regressions on the basis of the application domain, e.g., safety-critical systems may deserve more attention [88].

Minimizing the costs of software refactorings. The prioritization of testing efforts should be contingent upon the potential impact or risk associated with specific applications. This enables the development of efficient strategies for resource allocation in testing. In fact, addressing mutation analysis emerges as another crucial aspect, due to incrementally evaluating the nature of changes and their impact on software systems. Software quality engineers play a key role in pointing out which software refactorings are probably improving the performance characteristics of software systems [89].

Understanding the peculiarities of Artificial Intelligence (AI) applications. Examining AI applications, it becomes apparent that there exists a substantial research opportunity, given their widespread popularity. Virtually every newly implemented or released system emphasizes its intelligent capabilities. Hence, there is strategic importance in delving into performance regression testing within these systems. Software quality engineers can act on identifying (and possibly even anticipating) potential root causes of performance problems while developing AI-based software products [90].

Evaluating real-world software systems. A large portion of the selected studies lacks the evaluation on real scenarios, likely due to the challenges associated with conducting tests, which can be a resource-intensive endeavor. The majority of the studies focused on libraries, frameworks, DBMSs, and web applications. This is a shortcoming, software quality engineers are indeed encouraged to pursue the opportunity of evaluating real systems and collecting actual performance measurements. Such measurements might reveal glitches probably hidden in whatever system abstraction [91].

8. Threats to validity

As other empirical studies, the systematic mapping study herein presented is subject to threats to validity. Hereafter we discuss the validity concerning the four groups of common threats to validity [92].

Construct Validity. Threats of this nature deal with problems that may arise during the research design [93, 94], thus affecting the identification of relevant primary studies, and consequently the derived findings. To smooth the threat on the selection of the studies, we use the Scopus database, i.e., a widely recognized source for computer science. Besides, papers from other repositories are also included. The search string for Scopus is constructed interactively and tested with a control group. We also perform both backward and forward snowballing to complement the searches, thus consolidating the set of considered papers. Besides this general and always valid construct validity threat, another aspect is represented by the publishing period that we limit from 2012 to 2023. The upper bound is established according to when we performed the literature search, whereas the lower bound is motivated by the relevant set of papers in 2012 resulting as the top third proficient year of publication, as shown in Figure 3. Overall, we are aware that our findings strongly depends from the identified studies, however such papers belong to very diverse venues (journals, conferences, workshops) and span on a 10 years publication period. Hence, the final set of considered papers can be regarded as a good and unbiased representation of the research in performance regression testing domain.

Internal Validity. Study selection and data extraction may be influenced by subjective decisions and potentially introduce bias. As for study selection, each reviewer strictly adhered to the study protocol and tracked the inclusion and exclusion criteria applied for all candidate studies. Any uncertainties surrounding certain studies were discussed and consensually resolved in synchronous meetings. Concerning data extraction, we used similar procedures. Besides, the extracted data was also revised by a different reviewer during the analyses.

External Validity. We only considered papers written in English and this may pose a possible threat. As the majority of scientific papers are written in English, we believe this bias is minimal. We also removed papers that are not peer-reviewed primary studies or lack systematic evaluation. This threat to validity comes from the study design, as we aimed to select high-quality studies with sufficient pieces of information to address the research questions.

Conclusion Validity. An identified threat concerns the classification we used. The selected studies may also omit pieces of information relevant to our analyses. To reduce these threats, we meticulously extracted and reviewed the data from the studies, and iteratively evolved the classifications. We also held meetings to present and validate the classifications. Nevertheless, we cannot exclude other researchers may propose different classifications.

Concerning repeatability, all authors were actively engaged as reviewers in all steps of the systematic mapping process. We think this improved the clarity and documentation of the protocol. Moreover, we have provided all the necessary resources (study protocol, raw data, and scripts) to enable future replications of our study.

9. Conclusion

Nowadays, software systems evolve continuously, their frequent changes may inadvertently disrupt the system's behavior and provoke unwanted degradation of performance. To prevent this, performance regression testing has been extensively investigated in the literature. This paper presents a systematic mapping study that aims to characterize the current landscape of approaches for performance regression testing. We selected and examined 68 primary studies from the perspective of publication trends, approach characteristics, evaluation, and the main challenges reported within the field.

From the mapping study results, we can highlight the following main takeaway messages on the characteristics of the selected approaches: (i) the interest in the performance regression testing research field is constant over the years based on publications in different venues and the network of collaboration between the involved researchers; (ii) the techniques outlined by most approaches refer to profiling (28%), statistics (17.6%) and machine learning (17.6%) activities; (iii) of those studies that made the application domain explicit, web-based environments have been selected as target systems of interest (28%); (iv) the granularity of tests spans from the system level (29%) to system components/elements units (28%); (v) most of the approaches (36.8%) perform the performance regression testing in continuous time; (vi) about the programming languages mostly used in studies, we can point out Java (20.6%) and C/C++ (8.8%) as the most preferred languages; (vii) 57% of the studies proposes the development of new tools to perform their experiments (but 15 tools only have a valid website associated). Moreover, regarding how the approaches have been evaluated, the main findings are: (i) 58% of the studies revolve their evaluation around open source projects; (ii) engineering research is targeting 55% of the studies according to the empirical standards; (iii) most studies (68%) do not perform any comparison of their approach with other techniques/tools; and (iv) 45.6% of the studies identify the threats and discuss actions for mitigation.

The analysis conducted in this study enabled us to further elicit and discuss the several research opportunities, shedding light on potential future directions. We think that these systematic mapping results can help to identify a body of knowledge to support future research and software development activities by researchers and practitioners, respectively, in the field of performance regression testing.

Acknowledgements

We would like to thank the anonymous reviewers for their precious and constructive feedback that helped to improve the quality of the manuscript. This work has been partially funded by the MUR Department of Excellence 2023 - 2027 for GSSI, the MUR project PRIN 20228FT78M DREAM, and PNRR ECS00000041 VITALITY. Andre T. Endo is partially supported by grant #2023/00577-8, São Paulo Research Foundation (FAPESP).

References

[1] M. Harman, P. O'Hearn, From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis, in: IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), 2018, pp. 1–23.

[2] C. Jones, O. Bonsignour, The economics of software quality, Addison-Wesley Professional, 2011.

[3] J. Chen, W. Shang, E. Shihab, Perfjit: Test-level just-in-time prediction for performance regression introducing commits, IEEE Transactions on Software Engineering 48 (2020) 1529–1544.

[4] M. Pradel, M. Huggler, T. R. Gross, Performance regression testing of concurrent classes, International Symposium on Software Testing and Analysis, ISSTA 2014 (2014) 13 – 25. doi:10.1145/2610384.2610393.

[5] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, P. Flora, Mining performance regression testing repositories for automated performance analysis, in: International Conference on Quality Software, 2010, pp. 32–41.

[6] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and software technology 64 (2015) 1–18.

[7] N. Alshahwan, M. Harman, A. Marginean, Software testing research challenges: An industrial perspective, in: IEEE Conference on Software Testing, Verification and Validation (ICST), 2023, pp. 1–10.

[8] B. A. Kitchenham, S. Charters, Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007-001, Keele University and Durham University, UK, 2007.

[9] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: International Conference on Evaluation and Assessment in Software Engineering (EASE), 2008, pp. 1–10.

[10] A. Mathur, Foundations of software testing, seventh impression, 2012.

[11] M. Delamaro, J. Maldonado, M. Jino, Introdução ao teste de software. rio de janeiro, rj, 2007.

[12] W. E. Wong, J. R. Horgan, S. London, H. Agrawal, A study of effective regression testing in practice, in: PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering, IEEE, 1997, pp. 264–274.

[13] D. Nir, S. Tyszberowicz, A. Yehudai, Locating regression bugs, in: Hardware and Software: Verification and Testing: Third International Haifa Verification Conference, HVC 2007, Haifa, Israel, October 23-25, 2007. Proceedings 3, Springer, 2008, pp. 218–234.

[14] J. Chen, W. Shang, An exploratory study of performance regression introducing code changes, in: 2017 IEEE international conference on software maintenance and evolution (icsme), IEEE, 2017, pp. 341–352.

[15] B. Ba-Quttayyan, H. Mohd, F. Baharom, Regression testing systematic literature review–a preliminary analysis, International Journal of Engineering and Technology 7 (2018) 418–424.

[16] K. Hannigan, An Empirical Evaluation of the Indicators for Performance Regression Test Selection, Phd thesis, Rochester Institute of Technology, 2018.

[17] D. M. Kaushik, P. Fageria, Performance testing tools: A comparative study, International Journal of Innovative Science, Engineering & Technology 1 (2014) 264–267.

[18] B. M. Napoleão, K. R. Felizardo, F. d. Souza, F. Petrillo, S. Hallé, N. L. Vijaykumar, E. Y. Nakagawa, Establishing a search string to detect secondary studies in software engineering, in: 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2021, pp. 9–16. doi:10.1109/SEAA53835.2021.00010.

[19] V. Garousi, M. Felderer, M. V. Mäntylä, Guidelines for including grey literature and conducting multivocal literature reviews in software engineering, Information and Software Technology 106 (2019) 101–121.

[20] P. Arora, R. A. Bhatia, Systematic review of agent-based test case generation for regression testing, Computer Engineering and Computer Science 43 (2018) 447–470.

[21] X. Han, T. Yu, G. Yan, A systematic mapping study of software performance research, Software: Practice and Experience 53 (2023) 1249–1270.

[22] R. Kazmi, D. N. A. Jawawi, R. Mohamad, I. Ghani, Effective regression test case selection: A systematic literature review, ACM Comput. Surv. 50 (2017).

[23] E. Engström, P. Runeson, M. Skoglund, A systematic review on regression test selection techniques, Information and Software Technology 52 (2010) 14–30.

[24] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Softw. Test. Verification Reliab. 22 (2012) 67–120.

[25] N. B. Ali, E. Engström, M. Taromirad, M. R. Mousavi, N. M. Minhas, D. Helgesson, S. Kunze, M. Varshosaz, On the search for industry-relevant regression testing research, Empirical Software Engineering 24 (2019) 2020–2055.

[26] R. Greca, B. Miranda, A. Bertolino, State of practical applicability of regression testing research: A live systematic literature review, ACM Comput. Surv. 55 (2023).

[27] K. R. Felizardo, E. Mendes, M. Kalinowski, E. F. Souza, N. L. Vijaykumar, Using forward snowballing to update systematic reviews in software engineering, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, New York, NY, USA, 2016.

[28] C. Wohlin, M. Kalinowski, K. Romero Felizardo, E. Mendes, Successful combination of database search and snowballing for identification of primary studies in systematic literature studies, Information and Software Technology 147 (2022) 106908.

[29] D. Maplesden, E. Tempero, J. Hosking, J. Grundy, Performance analysis for object-oriented software: A systematic mapping, IEEE Transaction on Software Engineering 41 (2015) 691–710.

[30] H. Zhang, B. Muhammad, T. Paolo, Identifying relevant studies in software engineering, Information and Software Technology 53 (2011) 625–637.

[31] 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, EASE '14, 2014. URL: https://doi.org/10.1145/2601248.2601268. doi:10.1145/2601248.2601268.

[32] A. da Silva, K. Felizardo, de Souza Erica Ferreira, N. Vijaykumar, E. Nakagawa, Evaluating electronic databases for forward snowballing application to support secondary studies updates – emergent results, in: Proceedings 32nd Brazilian Sympo- sium on Software Engineering (SBES' 18), 2018.

[33] C. Wohlin, E. Mendes, K. R. Felizardo, M. Kalinowski, Guidelines for the search strategy to update systematic literature reviews in software engineering, Information and Software Technology 127 (2020) 106366.

[34] J. Chen, W. Shang, E. Shihab, Perfjit: Test-level just-in-time prediction for performance regression introducing commits, IEEE Transactions on Software Engineering 48 (2022) 1529–1544.

[35] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, C. Sporea, A. Toma, S. Sajedi, Locating performance regression root causes in the field operations of web-based systems: An experience report, IEEE Transactions on Software Engineering 48 (2022) 4986–5006.

[36] R. Jalan, A. Kejariwal, Trin-trin: Who's calling? a pin-based dynamic call graph extraction framework, International Journal of Parallel Programming 40 (2012) 410–442.

[37] D. Lee, S. K. Cha, A. H. Lee, A performance anomaly detection and analysis framework for dbms development, IEEE Transactions on Knowledge and Data Engineering 24 (2012) 1345–1360.

[38] F. S. Ocariza, Jr, On the effectiveness of bisection in performance regression localization, Empirical Software Engineering 27 (2022) 95.

[39] I. Jimenez, N. Watkins, M. Sevilla, J. Lofstead, C. Maltzahn, Quiho: Automated performance regression testing using inferred resource utilization profiles, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, pp. 273–284.

[40] L. Breiman, Random forests, Machine learning Vol. 45 (2001) 5–32.

[41] M. Alam, J. Gottschlich, N. Tatbul, J. S. Turek, T. Mattson, A. Muzahid, A zero-positive learning approach for diagnosing software performance regressions, Advances in Neural Information Processing Systems 32 (2019).

[42] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, W. Shang, Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report, in: Proceedings of the 13th International Conference on Mining Software Repositories, 2016, p. 1–12. doi:10.1145/2901739.2901774.

[43] J. Jung, H. Hu, J. Arulraj, T. Kim, W. Kang, Apollo: Automatic detection and diagnosis of performance regressions in database systems, in: Proceedings of the VLDB Endowment, volume 13, 2020, p. 57 – 70. doi:10.14778/3357377.3357382.

[44] X. Liu, Q. Zhou, J. Arulraj, A. Orso, Automatic detection of performance bugs in database systems using equivalent queries, in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 225–236.

[45] S. Mühlbauer, S. Apel, N. Siegmund, Accurate modeling of performance histories for evolving software systems, in: 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 640–652. doi:10.1109/ASE.2019.00065.

[46] C. Laaber, H. C. Gall, P. Leitner, Applying test case prioritization to software microbenchmarks, Empirical Software Engineering 26 (2021).

[47] A. Hindle, Green mining: Investigating power consumption across versions, in: 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 1301–1304. doi:10.1109/ICSE.2012.6227094.

[48] T. Yoshimura, R. Nakazawa, T. Chiba, Imagejockey: A framework for container performance engineering, in: 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), 2020, pp. 238–247. doi:10.1109/CLOUD49709.2020.00043.

[49] A. B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, P. F. Sweeney, Perphecy: Performance regression test selection made simple but effective, in: International Conference on Software Testing, Verification and Validation (ICST), 2017, pp. 103–113.

[50] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, S. Sajedi, Using black-box performance models to detect performance regressions under varying workloads: an empirical study, Empirical Software Engineering 25 (2020) 4130–4160.

[51] Q. Luo, D. Poshyvanyk, M. Grechanik, Mining performance regression inducing code changes in evolving software, in: Proceedings of the

13th International Conference on Mining Software Repositories, 2016, p. 25–36. doi:10.1145/2901739.2901765.

[52] A. Bhattacharyya, C. Amza, Pret: A tool for automatic phase-based regression testing, in: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2018, pp. 284–289.

[53] J. Wienke, S. Wrede, Performance regression testing and run-time verification of components in robotics systems, Advanced Robotics 31 (2017) 1177–1192. doi:10.1080/01691864.2017.1395360.

[54] P. Huang, X. Ma, D. Shen, Y. Zhou, Performance regression testing target prioritization via performance risk analysis, in: International Conference on Software Engineering, 2014, p. 60 – 71. doi:10.1145/2568225.2568232.

[55] T. Fiedor, J. Pavela, A. Rogalewicz, T. Vojnar, Perun: Performance version system, in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2022, pp. 499–503. doi:10.1109/ICSME55016.2022.00067.

[56] D. Alshoaibi, M. W. Mkaouer, A. Ouni, A. Wahaishi, T. Desell, M. Soui, Search-based detection of code changes introducing performance regression, Swarm and Evolutionary Computation 73 (2022) 101101.

[57] F. Hewson, J. Dietrich, S. Marsland, Performance regression testing on the java virtual machine using statistical test oracles, in: 2015 24th Australasian Software Engineering Conference, IEEE, 2015, pp. 18–27.

[58] D.-H. Lee, J.-J. Park, Square-wave like performance change detection using spc charts and anfis, in: IT Convergence and Security 2012, 2013, pp. 1097–1104.

[59] J. P. Sandoval Alcocer, A. Bergel, M. T. Valente, Prioritizing versions for performance regression testing: The pharo case, Science of Computer Programming 191 (2020) 102415. doi:https://doi.org/10.1016/j.scico.2020.102415.

[60] M. Grechanik, C. Fu, Q. Xie, Automatically finding performance problems with feedback-directed learning software testing, in: International Conference on Software Engineering (ICSE), 2012, pp. 156–166.

[61] D. ALShoaibi, H. Gupta, M. Mendelson, I. Jenhani, A. B. Mrad, M. W. Mkaouer, Learning to characterize performance regression introducing code changes, in: ACM/SIGAPP Symposium on Applied Computing (SAC), 2022, pp. 1590–1597.

[62] P. Stankiewicz, M. Kobilarov, Identifying performance regression conditions for testing evaluation of autonomous systems, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 4276–4281. doi:10.1109/IROS51168.2021.9636004.

[63] S. Ghaith, M. Wang, P. Perry, Z. M. Jiang, P. O'Sullivan, J. Murphy, Anomaly detection in performance regression testing by transaction profile estimation, Software Testing Verification and Reliability 26 (2016) 4 – 39. doi:10.1002/stvr.1573.

[64] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, P. Flora, An industrial case study on the automated detection of performance regressions in heterogeneous environments, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 159–168. doi:10.1109/ICSE.2015.144.

[65] W. Shang, A. E. Hassan, M. Nasser, P. Flora, Automated detection of performance regressions using regression models on clustered performance counters, in: ICPE 2015 - Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015, p. 15 – 26. doi:10.1145/2668930.2688052.

[66] T. H. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser, P. Flora, Automated detection of performance regressions using statistical process control techniques, in: ICPE'12 - Proceedings of the 3rd Joint WOSP/SIPEW International Conference on Performance Engineering, 2012, p. 299 – 310. doi:10.1145/2188286.2188344.

[67] T. H. Nguyen, M. Nagappan, A. E. Hassan, M. Nasser, P. Flora, An industrial case study of automatically identifying performance

regression-causes, in: 11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings, 2014, p. 232 – 241. doi:10.1145/2597073.2597092.

[68] M. Fagerström, E. E. Ismail, G. Liebel, R. Guliani, F. Larsson, K. Nordling, E. Knauss, P. Pelliccione, Verdict machinery: On the need to automatically make sense of test results, in: ISSTA 2016 - Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016, p. 225 – 234. doi:10.1145/2931037.2931064.

[69] M. Gómez, R. Rouvoy, B. Adams, L. Seinturier, Mining test repositories for automatic detection of ui performance regressions in android apps, Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016 (2016) 13 – 24. doi:10.1145/2901739.2901747.

[70] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanović, A. Van Hoorn, Microservices: A performance tester's dream or nightmare?, ICPE 2020 - Proceedings of the ACM/SPEC International Conference on Performance Engineering (2020) 138 – 149. doi:10.1145/3358960.3379124.

[71] G. Jin, L. Song, X. Shi, J. Scherpelz, S. Lu, Understanding and detecting real-world performance bugs, ACM SIGPLAN Notices 47 (2012) 77 – 87. doi:10.1145/2345156.2254075.

[72] D. Daly, Creating a virtuous cycle in performance testing at mongodb, in: Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2021, p. 33–41.

[73] J. Wu, J. Dong, R. Fang, W. Zhang, W. Wang, D. Zuo, Fadatest: Fast and adaptive performance regression testing of dynamic binary translation systems, in: Proceedings - International Conference on Software Engineering, 2022, p. 896 – 908.

[74] Y. Zhang, X. Xie, Y. Li, Y. Lin, S. Chen, Y. Liu, X. Li, Demystifying performance regressions in string solvers, IEEE Transactions on Software Engineering 49 (2023) 947–961.

[75] D. G. Reichelt, S. Kühne, How to detect performance changes in software history: Performance analysis of software system versions,

ICPE 2018 - Companion of the 2018 ACM/SPEC International Conference on Performance Engineering 2018-January (2018) 183 – 188. doi:10.1145/3185768.3186404.

[76] S. Mühlbauer, S. Apel, N. Siegmund, Identifying software performance changes across variants and versions, in: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, 2020, p. 611–622.

[77] S. Mostafa, X. Wang, T. Xie, Perfranker: Prioritization of performance regression tests for collection-intensive sotware, in: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2017, p. 23 – 34. doi:10.1145/3092703.3092725.

[78] Z. Ding, J. Chen, W. Shang, Towards the use of the readily available tests from the release pipeline as performance tests. arewe there yet, in: International Conference on Software Engineering, 2020, p. 1435 – 1446. doi:10.1145/3377811.3380351.

[79] C. Bezemer, E. Milon, A. Zaidman, J. Pouwelse, Detecting and analyzing i/o performance regressions, Journal of Software: Evolution and Process 26 (2014) 1193–1212. doi:https://doi.org/10.1002/smr.1657.

[80] A. Van Lamsweerde, Requirements engineering in the year 00: A research perspective, in: Proceedings of the International Conference on Software Engineering (ICSE), 2000, pp. 5–19.

[81] J. M. Stecklein, J. Dabney, B. Dick, B. Haskins, R. Lovell, G. Moroney, Error cost escalation through the project life cycle, in: 14th Annual International Symposium, JSC-CN-8435, 2004.

[82] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, Non-functional requirements in software engineering, volume 5, Springer Science & Business Media, 2012.

[83] F. Aquilani, S. Balsamo, P. Inverardi, Performance analysis at the software architectural design level, Performance Evaluation 45 (2001) 147–178.

[84] Y. Yang, X. Xia, D. Lo, T. Bi, J. Grundy, X. Yang, Predictive models in software engineering: Challenges and opportunities, ACM Transactions on Software Engineering and Methodology (TOSEM) 31 (2022) 1–72.

[85] M.-W. Wu, Y.-D. Lin, Open source software development: An overview, Computer 34 (2001) 33–38.

[86] K. Gallaba, Improving the robustness and efficiency of continuous integration and deployment, in: International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 619–623.

[87] D. Daly, W. Brown, H. Ingo, J. O'Leary, D. Bradford, The use of change point detection to identify software performance regressions in a continuous integration system, in: ICPE 2020 - Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2020, p. 67 – 75. doi:10.1145/3358960.3375791.

[88] J. V. Bukowski, Modeling and analyzing the effects of periodic inspection on the performance of safety-critical systems, IEEE Transactions on Reliability 50 (2001) 321–329.

[89] T. Laurent, P. Arcaini, C. Trubiani, A. Ventresque, Mutation-based analysis of queueing network performance models, J. Syst. Softw. 191 (2022) 111385.

[90] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, S. Wagner, Software engineering for ai-based systems: a survey, ACM Transactions on Software Engineering and Methodology (TOSEM) 31 (2022) 1–59.

[91] L. Song, S. Lu, Statistical debugging for real-world performance problems, ACM SIGPLAN Notices 49 (2014) 561–578.

[92] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: 23rd Asia-Pacific Software Engineering Conferencees, 2016, pp. 153–160.

[93] D. I. Sjøberg, G. R. Bergersen, Construct validity in software engineering, IEEE Transactions on Software Engineering 49 (2023) 1374–1396.

[94] D. I. Sjøberg, G. R. Bergersen, Improving the reporting of threats to construct validity, in: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, 2023, pp. 205–209.

[95] K.-T. Rehmann, C. Seo, D. Hwang, B. T. Truong, A. Boehm, D. H. Lee, Performance monitoring in sap hana's continuous integration process, SIGMETRICS Perform. Eval. Rev. 43 (2016) 43–52.

[96] M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, Continuous validation of performance test workloads, Automated Software Engineering 24 (2017) 189 – 231. doi:10.1007/s10515-016-0196-8.

[97] T. Yu, M. Pradel, Pinpointing and repairing performance bottlenecks in concurrent programs, Empirical Software Engineering 23 (2018) 3034 – 3071. doi:10.1007/s10664-017-9578-1.

[98] A. Swearngin, M. B. Cohen, B. E. John, R. K. E. Bellamy, Human performance regression testing, in: 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 152–161. doi:10.1109/ICSE.2013.6606561.

[99] S. Eid, S. Makady, M. Ismail, Detecting software performance problems using source code analysis techniques, Egyptian Informatics Journal 21 (2020) 219–229. doi:https://doi.org/10.1016/j.eij.2020.02.002.

[100] V. Horký, F. Haas, J. Kotrč, M. Lacina, P. Tůma, Performance regression unit testing: A case study, in: Computer Performance Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 149–163.

[101] S. Merchant, G. Prabhakar, Tool for performance tuning and regression analyses of hpc systems and applications, in: 2012 19th International Conference on High Performance Computing, 2012, pp. 1–6. doi:10.1109/HiPC.2012.6507528.

[102] E. C. Barboza, S. Jacob, M. Ketkar, M. Kishinevsky, P. Gratz, J. Hu, Automatic microprocessor performance bug detection, in: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 545–556. doi:10.1109/HPCA51647.2021.00053.

[103] D. G. Reichelt, S. Kühne, Better early than never: Performance test acceleration by regression test selection, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, 2018, p. 127–130. doi:10.1145/3185768.3186289.

[104] Y. Zhao, L. Xiao, X. Wang, Z. Chen, B. Chen, Y. Liu, Butterfly space: An architectural approach for investigating performance issues, in: IEEE International Conference on Software Architecture (ICSA), 2020, pp. 202–213. doi:10.1109/ICSA47634.2020.00027.

[105] D. L. D. L. J. W. Kim-Thomas RehmannKim, Alexander Böhm, Continuous performance testing for sap hana, in: First International Workshop on Reliable Data Services and Systems (RDSS), 2014.

[106] S. Eismann, D. E. Costa, L. Liao, C.-P. Bezemer, W. Shang, A. van Hoorn, S. Kounev, A case study on the stability of performance tests for serverless applications, Journal of Systems and Software 189 (2022) 111294. doi:https://doi.org/10.1016/j.jss.2022.111294.

[107] M. Lindon, C. Sanden, V. Shirikian, Rapid regression detection in software deployments through sequential testing, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2022, p. 3336 – 3346. doi:10.1145/3534678.3539099.

[108] R. Ramakrishnan, A. Kaur, Technique for detecting early-warning signals of performance deterioration in large scale software systems, in: ICPE 2017 - Proceedings of the 2017 ACM/SPEC International Conference on Performance Engineering, 2017, p. 213 – 222. doi:10.1145/3030207.3044533.

[109] R. Padhye, K. Sen, Travioli: A dynamic analysis for detecting data-structure traversals, in: IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 473–483. doi:10.1109/ICSE.2017.50.

[110] J. Cito, D. Suljoti, P. Leitner, S. Dustdar, Identifying root causes of web performance degradation using changepoint analysis, in: Web Engineering, Springer International Publishing, Cham, 2014, pp. 181–199.

[111] O. Javed, P. Singh, G. Reger, S. Toor, To test, or not to test: A proactive approach for deciding complete performance test initiation, in: 2022 IEEE International Conference on Big Data (Big Data), 2022, pp. 4758–4767. doi:10.1109/BigData55660.2022.10020543.

[112] H. Ingo, D. Daly, Automated system performance testing at mongodb, in: Proceedings of the Workshop on Testing Database Systems, DBTest 2020, 2020. doi:10.1145/3395032.3395323.

[113] C.-P. Bezemer, J. Pouwelse, B. Gregg, Understanding software performance regressions using differential flame graphs, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 535–539. doi:10.1109/SANER.2015.7081872.

[114] M. Grambow, D. Kovalev, C. Laaber, P. Leitner, D. Bermbach, Using microbenchmark suites to detect application performance changes, IEEE Transactions on Cloud Computing 11 (2023) 2575–2590. doi:10.1109/TCC.2022.3217947.

Appendix A. Included studies.

The following are presented the studies (S) included in this systematic mapping.

Table A.12: Final studies selected.

ID	Citation	ID	Citation	ID	Citation
S1	[34]	S24	[69]	S47	[95]
S2	[61]	S25	[51]	S48	[71]
S3	[56]	S26	[63]	S49	[96]
S4	[73]	S27	[64]	S50	[97]
S5	[74]	S28	[57]	S51	[78]
S6	[46]	S29	[65]	S52	[60]
S7	[72]	S30	[54]	S53	[75]
S8	[62]	S31	[67]	S54	[70]
S9	[35]	S32	[98]	S55	[79]
S10	[99]	S33	[58]	S56	[38]
S11	[48]	S34	[100]	S57	[55]
S12	[50]	S35	[101]	S58	[44]
S13	[59]	S36	[36]	S59	[102]
S14	[87]	S37	[47]	S60	[103]
S15	[43]	S38	[37]	S61	[104]
S16	[41]	S39	[66]	S62	[105]
S17	[52]	S40	[4]	S63	[106]
S18	[39]	S41	[53]	S64	[107]
S19	[14]	S42	[42]	S65	[108]
S20	[109]	S43	[110]	S66	[111]
S21	[77]	S44	[112]	S67	[113]
S22	[49]	S45	[45]	S68	[114]
S23	[68]	S46	[76]		

Appendix B. Mapping of individual studies to categories.

Tables B.13, B.14, B.15, B.16, B.17, B.18, B.19, and B.20 present the mappings of studies to the categorizations presented in Section 5.

Table B.13: Technique cluster (RQ2).

Technique cluster	**Studies**
Profiling	S11, S15, S20, S29, S35, S36, S38, S39, S40, S41, S42, S44, S50, S54, S55, S56, S63, S67, S68
Statistics	S4, S7, S13, S14, S18, S19, S22, S24, S28, S45, S47, S64
Machine Learning	S1, S2, S9, S12, S16, S17, S27, S31, S49, S46, S52, S59
Modeling	S26, S32, S57, S61, S65
Code Analysis	S37, S53, S60, S66
Simulation	S8, S43, S62
Evolutionary Algorithm	S3, S10, S25
Test Case Prioritization	S6, S21, S30
Logic	S33, S34
Code Mutation	S5, S58
Scraping	S48, S51
Interview	S23

Table B.14: Application Domain (RQ2).

Application Domain	**Studies**
Not specified	S1, S2, S3, S10, S14, S20, S22, S23, S27, S30, S31, S32, S33, S37, S40, S56, S57
Web-based systems	S9, S25, S28, S29, S42, S43, S52, S63, S64, S66
DBMS	S7, S15, S17, S38, S44, S47, S58, S62, S68
Libraries	S18, S19, S21, S34, S45, S53, S55, S60
Open source systems	S13, S46, S48, S49, S51, S61

Continues

Table B.14: Application Domain (RQ2)

Application Domain	**Studies**
Large-scale software systems	S12, S26, S39, S50, S65
Parallel and Distributed Computing	S16, S36, S67
Microservices	S11, S54
Software microbenchmark	S4, S6
String solvers	S5
Robotic Systems	S41
Autonomous system	S8
Mobile apps	S24
Microprocessors	S59
High Performance Computing	S35

Table B.15: Granularity of tests (RQ2).

Granularity of tests	**Studies**
Not specified	S1, S2, S3, S4, S5, S8, S9, S11, S12, S13, S14, S22, S23, S42, S43, S44, S45, S46, S56, S57, S63, S64, S65
Unit Tests	S6, S10, S19, S20, S21, S31, S32, S34, S35, S36, S38, S48, S51, S53, S55, S58, S60, S61, S62
System testing	S16, S25, S26, S27, S28, S29, S30, S33, S37, S39, S40, S47, S49, S50, S52, S54, S59, S66, S67, S68
Database Testing	S7, S15, S17, S18
User Interface testing	S24
Integration testing	S41

Table B.16: Frequency of testing (RQ2).

Frequency of testing	**Studies**
Continuous Time	S7, S9, S11, S17, S26, S27, S28, S29, S31, S32, S36, S40, S42, S43, S44, S49, S50, S52, S58, S59, S60, S61, S62, S64, S65
Commits	S14, S19, S21, S22, S30, S34, S37, S45, S46, S47, S51, S53, S55, S66, S67, S68
System versions	S1, S2, S3, S10, S12, S13, S15, S16, S18, S25, S39, S41, S48, S56, S57
After each test run	S4, S5, S6, S8,S23, S24, S63
Not specified	S20, S35, S54
Sample period	S33
Package building	S38

Table B.17: Target programming languages (RQ2).

Target programming languages	**Studies**
Java	S1, S6, S10, S17, S25, S28, S34, S40, S41, S42, S52, S53, S60, S61
C/C++	S3, S4, S30, S36, S50, S57
Python	S11, S14, S55
SQL	S58, S62
Pharo	S13
JavaScript	S20

Table B.18: Types of subject systems used (RQ3).

Type	Studies
Open source projects only	S3, S1, S2, S4, S5, S6, S7, S10, S42, S44, S13, S14, S15, S45, S46, S63, S17, S19, S20, S21, S22, S24, S25, S56, S57, S40, S30, S32, S48, S51, S58, S60, S61, S34, S37, S53, S54, S55, S67, S68
Industrial projects only	S8, S64, S65, S47, S41, S23, S62, S38
Open source and industrial projects	S9, S12, S26, S27, S29, S31, S49, S52, S39, S66
Benchmark suites only	S43, S11, S28, S59, S35
Open source projects and Benchmark suites	S16, S18, S50
Open source and industrial projects, and Benchmark suites	S36
Industrial projects and Benchmark suites	-
None	S33

Table B.19: Most used open source projects (RQ3).

Project Name	Studies
Dell DVD Store	S27, S29, S31, S39
Hadoop	S1, S19, S49, S51
JPetStore	S25, S26, S27, S52
MongoDB	S7, S44, S14, S22
MySQL	S16, S30, S48, S50
Apache Commons-IO	S10, S60, S53
Cassandra	S1, S17, S51
Firefox	S48, S50, S37
Git	S3, S2, S22
OpenMRS	S2, S42, S12

Continues

Table B.19: Most used open source projects (RQ3).

Project Name	**Studies**
PostgreSQL	S15, S30, S58

Table B.20: Threats to validity (RQ3).

Does it include threats to validity?	**Studies**
Yes	S3, S1, S5, S6, S9, S10, S42, S43, S65, S12, S13, S45, S46, S63, S18, S41, S19, S20, S21, S22, S23, S24, S56, S29, S30, S49, S50, S51, S52, S54, S55
No	S2, S4, S7, S8, S44, S64, S11, S14, S47, S16, S17, S57, S26, S27, S28, S40, S31, S32, S48, S59, S60, S33, S34, S35, S38, S66
Partially	S15, S25, S58, S61, S62, S36, S37, S39, S53, S67, S68