The Journal of Systems and Software 207 (2024) 111833

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Architectural support for software performance in continuous software engineering: A systematic mapping study[☆]

Romina Eramo [a,*], Michele Tucci [b], Daniele Di Pompeo [b], Vittorio Cortellessa [b], Antinisca Di Marco [b], Davide Taibi [c,d]

[a] University of Teramo, Teramo, Italy
[b] University of L'Aquila, L'Aquila, Italy
[c] University of Oulu, Oulu, Finland
[d] Tampere University, Tampere, Finland

ARTICLE INFO

Keywords:
Software architecture
Software performance
Continuous software engineering
DevOps

ABSTRACT

The continuous software engineering paradigm is gaining popularity in modern development practices, where the interleaving of design and runtime activities is induced by the continuous evolution of software systems. In this context, performance assessment is not easy, but recent studies have shown that architectural models evolving with the software can support this goal. In this paper, we present a mapping study aimed at classifying existing scientific contributions that deal with the architectural support for performance-targeted continuous software engineering. We have applied the systematic mapping methodology to an initial set of 215 potentially relevant papers and selected 66 primary studies that we have analyzed to characterize and classify the current state of research. This classification helps to focus on the main aspects that are being considered in this domain and, mostly, on the emerging findings and implications for future research.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board. (see [https://www.sciencedirect.com/science/article/pii/S0164121221002168] for an example for where to place the statement and how to format it).

1. Introduction

Continuous software engineering (CSE) is a promising software process that interleaves business strategy (i.e., requirement engineering), development, and operations on a continuum. It aims to produce a better software product and create more successful implementations that satisfy the relevant requirements and constraints. Similarly, the recent emphasis on DevOps recognizes that the integration between software development and its operational distribution must be continuous. DevOps improves end-to-end collaboration between the stakeholders, development, and operations teams. In addition, they have been successfully employed in disciplines such as security and testing. Software performance (SP) is an essential quality aspect for the adoption and success of a software system. Researchers and industry practitioners have identified the importance of integrating performance engineering practices in continuous development processes in a timely and efficient way (Ferme and Pautasso, 2017). However, current software performance engineering methods are not tailored for environments using CSE processes and practices are lagging (Bezemer et al., 2018; Kudrjavets et al., 2022).

Although SP is a non-functional property related to the platform on which the software is deployed, performance assessment, in the last two decades, has been mainly estimated at the design level through methods, such as software architecture (SA) (Brosig et al., 2011; Cortellessa et al., 2011; Martens et al., 2010). SAs can be transformed into performance models, whose indices can be exploited to compare SA alternatives. Such a design-time performance assessment does not extensively consider several aspects of the target platform characteristics. However, these early-stage comparative analyses that show differences evident in the alternative results certainly support architects to make decisions with an enhanced view of their performance effects.

The rise of the continuous engineering paradigm has substantially changed in the last decade of the software process. More often, it is nowadays required that software engineering follows a continuous loop between the running code and design models such that these two sides

☆ Editor: Uwe Zdun.
* Corresponding author.
E-mail addresses: reramo@unite.it (R. Eramo), michele.tucci@univaq.it (M. Tucci), daniele.dipompeo@univaq.it (D. Di Pompeo), vittorio.cortellessa@univaq.it (V. Cortellessa), antinisca.dimarco@univaq.it (A. Di Marco), davide.taibi@oulu.fi (D. Taibi).

https://doi.org/10.1016/j.jss.2023.111833
Received 14 April 2022; Received in revised form 7 April 2023; Accepted 2 September 2023
Available online 29 September 2023

of the process can reciprocally feed each other (Fitzgerald and Stol, 2017). For example, runtime data can be collected from execution traces to feed software models. Software models are then aimed at checking either functional or non-functional properties. The analysis of software models in the context of incoming execution scenarios can suggest just-in-time refactoring/adaptation actions that keep the software behavior acceptable when these scenarios occur (Mazkatli et al., 2020; Spinner et al., 2016, 2019). In the context of CSE processes, architectural models appear to have gained relevance, among others, for supporting performance-related decisions (Arcelli et al., 2019; Cortellessa et al., 2022).

Despite rising interest in embracing the continuous architecting approaches and performance engineering practices, there has been a little consensus in the literature on the appropriateness of different performance engineering techniques that can be used in a continuous engineering process. A limited number of studies that consider continuous engineering in some specific aspects of self-adaptive systems and microservices have been published (Koziolek, 2010; Becker et al., 2012; Pahl and Jamshidi, 2016; Taibi et al., 2019; Jabbari et al., 2016).

However, current CSE and DevOps practices focus on rapid delivery, minimizing time to release for new features, mitigating risks, driving new efficiencies, and establishing a continuous delivery pipeline. Efficient and automated performance engineering tools are critical and pose relevant challenges in accomplishing this mission. Thus, there is still a need for a study that systematically investigates all the key publications on this topic and identifies possible performance engineering techniques applicable to continuous engineering processes.

In this study, we conduct a mapping study of the existing literature (Petersen et al., 2015) to investigate the contributions of the scientific community to architectural support for SP within CSE. Furthermore, the study aims to characterize and classify the current research scenario to better structure our understanding of the topic and identify research directions worth investigating in this domain soon.

The main contributions of this study include:

- A reusable framework for classifying, comparing, and evaluating solutions, methods, and techniques specific to architectural support for software performance in continuous software engineering;
- A systematic map of the state of research in the domain of architectural support for SP in CSE in terms of the performance areas, domains, addressed problems, and adopted instruments;
- A discussion of the emerging trends, gaps in the literature, and their implications for future research.

The remainder of this paper is organized as follows: In Section 2, we provide a background review and compare the existing literature. In Section 3, we define our target question and illustrate the process that we have adopted to conduct the mapping study; In Sections 4 and 8, we describe and analyze the results obtained to answer our target questions. In Section 9, we discuss the threats to validity of our study, and finally, Section 10 presents the concluding remarks.

2. Background and related work

This section provides some background information and presents the synergies among the main concepts involved; other studies on related topics are also presented in this section.

2.1. Main concepts

Continuous Software Engineering (CSE). This refers to the capability to develop, release, and learn from software in rapid parallel cycles. This includes determining new functionality to build, evolve and refactor the architecture, develop the functionality, validate it, release it to customers and collect experimental feedback from the customers to inform the next cycle of development (Tichy et al., 2017).

The definition of CSE is prone to interpretations and is often used in conjunction with other continuous activities that emerge during the entire software (engineering) lifecycle (Fitzgerald and Stol, 2017). In particular, the activities considered in the *development* phase are: continuous integration, continuous deployment/release, continuous delivery, and continuous verification/testing. Whereas, the *operation* phase concerns the end of the process, where handover of the release is initiated; in this phase, particular attention is devoted to the continuous use of these systems, after the initial adoption, as well as continuous monitoring, to observe and detect compliance issues and risks. The most recent stand out of the *DevOps* (Ebert et al., 2016) practices, which promote the integration between development and operations, confirms that these areas are closely interact to achieve CSE. Finally, a closer and continuous linkage between business management and software development functions is also necessary to benefit activities such as business planning; the *BizDev* (Fitzgerald and Stol, 2017) phenomenon complements DevOps, integrating business management with software development and operations functions.

Software Performance (SP). This represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed at meeting performance requirements (Woodside et al., 2007). This field focuses on the quantitative evaluation of modern software systems (e.g., data-intensive, autonomous, distributed, adaptive, and embedded systems) and trade-offs between performance and other quality of service (QoS) attributes (e.g., security, reliability, and availability). In the last few decades, numerous performance engineering methods, methodologies, and techniques have been developed for system evaluation (Merseguer et al., 2017).

SP assessment is a crucial task in software engineering to ensure that a new software release does not impair the user-perceived performance. Performance degradation can occur in various forms, such as high response time latency, low throughput, and excessive resource utilization. Although these arguments would suggest that performance should be assessed on every change, recent studies on continuous engineering shows that it is not standard practice yet (Laaber, 2019).

Software Architecture (SA). The SA of a software system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them (Bass et al., 2003). SA is often the first design artifact to represent decisions on how the requirements of all types are to be achieved. It shows the correspondence between the requirements and the constructed system, thereby providing a rationale for the design decisions (Hasselbring, 2018). The design of the overall system structure, particularly in large and complex systems, is an essential factor. For instance, performance depends largely upon the complexity of the required communication, coordination, or cooperation among the different components, particularly in complex distributed systems.

The need for SA evaluation is based on the realization that software development, similar to all engineering disciplines, is a process of continuous modeling and refinement. Detecting architectural problems before the bulk of the development work is completed allows re-architecting activities to take place in due time. At the same time, tuning activities enhance and maintain the SP during the software lifetime (Del Rosso, 2006).

Synergies between CSE, SP, and SA. CSE involves several challenges in terms of SA evolution and the detection/resolution of problems related to software quality attributes, such as performance (Fitzgerald and Stol, 2017). Software development, in practice, concerns the continuous evolution of software, primarily owing to new incoming requirements. It is possible that SA is not adequate to embed new required functionalities, thus imposing a heavy and complex software evolution.

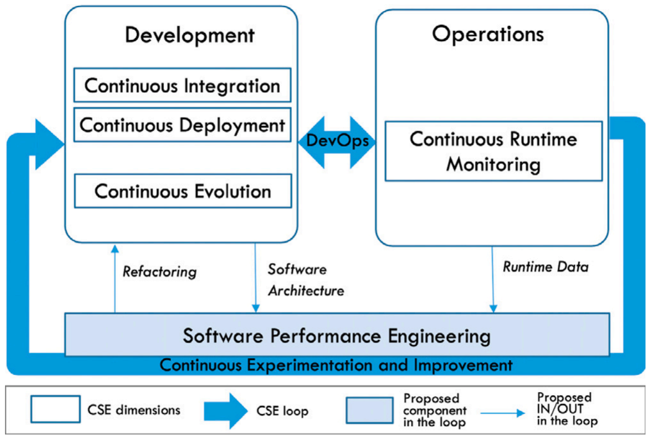

Development
Continuous Integration
Continuous Deployment
Continuous Evolution
DevOps
Operations
Continuous Runtime Monitoring
Refactoring
Software Architecture
Runtime Data
Software Performance Engineering
Continuous Experimentation and Improvement
CSE dimensions
CSE loop
Proposed component in the loop
Proposed IN/OUT in the loop

Fig. 1. Overview of the considered context.

This is compounded by situations in which the original developers are no longer available. In such cases, the system maintainability is strongly related to its architecture (Daneva and Bolscher, 2020).

With continuous engineering practices, developers have greater control and visibility of defects as well as early access to the state of quality attributes, enabling them to remedy any potential issues during the system development. Interactions between design-time and run-time in software engineering allow for dynamic adaptation and ensure non-functional properties and end-user expectations (Brunelière et al., 2018). Notably, continuous monitoring is considered an enabler for the early detection of QoS problems, such as performance degradation (Hasselbring and van Hoorn, 2020).

Fig. 1 illustrates the context of this study. We considered the holistic view of the CSE proposed in Fitzgerald and Stol (2017) and tailored the figure by focusing only on the continuous activities that are central to SP (in the white rectangles) and by adding the specific task of performance assessment (in the light blue rectangle).

This figure also shows the bridge artifacts between CSE and SP. SA and runtime data are output artifacts of CSE that feed the SP, whereas refactoring enters the CSE. SA is the abstraction that represents the best trade-off between model complexity and expressiveness and allows the assessment of the performance characteristics of a (software) system. Runtime data represents the running system that provides all parameters to set the performance model defined by the SA. Refactoring consists of suggestions on how to change the software system to solve or mitigate performance degradation.

Based on the above tailoring, we formulate a query to extract the literature object of our analysis.

2.2. Related work

In this section, we discuss secondary studies that somehow address the role of SA and SP in the CSE paradigm.

Koziolek (2010) conducted a holistic literature review classifying the approaches concerning performance prediction and measurement for component-based software systems based on studies published from 1999 to 2009. These approaches introduce specialized modeling languages for the performance of software components and aim to understand the performance of a designed architecture instead of code-centric performance fixes. The review acknowledges the limited support for the runtime life-cycle stage of software components and the lack of consensus on the performance modeling language. In fact, none of the reviewed approaches was ready to gain widespread use due to limited tool support, fundamental in the case of CSE. The surveyed methods support modeling the runtime life-cycle stage of software components only in a limited way (i.e., they only included the workload and the usage profile modeling of the component-based system at runtime). For continuous performance improvement, dynamic software refactoring is of paramount importance. However, the reviewed primary studies in (Koziolek, 2010) partially supported dynamic and automated mechanisms for CSE for performance aspects. Furthermore, online performance monitoring at runtime was not fully combined with modeling techniques to react on changing usage profiles and deployment environments. As an extension of the review published by Koziolek (2010), we present an updated analysis of the literature in our study including papers published until February 2022. Differently from (Koziolek, 2010), our work focuses on publications investigating performance engineering methods that can be applied in the context of CSE and considering approaches applicable to all kind of systems without limiting our study to component-based software systems.

In a subsequent holistic literature review, Becker et al. (2012) specifically investigated model-driven performance engineering approaches for self-adaptive systems based on studies published from 2004 to 2011. The authors provided a thorough classification scheme, presented as a feature diagram. They distinguished between the reactive and proactive adaptation strategies, and they derived two main categories of adaptation: design-time and run-time.

Self-adaptation is the ability of the system to decide autonomously (i.e., without or with minimal human intervention) how to adapt to accommodate changes in its context and environment, and to manage the uncertainty in the environment in which the software is deployed, and during the execution (Weyns, 2020). Self-adaption is enabled because self-adaptive systems use an explicit representation of their own structure, behavior, and goals (de Lemos et al., 2013). Recent efforts have been devoted to investigating motivation and the application of self-adaptation in practice (Weyn et al., 2022). In this context, CSE defines a continuous engineering process needed to quickly respond to market and new customer requirements, i.e., to build solutions that much more accurately align with dynamic customer needs (Bosch, 2014). A number of continuous activities (such as continuous monitoring, continuous integration, and so on) are part of an overall CSE (Fitzgerald and Stol, 2017). CSE and (self-)adaptation are two different run-time mechanisms in the sense that self-adaptation is the ability of a system to manage changes and uncertainty, while CSE is a dynamic process that continuously engineers the system allowing to add new features, functionalities, and abilities, or new smarter implementations of them.

While the aforementioned study (i.e., Becker et al. (2012)) consistently outlines performance engineering in self-adaptation targeting model-driven performance approaches, we seek to extend the area of interest to a more general interleaving of runtime knowledge and architectural models in CSE, without limiting the study to model-driven performance approaches.

Recently, different studies have covered the different aspects of CSE (Pahl and Jamshidi, 2016; Taibi et al., 2019; Jabbari et al., 2016) in several contexts. However, differently from our work, they did not specifically consider SP engineering. Pahl and Jamshidi (2016) have presented a systematic mapping study of 21 papers published from 2014 to 2015 to identify, taxonomically classify, and systematically compare the existing research body on microservices and their application in the cloud, by positioning them within the context of continuous development. Taibi et al. (2019) have presented a systematic mapping study of continuous architecture with microservices and DevOps, and included 23 studies published from 2014 to 2017 in their investigation. They provided an overview of the architectural styles of microservices applications, highlighting the advantages and disadvantages of each style. However, no consideration was given to non-functional properties, such as performance. Jabbari et al. (2016) presented a systematic mapping study on the classification of DevOps and included 49 papers published from 2011 to 2016. They investigated how DevOps was exploited during software development processes. They found that few primary studies exploited model-driven engineering techniques and focused on quality assurance.

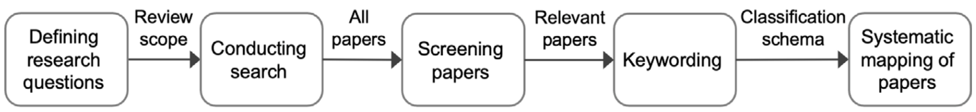


Fig. 2. Mapping process.

Finally, Bezemer et al. (2019) conducted an industrial survey to gain insights into how performance is addressed in industrial DevOps settings. In particular, they have investigated the frequency of executing performance evaluations, the tools being used, the granularity of the performance data obtained, and the use of model-based techniques.

In contrast to the aforementioned papers, in this paper, we execute a systematic mapping study that investigates how performance is assessed in the context of CSE by providing a classification schema able to classify primary studies concerning research areas, addressed target problems, provided contributions, devised methodologies, studied performance indices, and type of used data. Unlike other related work, the focus of this study is the combination of SA and SP within CSE.

3. Research method

To gain insights into the current research practices on the architectural support for SP within CSE, we conducted a systematic mapping study of the literature based on the guidelines proposed by Petersen et al. (2015), and the "snowballing" process defined by Wohlin (2014).

The process adopted in this study consists of five steps, as shown in Fig. 2. In the first step, we *define the research questions* and identify the scope of the review to be incorporated in the next steps. Subsequently, we *conduct a literature search* to retrieve a list of relevant publications that are then selected by applying the inclusion and exclusion criteria in the *papers selection* step. The selected publications are the input for the *data extraction*, where we categorize the relevant publications by considering their full text. As output, we obtain a classification schema that is used as input for *mapping the data retrieved from the papers* to the questions.

In the following section, we describe the five aforementioned steps of the mapping process. In Sections 4–8, we present the results of the analysis and mapping of the papers. Moreover, to simplify the replicability of this study, a complete replication package is made publicly available (Eramo et al., 2021).

3.1. Defining research questions

To investigate the contributions of the scientific community on the architectural support for SP within CSE, we formulated the following research questions:

RQ1: What ***research areas*** *and* ***target systems*** *have been investigated?* The aim of this research question is two-fold: *(i)* to highlight the research areas that are focused on providing solutions in this field; and *(ii)* to extract the subject systems on which the application or technique is intended to apply (we refer to this by the term "target system" (Bjørner, 2006; Bryant et al., 2010)).

The rationale for this RQ is strongly related to the goal of this study and it aims to define how and what degree of performance engineering is exploited in CSE. It also determines which application domains have been considered in the selected studies. This helps us to understand the maturity of continuous performance assessment and to determine the applications for which performance is considered a key constraint.

RQ2: What and how ***performance problems*** *have been addressed?* This research question focuses on the identification of the SP engineering problems targeted in a CSE process and the solutions proposed to address them.

Several issues can be addressed in SP engineering, including requirement specification, modeling, analysis, prediction, and suggestions to improve the software system performance. The rationale behind this RQ is strictly related to the identification of the SP target problems considered by the researchers and the related contributions proposed in the context of CSE.

RQ3: What ***instruments*** *have been adopted?* Several instruments can be used to address the performance issues. We partitioned these into three categories of keywords: input data, methodologies/techniques, and performance output measures/indices. The first category includes the types of data that are used to conduct the performance analysis, and spans from runtime data through requirements to software/performance models. Examples of the second category are patterns/anti-patterns recognition, performance prediction or testing. The third category aims to identify the target metrics, such as response time, throughput and network bandwidth. This research question aims to identify which, and with what degree instruments have been applied in the context of CSE.

The rationale behind this question is strictly related to determining the characteristics and limits of the proposed solutions in the SP and CSE domains.

RQ4: What are the gaps in current research ***gaps*** *and the implications for future research?* This research question combines the different viewpoints highlighted in the previous three RQs, and aims to identify contexts that have been hitherto most or least investigated. For example, how intensively has *performance assessment* (as a problem) been investigated in the context of *continuous monitoring* (as a research area) on *distributed systems* (as a target)?

We are particularly interested in highlighting combinations that exhibit low intensities. We expect that some of these combinations to raise negligible interest and others to represent research gaps. Moreover, we are focused on identifying areas worth investigating in the near future.

3.2. Conducting search

The search process involves the identification of search strings, the outline of the most relevant bibliographic sources and search terms, the definition of the inclusion and exclusion criteria, and the query execution.

Search strings. We defined the search keywords based on the PICO[1] terms (Kitchenham and Charters, 2007) in our questions structure. As suggested by Kitchenham and Charters (2007), the comparison and outcome terms cannot always be considered in software engineering if the research focuses on general investigation. hence, we extracted the keywords from the population and intervention terms.

We refined the search terms and the related search strings to ensure that relevant studies were returned by combining the keywords and reviewing the titles and abstracts of the search results. The final set of keywords is listed in Table 1. The resulting query was then adapted

[1] PICO elements include: problem/patient/population, intervention/indicator, comparison, outcome (Huang et al., 2006).

Table 1
Definition of keywords.

Population	P terms
Software performance	Software performance
Intervention	I terms
Software architecture	Software architecture
Continuous Software Engineering	DevOps, continuous integration, continuous deployment, continuous development, continuous improvement, DevOps, continuous evolution, continuous monitoring

Table 2
Inclusion and exclusion criteria.

Criteria	Assessment criteria	Step
Inclusion	The paper covers software performance engineering issues	All
	The paper proposes model-based or architectural approaches for CSE/DevOps or contributes to (self-)adaptation/refactoring targeted to software performance	All
Exclusion	The paper is not fully written in English	T/A
	The paper is not peer-reviewed (i.e., blog, forum, etc.)	T/A
	The paper is a duplicate (only consider the most recent version)	T/A
	The paper is a position papers, book chapter or work plan (i.e., the paper does not report results)	T/A
	The paper does not fully or partly focus on software performance	All
	The paper does not fully or partly focus on software architecture or software engineering	All

to the syntax of each bibliographic source. All the queries applied to the different bibliographic sources are reported in the replication package (Eramo et al., 2021).

(“continuous software engineering” OR “continuous integration” OR “continuous deployment” OR “continuous development” OR “continuous improvement” OR DevOps OR “continuous evolution” OR “continuous monitoring”) AND “software architecture” AND “software performance”

Query 1: Baseline search string.

Bibliographic sources. We selected a list of relevant bibliographic sources following the suggestions of Kitchenham and Charters (2007), as these sources were recognized as the most representative in the software engineering domain and were used in many reviews. The list includes: *ACM Digital Library, IEEEXplore Digital Library, Scopus, and SpringerLink*.

Inclusion and exclusion criteria. We defined the inclusion and exclusion criteria to be applied to the title and abstract (T/A) or to the full text (All), as reported in Table 2.

Search. Finally, the search was conducted on March 1st, 2022, and all the publications hitherto available were included. The application of the search terms returned 215 papers, which was the result of merging the papers from the bibliographic sources considered, as depicted on the left side of Fig. 3. Upon removing the duplicate papers, we obtained 195 papers.

We validated the search string with a “golden set” of papers that ensured that we did not leave out relevant works. The papers considered in the golden list were: [SP2], [SP9], [SP10], [SP12], [SP31], and [SP49].

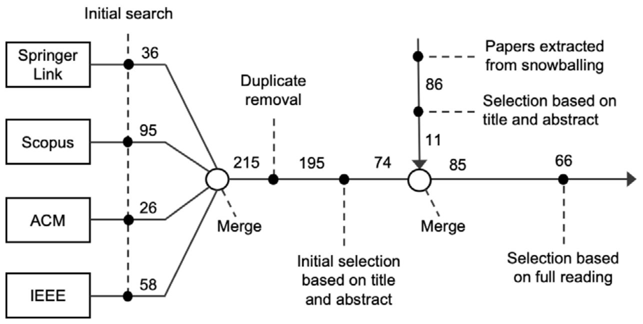

Initial search
Springer Link
36
Scopus
95
ACM
26
IEEE
58
Duplicate removal
Merge
215
195
74
Initial selection based on title and abstract
Papers extracted from snowballing
86
Selection based on title and abstract
11
Merge
85
66
Selection based on full reading

Fig. 3. Overview and numbers of search and selection process.

3.3. Papers selection

After obtaining the initial set of papers, we applied the selection process described in this section. An overview and numbers of this process is depicted in Fig. 3.

Testing the applicability of the inclusion and exclusion criteria. Before applying the inclusion and exclusion criteria, all the authors tested their applicability iteratively, on a subset of 20 randomly selected papers. Based on the disagreements, and on a shared discussions, we clarified the inclusion and exclusion criteria.

Applying the inclusion and exclusion criteria to the title and abstract. The refined criteria were applied to the remaining 195 papers (Table 2). We have included papers that meet all the inclusion criteria and excluded those that meet any of the exclusion criteria. Each paper was read by two authors; in the case of disagreement, a third author helped to resolve the disagreements. For 32 papers, the authors discussed and cleared possible disagreements. Out of the 195 initial papers, we included 74 papers based on the title and abstract. The inter-rater agreement before the third author was involved was 0.75, obtained using Cohen’s kappa coefficient, which indicated a substantial agreement between the authors (Emam, 1999).

Snowballing. We performed the snowballing process (Wohlin, 2014), by considering all the references presented in the retrieved papers and evaluating all the papers referencing the retrieved papers, which resulted in one additional relevant paper. We applied the same process to papers retrieved from the initial search. A snowballing search was conducted in March 2022. We identified 86 potential papers, but only 11 were included (after applying the inclusion and exclusion criteria to the title and abstract) in order to compose the final set of publications that were subjected to full reading and data extraction.

Full reading. The screening of the remaining 85 papers was performed independently by two authors. We ensured that the papers were randomly assigned such that each author had a similar number of papers assigned. Moreover, we permuted the assignments to enable a good balance between each pair.

We read the 85 papers in full and applied the criteria defined in Table 2. To improve the reliability of our study (Wohlin et al., 2013), we sought the services of a third author in two papers to reach a final decision. In this case, the inter-rater agreement before the third author was involved was strong (Cohen’s kappa coefficient = 0.94; almost perfect agreement). Based on this process, we selected a total of 66 papers for the review.

3.4. Data extraction and analysis

To ensure a rigorous data extraction process and to ease the management of the extracted data, a well-structured classification framework was rigorously designed, as explained in this section.

To answer our RQs, we extracted a set of information from the 66 selected papers. Notably, we defined the main concepts and corresponding data in our study by following a systematic process called *keywording*. The goal of this process is to effectively develop a classification scheme so that it fits the selected papers and considers their research focus into account (Petersen et al., 2008). In particular, we identified the codes for our coding schema using a semi-automated process in the following two steps:

1. *Automatic identification of the most recurrent keywords in the papers*. We used natural language processing (NLP) techniques to automatically identify the keywords that were most frequently mentioned in the abstracts of the selected papers. We started by collecting the abstracts from a single dataset that constituted the text corpus for the processing. The corpus was pre-processed in two phases: *noise removal* and *normalization*. In the *noise removal* phase, we performed an initial clean-up by converting the text to lowercase and by removing punctuations, tags, special characters, and digits. We then applied two normalization techniques: *stemming* to remove suffixes and *lemmatization* to group together words having the same root. As a final pre-processing step, we removed the prepositions, pronouns, and conjunctions. Thus, we created a vector of words counts by deriving a *bag-of-words* model for the text. In this model, words order and grammar information are not considered because the entire text is represented by the multiset of its words from which one can derive their multiplicity. The vector of words counts was then used to obtain the 50 most frequent single words and two words (bi-grams) and three words (tri-gram) combinations.
2. *Manual refinement of the keywords.* We refined our collection of keywords and concepts by reading the abstract of each paper. We combined together keywords from different papers to develop a high level understanding of the nature and contribution of the research. This helped us to define a set of categories of keywords that is representative of the research questions. However, the paper abstracts were too limited to define all meaningful keywords. Therefore, we thoroughly examined all the sections of the papers to consolidate our classification schema. We performed a double round of reviews by shuffling reviewers (among the authors of the paper) after the first round. Finally, upon obtaining a consolidated set of keywords, we have re-organized the original categories to obtain the final classification used hereafter.

We assigned each author a set of 10 randomly selected papers, to validate the coding schema and keywords, and to ensure a common understanding among the researchers. Subsequently, we discussed on the results of the coding and possible inconsistencies, and we finalized the schema.

The resulting classification framework is presented in Table 3. It comprises seven categories, with groupings of pertinent extracted keywords. A detailed description of each keyword is provided in Appendix C. Each category addresses the corresponding research questions by using the metrics described in details below.

For RQ_1, we extracted the information on the research area and target system. Both research areas and targets may be either fully or partially investigated. The primary goals of a paper are included in the fully-investigated research areas and targets.

In a partially-investigated research area, either the primary goals of the paper are considered, or it is a secondary area that supports the primary goals (while a partially-investigated target implies that the study can support that targeted system, even if it is not described as central in the paper). As an example, we considered papers, such as [SP40], where *continuous software engineering* has been partially investigated because the paper mainly focuses on *software performance engineering*. It is important to note that papers might fully investigate one area and partially investigate another one, and therefore there might be more than one research area assigned to each paper.

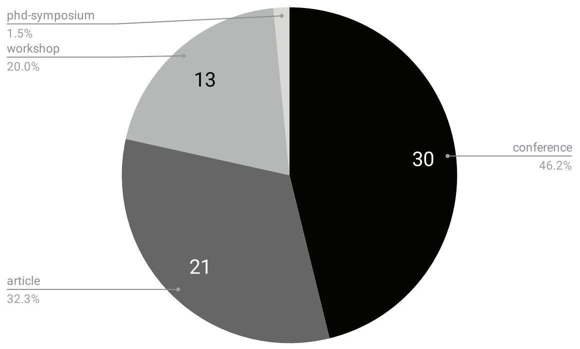


Fig. 4. Selected paper types.

Regarding the problems addressed in the selected papers for RQ_2, we extracted information on the primary and secondary problems and contributions reported by the selected papers. As an example, we considered *performance requirement* as the primary problem and *quality of service* as the secondary problem for paper [SP58], while we considered *performance analysis approach* as main contribution and *performance modeling approach* and *performance prediction approach* as the secondary contributions for the paper [SP45]. The same approach was applied also for RQ_3.

After extracting all the information from the selected papers, we analyzed the data by counting the number of papers obtained for each data group and metrics (see Table 3). Therefore, for (RQ_1), we counted the number of papers that fully investigated a topic, partially investigated it or considered the topic (either fully or partially investigated it). Similarly, for (RQ_2), we counted the papers that considered each research problem (primary, secondary, or both). Following the previous approach, (RQ_3) was analyzed by counting the number of fully- and partially-evaluated indices, used data, and methods.

For the first three RQs, we considered the individual keyword results. RQ_4 has been introduced to observe the results across different keywords, with the goal of identifying contexts that have scarcely been investigated. To achieve this goal, we introduced bubble plots, which allowed for a straightforward comparison of how intensively a certain context was investigated in comparison to other contexts. In the plots, the size of the bubbles represents the number of papers that investigate a specific keyword at the intersection of a research area (x-axis) and domain (y-axis). The exact number of papers is annotated for each bubble. In this way, one can visually identify areas of the plot where no bubbles or only small bubbles are observed, thereby establishing combinations of research areas and target systems where a certain keyword appears to be seldom investigated in the considered literature. Moreover, we analyzed the combination of the previous results and future improvements and direction described in the selected papers to identify a set of implications for future research.

A complete list of the keywords and metrics used for the analysis is presented in Table 3.

4. Results overview

We selected 66 peer-reviewed publications, including 21 (32.3%) articles, 30 conference papers (46.2%) and 15 workshop (or others satellite events) papers (21.5%), as shown in Fig. 4. The selected papers were presented at 33 different venues. Fig. 5 depicts the list of venues considered by at least two of the selected papers.

The selected papers show a continuously growing interest on performance-targeted CSE between 2016 and 2022, while a very small number of publications have been published until 2015, which is in line with the fact that continuous development and DevOps have emerged only recently (Fitzgerald and Stol, 2017). Thus, we can gather that the intuition of supporting SP through continuous engineering solutions has

Table 3
Classification framework: Data Extraction, Keywords, and Metrics adopted for the analysis. Keywords in blue have been obtained in the manual keyword refinement step.

RQ	Categories	Keywords	Metrics
RQ1	Research area	Software performance engineering, Software architecture, Continuous Software Engineering, DevOps, Continuous Monitoring, Agile software development.	#Fully investigated topics (F) #Partially investigated topics (P) #Investigated topics (F, P)
	Target system	Embedded/CPS, Cloud, Real-time, Distributed, Data intensive Software intensive, Component-based software/Microservices/SOA.	
RQ2	Research problems	Performance evaluation/assessment, Performance requirement, Quality of service, Resource allocation/deployment, Uncertainty.	#Main target problem/contributions (M) #Secondary target problem/contributions (S) #Main or Secondary target problem/contributions (M, S)
	Contributions	Performance analysis approach, Domain specific languages, Continuous engineering framework, Performance modeling approach, Tool support, Performance prediction approach, Self-adaptation.	
RQ3	Methodologies	Performance model, Model based engineering / Model driven engineering (MBE/MDE), Performance antipattern/Root cause/Bottleneck detection, Performance prediction techniques, Performance analysis techniques, Parametric dependency, Performance testing/Load Testing/Benchmarking, Performance model generation/extraction, Simulation, Machine Learning, (Multi-objective) Optimization.	#Fully evaluated indices/data/methods (F) #Partially evaluated indices/data/methods (P) #Fully or Partially evaluated indices/data/methods (F, P)
	Indices (output)	Response time, Utilization, Throughput, Resource demand, Network bandwidth, Memory/Memory Leaks.	
	Used Data (input)	Runtime/Monitored, Workload, Requirements, Performance model, Software model, Data analytics.	
RQ4	Research area and Target system combined with data from RQ2 and RQ3	All keywords of RQ1 combined with specific keywords of RQ2 and RQ3.	#Fully or Partially addressing a specific combination of keywords (F, P)

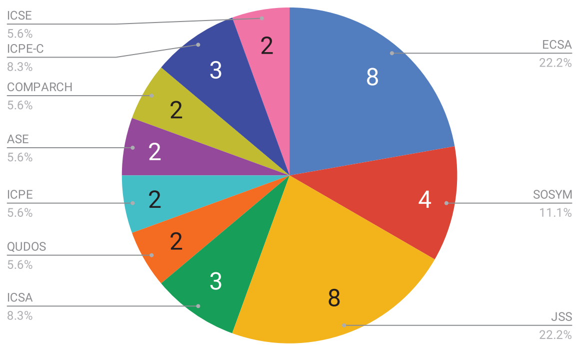


Fig. 5. Selected paper per venues.

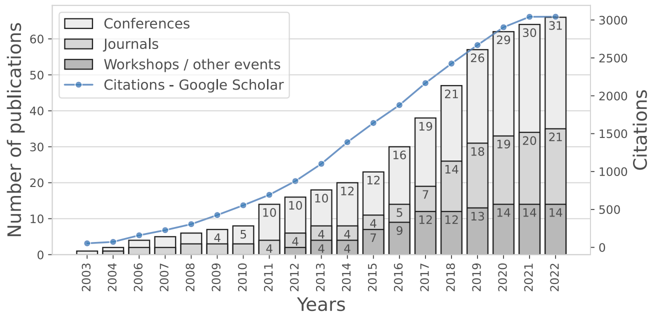


Fig. 6. Cumulative number of primary studies and citations in each year, and by type of publication.

been strengthened since 2015, although there has been a decrease in the number of publications in 2020 and 2021.

The cumulative number of citations per year for the primary studies (Fig. 6, blue line, source: Google Scholar) highlights that the growing interest concerns not only the publications but also the number of citations obtained from the studies of the dataset. Moreover, citations have grown rapidly since 2016. It is worth noting that the entire dataset has a total of more than 3000 citations to date (i.e., they have more than doubled in the last six years). This result indicates an important growing interest in the context of this study.

Thus, although the DevOps and CSE domains are relatively young compared with performance engineering, significant contributions have been made in the last ten years and researchers are becoming increasingly active (Fig. 6).

In the following sections, we present the results of this study aimed at answering our research questions (see Section 3.1). For each extracted piece of information, we report both the quantitative data and an interpretation of the results obtained.[2]

5. What research areas and target systems have been investigated (rq_1)

The topic considered combines several aspects, as we discussed in Section 2, which may attract interest from different disciplines and for different scopes of research. To provide an overview of this research topic, we describe the research areas focused on providing solutions and the target systems for which solutions have been developed.

Fig. 7 depicts the principal **research areas** that are focus areas of the selected papers (each study may contribute to more than one area).

[2] For sake of readability, some of the results have been summarized; the interested reader may consult the extended version at http://arxiv.org/abs/2304.02489.

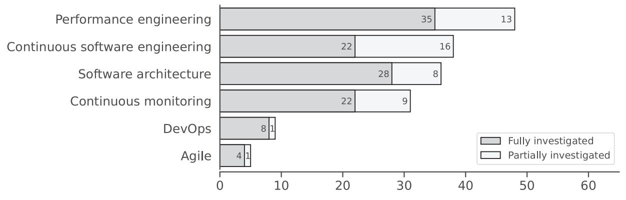


Fig. 7. Research areas - results.

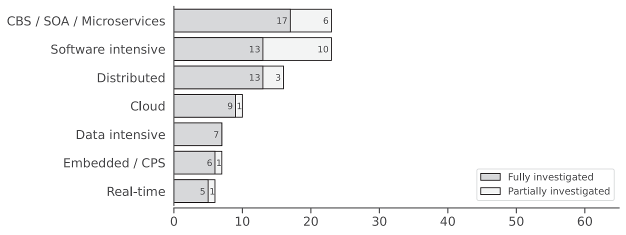


Fig. 8. Target systems - results.

The bar chart in the figure compares the identified research areas with respect to the number of papers[3] For each research area, we stacked two different bars to combine both the *fully* and *partially investigated* areas (as described in Section 3.4).

As expected, the selected papers mainly focus on the areas of *software performance engineering* and *software architecture*, which is in agreement with the information on publication venues obtained in the publication trends.

Many studies have contributed to *continuous software engineering* that considered at least one of the continuous dimensions introduced in Section 2.

Significant attention has been paid to the continuous monitoring of data to realize continuous development, delivery, and integration, which improves system performance at the level of SA. As the monitored data enable performance analysis, continuous monitoring represents a fundamental capability to provide CSE. In general, the results confirm that in the areas of SA and SP, CSE is an emerging topic that is progressively gaining the interest of researchers and practitioners. Existing conferences and systematic reviews on DevOps suggest that software engineering researchers have a strong interest in this topic. Despite this, only a few papers focus on DevOps, highlighting an interesting gap to be addressed by the research community. However, the limited number of articles on agile is an expected result. Even as a precursor to DevOps, agile development is more code-focused and produces less documentation (e.g., software/design models), not supporting SA-based SPE.

Although a large number of selected papers have been fully identified as contributors to the CSE field, the number of papers that partially investigate this area has increased. Beyond that, several selected papers have not been placed within the context of CSE or DevOps, which can be attributed to the fact that these papers do not explicitly place themselves in these areas, even if they actually offer solutions that cover several aspects of an iterative development process, wherein updates are made continuously. However, this confirms that this emerging theme is gaining ground.

Fig. 8 describes the specific type of **target systems** (as case studies) in the selected papers. These keywords are not meant to be mutually exclusive, in that a system could be, for example, at the same time a distributed and realtime system. We have basically identified, in each paper, the main characteristics of the systems to which the paper approach/solutions have been (or can be potentially) applied, where they have been unambiguously identified.

Component-based software and *software intensive* systems have been most investigated. These targets are characterized by the existence of different components, services, or subsystems. These can be independent of each other (e.g., microservices) or have strong dependencies and relationships amongst themselves and with the environment, as in many complex systems. However, in both cases, the presence of heterogeneous components makes it necessary to integrate these activities into continuous development and maintenance (e.g., DevOps). The component-based development paradigm is based on the concept of reuse within distinct components (e.g., services), enabling integration. Once integrated and implemented, these components must enter a continuous dimension, that is, to know and analyze the behavior after the integration and not only of the single component; therefore, they are also important in the context of performance.

Next, the results show the targets of *cloud* and *distributed systems*, characterized by the technology stack and infrastructure complexity. Cloud nodes may attain performance orders of magnitude worse than other nodes (Armbrust et al., 2009). For instance, if during the hosting of a mission-critical service or a scientific application, performance variability and availability become a concern, cloud monitoring is required to continuously measure and assess the infrastructure or application behavior (in terms of performance, reliability, power usage, and security) to adapt the system to changes or to apply corrective measures. Generally, we can observe that in open systems characterized by uncontrolled requests, continuous engineering, which supports their integration and evolution, is fundamental to identifying the occurrence of further problems not observed before.

Finally, we note that more recent and innovative targets are not yet widely investigated. For instance, in the case of *data intensive* systems, the massive use of big data and machine learning requires efficient management of resources, performance, and security.

Main findings:

- Significant attention is paid to continuous monitoring of data, which represents a fundamental capability to provide CSE;
- CSE and DevOps are gaining ground: most studies offer solutions covering several aspects of the iterative development process where updates are made continuously;
- Software intensive systems, especially when component-based, are the most investigated ones: their heterogeneous components require the integration of their activities in a continuous development and maintenance;
- More recent and innovative targets, such as cloud and data intensive systems, have not been widely investigated.

6. What and how performance problems have been addressed (RQ$_2$)

We classified the 66 selected papers by considering the research problems addressed by them (we divided the target problems into five categories), and their research contributions (we identified seven different types of contributions). In particular, the selected papers were thematically associated with at least one target problem and at least one research contribution based on their research directions and scope. We provide a detailed overview of the performance problems that have been addressed and illustrate them with exemplary papers.

Fig. 9 presents the problems targeted by the selected papers (**research problems**). The results obtained confirm that most of the selected papers are aimed at solving performance evaluation and assessment problems. This is a relevant goal in the development of modern

[3] Although our topic is characterized by the dimensions shown in this figure, the fact that the selected papers contribute to these specific fields of research is not obvious. For instance, a paper may just use a specific performance evaluation technique without contributing in that area.

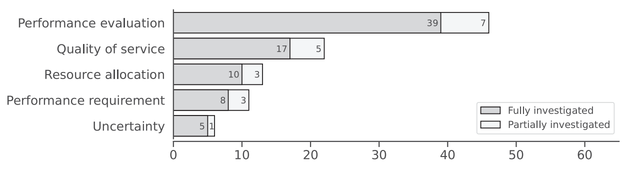


Fig. 9. Research problems - results.

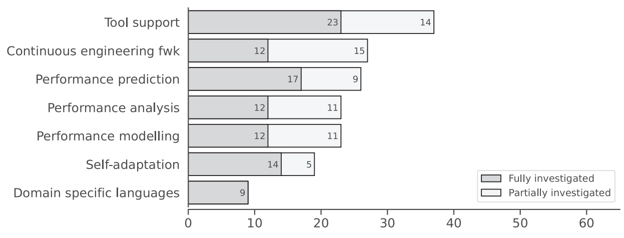


Fig. 10. Research contributions - results.

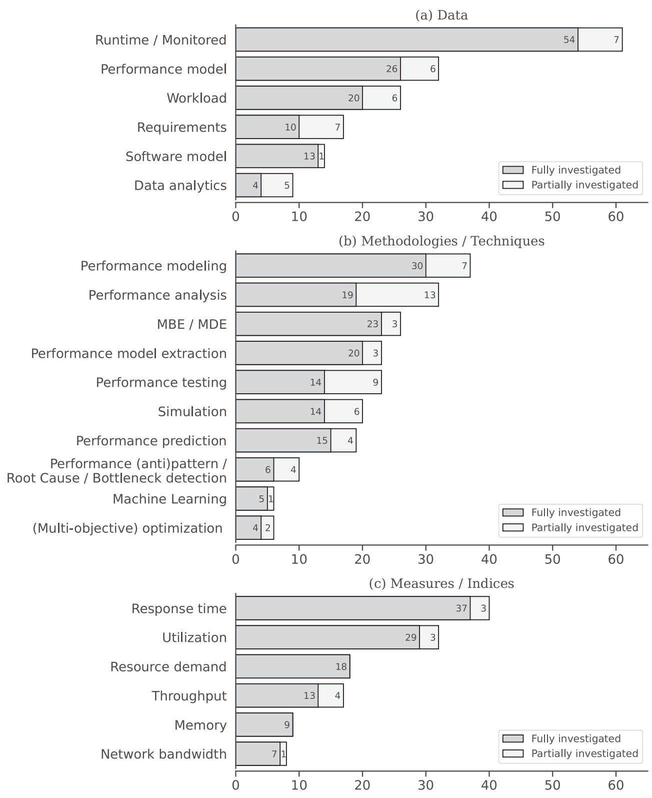


Fig. 11. Data, methodologies/techniques, measures/indices - results.

systems, where increasingly agile paradigms require that performance analyses be in a continuous dimension to be effective.

Part of the selected papers aims to provide *QoS*, where the target is to satisfy different requirements/properties for the overall quality of the system, including guaranteeing a certain level of performance.

Managing the *uncertainty* of the system behavior is an emergent topic (papers addressing this issue are recent publications), and it is cross-cutting to the target problems previously mentioned. Although having the complete model of the system represents the ideal situation, in practice, only partial and limited measures are available. Consequently, specialized performance analysis or prediction techniques must work with uncertain knowledge. The proposed studies and their discussions on the different types of uncertainty highlight relevant issues and offer new research ideas.

Fig. 10 describes the principal **research contributions**. From our analysis, we can state that most of the studies contribute to the continuous performance assessment of the system with *performance analysis, performance modeling, and performance prediction* approaches. Most of the approaches use performance evaluation to achieve the desired QoS objectives in software systems.

More than half of the research problems considered were addressed using the support of dedicated *tools* or CSE *frameworks* and were well partitioned between performance prediction, performance analysis, and performance modeling. This is an interesting result, as in the context of this study, the quality and performance requirements demand the support of continuous engineering frameworks or dedicated tools.

Other research problems were dedicated to *self-adaptation* approaches and, to a lesser extent, *domain-specific languages*. The latter result shows that few studies have exploited abstraction for continuous performance control, although they consider large heterogeneous runtime data. Raising the level of abstraction of the specification would favor increased automation and interoperability.

Main findings:

- Performance prediction, performance analysis and performance modeling have been further explored in order to offer adequate support in continuous development;
- A relevant number of self-adaptation approaches have been proposed to ensure the quality of services (including performance);
- Quality and performance requirements demand the support of continuous engineering frameworks or dedicated tools.

7. What instruments have been adopted (RQ_3)

Performance analysis can be conducted by adopting multiple techniques with different output-targeted metrics and with the support of different types of input data. In this section, we aim to identify the instruments that are adopted more often in the context of the study.

Fig. 11(a) reports the input **data**. In regard to CSE, a system is continuously monitored to feed performance indices back into a performance model that supports predictive analyses.

A total of 61 and 32 papers (of 66) used *runtime/monitored* and *performance model* as input data, respectively. Lesser number of papers consider the performance model as the input data because performance models have only been integrated into software engineering processes in the last few years. Whereas runtime performance assessment and fixing have long been considered as common practices. Majority of the papers that used monitored data also proposed a continuous approach to monitoring performance features and then used them mainly for analysis and prediction.

A significant number of papers considered the *software model* as an input to the process. This is likely due to the different notations that are usually adopted for representing software models, preventing an automated full integration in the performance assessment task.

The results here also provide evidence that *data analytics* has not yet been largely considered in this domain. However, interest in data analytics has grown over time; thus, data analytics is expected to become a primary source of inputs in the next few years.

Fig. 11(b) presents the **methodologies/techniques** used in the selected papers. As expected, the majority of papers are focused on *performance modeling* and *performance analysis*. It is necessary to build and analyze models to address the performance issues early in the lifecycle. *Model based software engineering* and *model driven engineering* techniques were also widely considered, as they did not restrict the adoption of models in the performance domain. A considerable number of papers deal with *performance model extraction* and *performance testing* techniques which were typically adopted when studying performance issues on existing running software systems. Finally, it is observed that although in the last few years the adoption of *machine learning* and *multi-objective optimization* techniques has spread in diverse fields in the context of CSE, they are still marginally considered.

It is observed that this occurs despite the fact that *performance modeling* is fully investigated. In certain contexts, extensive performance analysis can be difficult owing to the lack of system measurements and parameter values. Hence, in such cases, performance modeling can be fully investigated, but the analysis remains marginal among the contributions of the papers.

Fig. 11(c) shows the targeted output **measures/indices**. The three typical performance indices, namely, *response time*, *utilization*, and *throughput*, are the most widely targeted ones in the considered papers. Although, on one hand, this can be seen as an expected outcome, on the other hand it is somehow surprising that *memory* and *network bandwidth* have been significantly less studied in this context. These two measures may play crucial roles in the performance assessment of modern heterogeneous distributed software systems. Hence this result evidences a lack of investigation in this direction.

Main findings:

- Approaches of performance modeling and analysis techniques that take as input monitored data and produce response time and utilization indices as output are widely used methodologies. Requirements and data analytics rarely enter the process to target memory and network bandwidth;
- Even if machine learning and multi-objective optimization techniques are being increasingly studied, they are still marginally considered in the context of CSE;
- Model-based and model-driven techniques are widely considered, as they do not restrict the adoption of models to the performance domain, but in several cases, software models are also considered.

8. Current research gaps and future directions (RQ$_4$)

In this section, we aim at detecting potential research gaps by visualizing the number of papers that lie at the intersection of research areas and target systems for each keyword of interest. In the following section, we discuss our findings in the categories of keywords: *target problems*, *research contributions*, *used methodologies and techniques*, *used performance measures and indices*, and *input data*. To represent our results, we developed bubble plots, as described in Section 3. Moreover, we discuss the implications for future research based on the research gaps analyzed and future directions described in the selected papers.

8.1. Research problems and contributions

Figs. 12 and 13 show the bubble plots for the *Performance evaluation* and *Uncertainty* research problems, respectively. The plots present a very different situation. Many studies have targeted the problem of performance evaluation, especially in certain areas, and few papers have considered uncertainty in general. A reason for this may be attributed to the fact that uncertainty has emerged only recently as a distinct concern in software engineering and its inclusion in continuous engineering practices is still very limited.

In contrast, because the general problem of performance evaluation is specifically targeted by our study, a greater number of papers are expected to consider it. From Fig. 12, it is evident that certain research areas (continuous monitoring, DevOps, and agile) never intersect with certain target systems (real-time, embedded and CPS) when pursuing performance evaluation. This could simply be due to the scarce adoption of DevOps and agile practices in these systems.

A further gap appears in the area of the development of software intensive systems, even if it is continuously evolving owing to the adoption of new technologies, such as cloud computing, IoT, and artificial

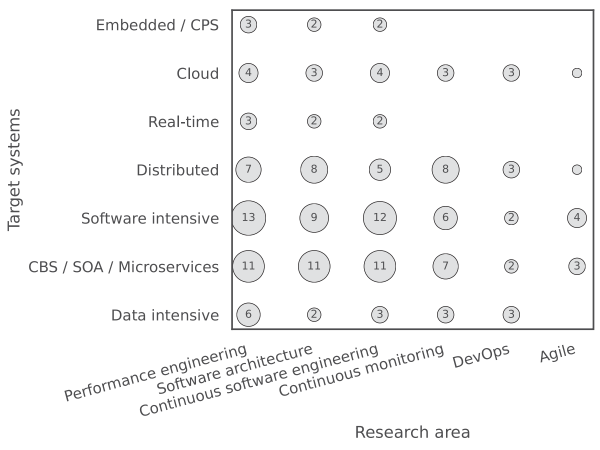


Fig. 12. Number of papers investigating the keyword *performance evaluation (target problem)* at the intersection of research areas and target systems..

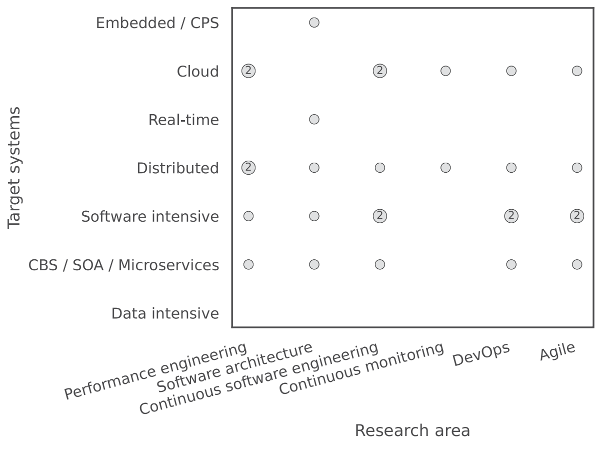


Fig. 13. Number of papers investigating the keyword *uncertainty (target problem)* at the intersection of research areas and target systems..

intelligence. Continuous engineering and DevOps can benefit from new performance engineering solutions to achieve more pervasive software (for example in smart cities, smart manufacturing, and smart mobility).

Among several research contributions identified, we reported the plots for *continuous engineering framework* (Fig. 14) and *performance prediction* (Fig. 15) as we considered them to be relevant for the purposes of our study. In Fig. 14, we observe that most of the papers proposing a novel continuous engineering framework are gathered in the lower half of the plot. The target systems for which most of these frameworks are designed are CBS/SOA/Microservices, software intensive systems, and distributed systems. Predictably, CSE is the most targeted research area in this field. However, continuous engineering frameworks are rarely or never proposed in real-time and embedded systems.

Fig. 15 shows that only a few papers proposed performance prediction approaches in embedded, real-time, and data-intensive systems. In addition, only three papers appear to focus on both DevOps and agile. This may represent a research gap about to be filled in the next few years because approaches based on artificial intelligence and machine learning, such as those in the AIOps (Dang et al., 2019) field, are rapidly emerging as a new way of modeling and predicting performance that can be more easily integrated with current DevOps practices.

8.2. Data, methodologies/techniques, and measures/indices

The types of data that are used as input to the approaches play a significant role in establishing the situations in which an approach

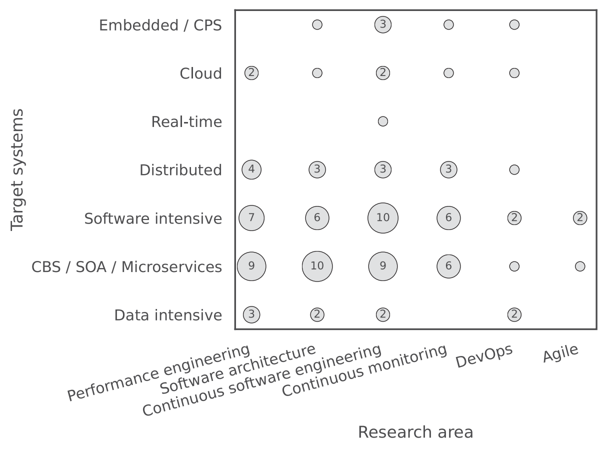


Fig. 14. Number of papers investigating the keyword *continuous engineering framework (research contribution)* at the intersection of research areas and target systems.

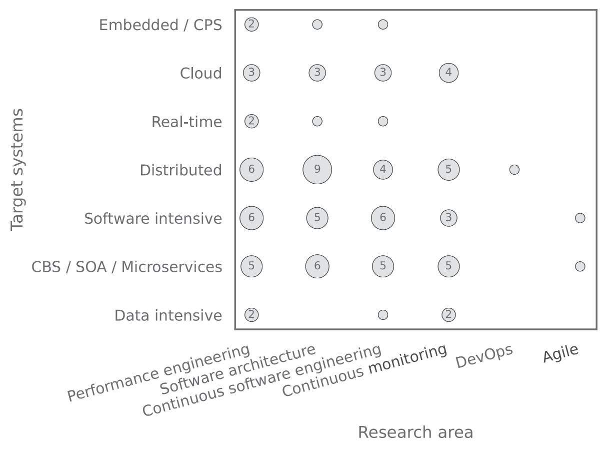


Fig. 15. Number of papers investigating the keyword *performance prediction (research contribution)* at the intersection of research areas and target systems.

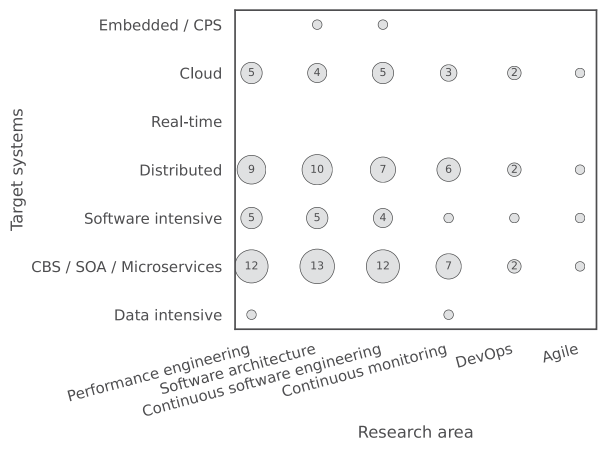


Fig. 16. Input data - Workload.

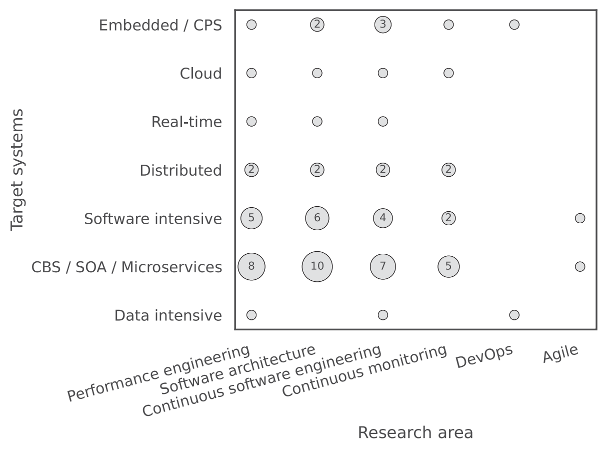


Fig. 17. Input data - Requirements.

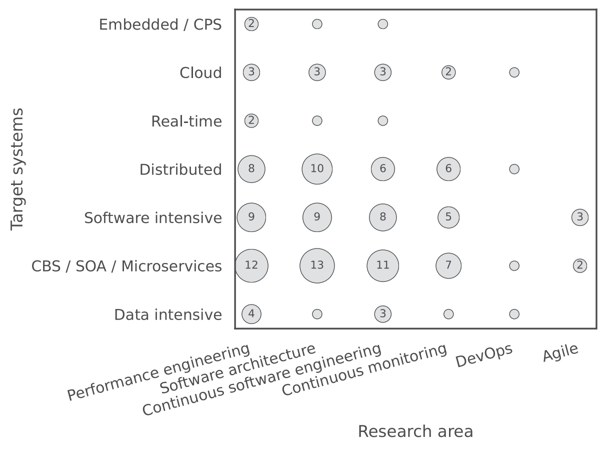


Fig. 18. Methodologies/techniques - Performance model.

can be applied and the type of information required to initiate the process. *Workload* (Fig. 16) and *requirements* (Fig. 17) are the types of data, consideration of which appears to be related to the specific target system. For instance, while the workload is often considered in the *cloud*, *distributed*, and *CBS/SOA/microservices* systems, it is rarely considered in *embedded*, *real-time*, and *data intensive* systems. The lack of consideration of workload in embedded and real-time systems is expected, whereas in data intensive systems it presents a research opportunity. When considering the use of requirements, we are presented with a different situation. From the number of papers in the bubble plot, it appears that *DevOps* and *agile* do not put much emphasis on the requirements when assessing the performance; this is unexpected because they consider the specification of requirements in their processes.

In addition, the *software intensive* and *CBS/SOA/micorservices* systems often consider the requirements as the starting point for the development of performance engineering approaches.

The methodologies and techniques that are employed in the approaches of our study represent a compelling source of information for discovering the current research interests and gaps. For instance, when examining the use of performance models (Fig. 18), it is clear that the DevOps and agile research areas lag behind the other areas in terms of the number of papers.

A different picture is presented using performance testing, as shown in Fig. 19. In this case, while all the research areas are almost equally represented, a lack of focus on the adoption of performance testing, load testing, and benchmarking is evident in real-time and embedded systems, as the number of papers contributing to above-mentioned aspects in the real-time embedded systems are 0 and 3, respectively. Generation or extraction of a performance model (Fig. 20) is a special use case in the adoption of performance models. Therefore, unsurprisingly, we can still count a few papers in the DevOps and agile research areas, whereas distributed and microservice systems seem to rely the most on the automated generation of performance models. Finally, in regards to the use of simulation (shown in Fig. 21), we observed

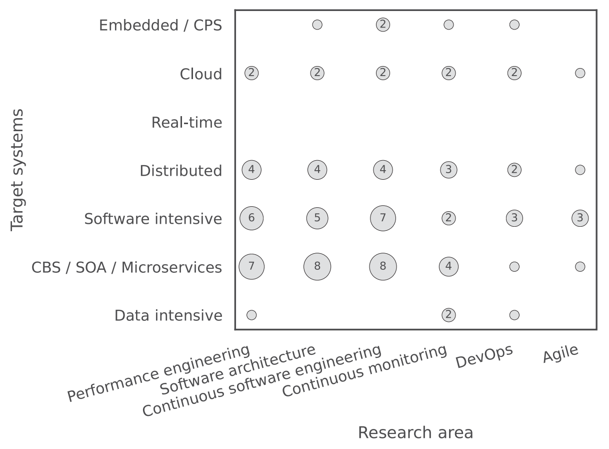


Fig. 19. Methodologies/techniques - Performance testing/Load Testing/Benchmarking.

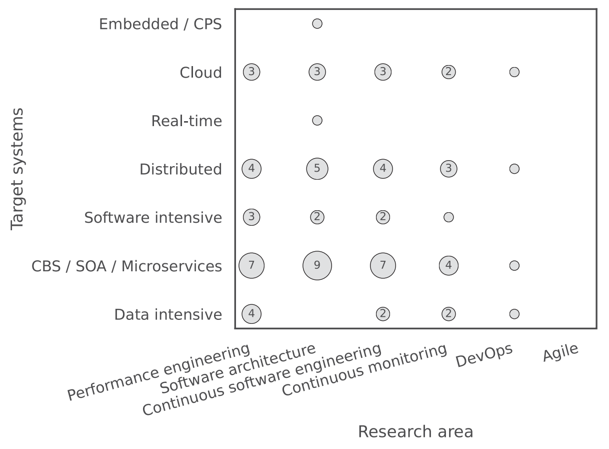


Fig. 20. Methodologies/techniques - Performance model generation/extraction.

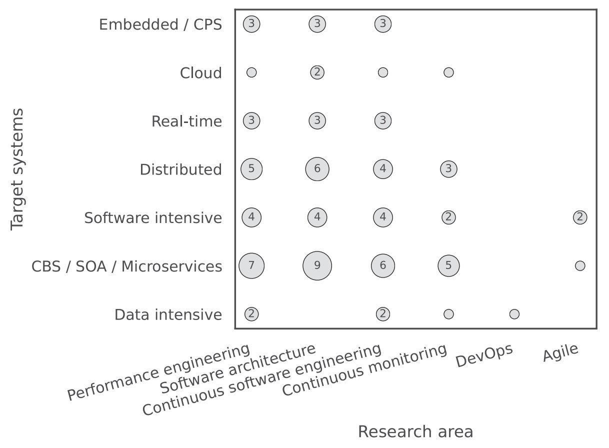


Fig. 21. Methodologies/techniques - Simulation.

that, in contrast to other methodologies, simulation has been employed in several papers for real-time and embedded systems. In addition, DevOps and agile appear to be less represented than in other research areas.

When investigating the performance measures and indices that were targeted as outputs in the selected papers, we observed two radically different situations in the number of papers that considered the keywords *response time* (Fig. 22) and *memory/memory leaks* (Fig. 23). In software performance engineering, response time and memory are

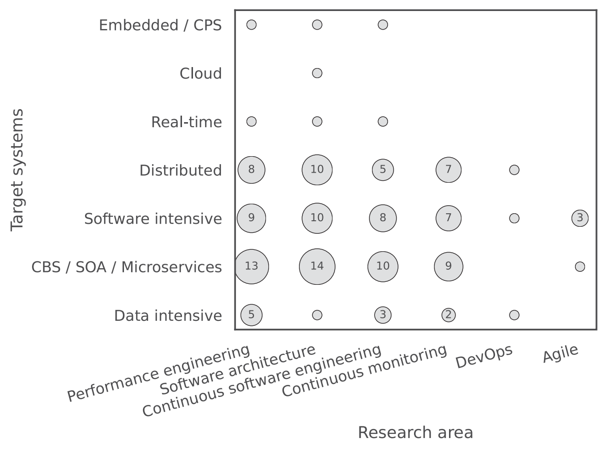


Fig. 22. Output measures/indices - Response time.

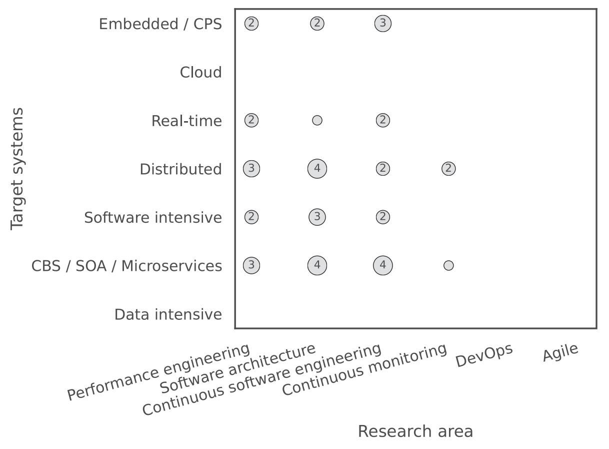


Fig. 23. Output measures/indices - Memory/Memory Leaks.

among the primary measures of interest as they are usually employed to assess the quality of service and operating costs, respectively. Nonetheless, memory is rarely considered in the papers selected for this study. Even more interesting is the fact that *DevOps* and *agile* seem to not consider memory at all in any target system. We expect memory use to become critical as ML and data-intensive systems continue to increase. In contrast, response time is considered more often in general and in particular in the domains of *distributed systems*, *software intensive systems*, and *CBS/SOA/imcroservices*. This paints a picture in which performance measures that impact the quality of service are the foremost concern in research related to CSE, thus resulting in a substantial gap in the investigation of issues related to memory usage and how these can affect the cost of providing a service.

Main findings:

- Although performance evaluation is a very well-established research topic, the conjunction with the latest trends in the software industry, such as agile and DevOps, is overdue.
- Researchers have started to study software performance under uncertainty. The research area has not yet been sufficiently investigated but may possess great growing potential.
- There is no evidence of the experience of companies in conducting performance analytics on cloud systems.

8.3. Implications for future research

In this section we discuss the implications of this study and challenges for future research.

8.3.1. Towards a culture of quality in CSE

The results obtained suggest an interest in a culture of quality, indicating that analysis and verification occur early in the CSE pipeline (as for testing in DevOps), making it easier to discover and fix defects with a collaborative approach to product improvement. In order to bring up the quality characteristics of software (architecture) and its continuous improvement (and re-architecting), QoS analysis must become an integrated activity in the entire software development life cycle, which requires continuous exposure of quality characteristics exposed to analysis.

As discussed earlier, most of the selected studies focused on a few of the performance properties. However, there is a need to strengthen the support for various properties in both performance and, in general, software quality during the continuous engineering of the system. Some challenging quality aspects to consider are: verification of correctness properties, such as architectural mismatches, evaluation of the architectural runtime models with respect to fidelity and usefulness for human inspection, extending scalability by considering influencing factors such as variation in the complexity of user behavior in experiments, supporting state management and resource provisioning mechanisms, and introducing time consumption for memory allocation and release operations to increase the prediction quality of the model. Similarly, other QoS properties, such as reliability, consistency, safety and security, and availability can be modeled and analyzed in a CSE framework.

8.3.2. Performance engineering benefits in DevOps

DevOps is gaining widespread adoption in industry. However, its principles of rapid changes, development automation, and fast feedback loop (often relying on dynamic cloud environments) conflict with the complexity of the current performance engineering approaches. Thus, performance engineering frameworks should be improved for adoption in rapidly changing systems. The structures and behaviors of the modern systems change frequently and require continuous relearning of the failure models in order to retain the prediction quality.

Moreover, these systems are characterized by a continuous stream of available data. Performance models should be built periodically, incrementally, or even continuously, and be triggered by changes to components in the monitored environment. Models can be quickly rebuilt once a potential problem is detected using only the most recent data only, and then used to compare with previous model results.

8.3.3. Data-driven methods and machine learning

Performance and load tests produce a large amount of data that can be difficult to analyze. Data-driven methods provide powerful insights into optimizing performance, building new features, and preventing problems with services, especially in distributed (enterprise) applications, in-memory databases, and big data systems.

Despite developments in modern software engineering technology, there is no established methodology for systematically employing performance engineering and data-driven engineering in continuous development.

In addition, performance evaluation based on machine learning can become an integral part of the continuous engineering process. The performance model must learn autonomously and improve itself during system operation in a production environment.

8.3.4. Continuous controlling system uncertainty

As discussed in Sections 6 and 8, uncertainty has been addressed marginally in the papers selected in this study. However, both in academia and industry, significant attention is paid to the contexts characterized by a high degree of uncertainty. To reduce uncertainty and obtain feedback on products/software as soon as possible, it is important to test assumptions and hypotheses in short cycles (Fitzgerald and Stol, 2017).

Continuous monitoring and frequent re-assessment and re-architecting are necessary to reduce the uncertainty in CSE and DevOps. An efficient analysis method can be used to propagate the effect of uncertain parameters in software systems and calculate the robustness of the performance indices, thereby enhancing the flexibility in addressing uncertainty.

8.3.5. Integration and abstraction

A recurring issue is the need to integrate methods and tools into the continuous and DevOps pipelines. Hence, it is necessary to design and develop performance engineering approaches that can be integrated with other tools and methods used in the context of this study.

Continuous monitoring is a fundamental process in CSE. In the context of implications for future research, further investigation on adaptive monitoring and analysis infrastructure that can automatically update the system and performance/quality models is needed.

External capabilities can be integrated using approaches to represent systems as black-box components by integrating black-box monitoring techniques, or creating resource profiles describing specific enterprise applications (using standard measurement solutions), instead of relying on a custom solution to collect the required data. This can be improved by supporting the collection of arbitrary information on the status of the monitored application, which requires the system integration of the corresponding type on a dynamic basis.

Using a higher abstraction level can help reduce the integration efforts. Future research can target the development of a model-based framework that considers the definition of (domain-specific) languages and automation mechanisms to ensure, by design, the potential for the monitoring, analysis, testing, and simulation in CSE. As discussed above, machine learning-supported and data-driven approaches can be used to (continuously) learn and tune the models.

8.3.6. Implications for practitioners and researchers

Practitioners can benefit from the presented results to understand how software performance engineering and software architecture can support practices in CSE and DevOps. Moreover, they can benefit from the classification of methodologies applicable to architectural support for performance engineering in the context of continuous software development.

Researchers can benefit from our results to understand research trends and research gaps, and to better focus their future work. In particular, the software performance community can leverage this work to understand whether their approaches are applicable in a CSE/DevOps context. Moreover, this study helps them define the requirements that drive the development of their tools to increase the chances of industrial adoption. In contrast, researchers in the field of CSE/DevOps can identify which methodologies and techniques can help improve software performance at some stage in the development cycle.

Main findings:

- There is a growing interest in a culture of quality, where quality properties are continuously exposed to analysis and verification during the software development life-cycle;
- The complexity of the current performance engineering approaches should improve their suitability for the DevOps principles of rapid changes, development automation, and fast feedback loop;
- There is a need for an established methodology for the systematic employment of performance engineering and machine learning/data-driven engineering in continuous development;
- Continuous monitoring and frequent re-assessment and re-architecting are necessary to reduce uncertainty in CSE and DevOps;
- Using higher abstraction levels (i.e. by means of a model-based framework or domain-specific languages and automation mechanisms) can help in reducing the effort of integrating heterogeneous methods and components in systems.

9. Threats to validity

Systematic Literature Review results might be affected by some threats mainly related to the correctness and completeness of the survey. In this section, we determined these threats according to the guidelines proposed by Wohlin et al. (2012): construct, internal, external, and conclusion validity threats. Moreover, we identified the actions required to mitigate them.

9.1. Construct validity

Construct validity is related to the generalization of the result to the concept or theory behind the execution of the study execution (Wohlin et al., 2012). We identified the threats related to the potentially subjective analysis of the selected studies. As recommended by the guidelines of Kitchenham and Charters (2007), data extraction was performed independently by two or more researchers and, in case of discrepancies, a third author was involved in the discussion to resolve any disagreement. The quality of each selected paper was checked according to the protocol proposed by Dybå and Dingsøyr (2008).

9.2. Internal validity

Internal validity threats are related to possible incorrect conclusions about the causal relationships between the treatment and outcome (Wohlin et al., 2012). In the case of secondary studies, internal validity represents how well the findings represent those reported in literature.

To address these threats, we rigorously defined the study protocol, including the data-extraction form. The data extraction form was first validated by all authors by extracting information from 10 randomly selected papers. Considering the data analysis process, threats are minimal, as we only adopted descriptive statistical techniques when dealing with quantitative data.

When considering qualitative data, keywords were defined using a semi-automated approach to transform them into quantitative data. In regards to keyword definition, we first applied natural language techniques to reduce the subjectivity of the terms selected and then manually refined the keywords collaboratively.

Finally, 10 studies were randomly selected by all the researchers to verify whether the results were consistent, independent of the researcher performing the extraction. Disagreements were discussed and resolved collaboratively when needed.

9.3. External validity

External validity threats are related to the ability to generalize the result (Wohlin et al., 2012). In secondary studies, the external validity depends on the representativeness of the selected studies. If the selected studies are not externally valid, the synthesis of their content may not be valid. In our study, we were not able to evaluate the external validity of all the included studies.

To address this threat, we applied our search string to multiple bibliographic sources, including the SpringerLink, Scopus, ACM Digital Library, and IEEEXplore Digital Library. The usage of different bibliographic sources enabled us to guarantee to obtain the vast majority of papers. Moreover, we also complemented our search by performing a snowballing activity. The inclusion of papers written only in English may have biased our results. Studies in other languages may be relevant. However, we have adopted English only as it is the language most widely used for scientific papers, and we can consider the bias related to this threat as minimal. We only included peer-reviewed papers, without considering grey literature (e.g., technical reports, master theses, and web forums, etc.). Because we aimed to identify only high-quality scientific studies, we believed that this threat was minimal.

9.4. Conclusion validity

Conclusion validity is related to the reliability of the conclusions drawn from the results (Wohlin et al., 2012).

One of these is related to the potential non-inclusion of some studies. To mitigate this threat, we carefully applied the search strategy and performed the search in eight digital libraries in conjunction with the snowballing process considering all the references presented in the retrieved papers and evaluating all the papers that reference the retrieved ones, which resulted in one additional relevant paper. We applied a broad search string, leading to a large set of articles and enabled us to include more possible results. We defined the inclusion and exclusion criteria and first applied them to the title and abstract. However, we did not rely exclusively on titles and abstracts; before accepting a paper based on the title and abstract, we browsed the full text and applied our inclusion and exclusion criteria again.

Another possible conclusion validity threat is related to the incorrect interpretation of the results. To mitigate this threat, all authors carefully reviewed the results. However, other researchers may provide different interpretations.

10. Conclusion

This paper presented a mapping study on the architectural support for SP within CSE. Of 215 relevant studies, we selected 66 primary studies, which were analyzed to answer our research questions. Thus, we have taken a deeper look at the research context, and therefore we provided ideas to researchers and developers to address the challenges related to this topic, including the fact that knowledge gaps and future topics of research have not yet been thoroughly investigated in this context. In particular, we analyzed the publication trends, the research areas and target systems, target problems and contributions, and specific characteristics of the selected primary studies through a classification framework.

This study shows that SP and SA are aspects well considered in CSE, where the most affected dimensions are continuous monitoring and continuous improvement. The results of this study also show that SPE approaches and methodologies are sufficiently mature (owing to the support of specific frameworks and tools) to be applied in continuous practices, with a prevalence in the use of data monitored at runtime. In general, SA is considered to offer specific support; in many cases, SA models are used as input for the analysis and prediction of performance as well as architectural parameters and configurations. More support has been provided to distributed systems, component-based systems, SOA, micro services, and software intensive systems in general. Other contexts, such as data-intensive or embedded systems, have fewer applications. The most interesting gaps are identified in cloud systems and systems where uncertainty needs to be investigated.

CRediT authorship contribution statement

Romina Eramo: Conceptualization, Methodology, Data analysis, Writing – original draft, Supervision, Reviewing. **Michele Tucci:** Data analysis, Machine learning expertise, Data curation, Editing. **Daniele Di Pompeo:** Dataset generation, Data analysis, Data curation, Editing. **Vittorio Cortellessa:** Data analysis, Editing. **Antinisca Di Marco:** Data analysis, Editing. **Davide Taibi:** Methodology, Supervision, Reviewing, Editing.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was partially supported by the Adoms grant from the Ulla Tuominen Foundation, Finland and the MuFAno grant from the Academy of Finland (grant n. 349488), the AIDOaRt project grant from the ECSEL Joint Undertaking (JU) (grant n. 101007350). Daniele Di Pompeo and Michele Tucci are supported by the European Union - NextGenerationEU - National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) - Project: SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics - Prot. IR0000013 - Avviso n. 3264 del 28/12/2021.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2023.111833.

References

Arcelli, D., Cortellessa, V., Di Pompeo, D., Eramo, R., Tucci, M., 2019. Exploiting architecture/runtime model-driven traceability for performance improvement. In: IEEE International Conference on Software Architecture, ICSA 2019, Hamburg, Germany, March 25-29, 2019. IEEE, pp. 81–90. http://dx.doi.org/10.1109/ICSA.2019.00017.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2009. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep., University of California at Berkeley, URL http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice. In: SEI series in software engineering, Addison-Wesley, URL http://books.google.fi/books?id=mdiIu8Kk1WMC.

Becker, M., Luckey, M., Becker, S., 2012. Model-driven performance engineering of self-adaptive systems: a survey. In: Proc. of the 8th Int. ACM SIGSOFT Conf. on Quality of Sw Arch. (QoSA '12). pp. 117–122.

Bezemer, C.-P., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., Shang, W., van Hoorn, A., Villavicencio, M., Walter, J., Willnecker, F., 2019. How is performance addressed in DevOps? In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. ICPE '19, Association for Computing Machinery, New York, NY, USA, pp. 45–50. http://dx.doi.org/10.1145/3297663.3309672.

Bezemer, C.-P., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., Shang, W., van Hoorn, A., Villaviencio, M., Walter, J., Willnecker, F., 2018. How is performance addressed in DevOps? A survey on industrial practices. http://dx.doi.org/10.48550/ARXIV.1808.06915.

Bjørner, D., 2006. The Tryptych of Software Engineering. Software Engineering 3 – Domains, Requirements, and Software Design. Springer Verlag, URL https://link.springer.com/book/10.1007/3-540-33653-2.

Bosch, J., 2014. Continuous software engineering: An introduction. In: Continuous Software Engineering. Springer, pp. 3–13.

Brosig, F., Huber, N., Kounev, S., 2011. Automated extraction of architecture-level performance models of distributed component-based systems. In: 26th Int. Conf. on Automated Sw Eng. (ASE '11). pp. 183–192.

Brunelière, H., Eramo, R., Gómez, A., Besnard, V., Bruel, J., Gogolla, M., Kästner, A., Rutle, A., 2018. Model-driven engineering for design-runtime interaction in complex systems: Scientific challenges and roadmap - report on the mde@derun 2018 workshop. In: Mazzara, M., Ober, I., Salaün, G. (Eds.), Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers. In: Lecture Notes in Computer Science, vol. 11176, Springer, pp. 536–543. http://dx.doi.org/10.1007/978-3-030-04771-9_40.

Bryant, B.R., Gray, J., Mernik, M., 2010. Domain-specific software engineering. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. FoSER '10, pp. 65–68. http://dx.doi.org/10.1145/1882362.1882376.

Cortellessa, V., Di Marco, A., Inverardi, P., 2011. Model-Based Software Performance Analysis. Springer.

Cortellessa, V., Di Pompeo, D., Eramo, R., Tucci, M., 2022. A model-driven approach for continuous performance engineering in microservice-based systems. Journal of Systems and Software 183, 111084. http://dx.doi.org/10.1016/j.jss.2021.111084, https://doi.org/10.1016/j.jss.2021.111084.

Daneva, M., Bolscher, R., 2020. What we know about software architecture styles in continuous delivery and DevOps? In: van Sinderen, M., Maciaszek, L.A. (Eds.), Software Technologies. pp. 26–39.

Dang, Y., Lin, Q., Huang, P., 2019. Aiops: Real-world challenges and research innovations. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). pp. 4–5. http://dx.doi.org/10.1109/ICSE-Companion.2019.00023.

de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Andersson, J., Litoiu, M., Schmerl, B.R., Weyns, D., Baresi, L., Bencomo, N., Brun, Y., Cámara, J., Calinescu, R., Cohen, M.B., Gorla, A., Grassi, V., Grunske, L., Inverardi, P., Jézéquel, J., Malek, S., Mirandola, R., Mori, M., Müller, H.A., Rouvoy, R., Rubira, C.M.F., Rutten, É., Shaw, M., Tamburrelli, G., Tamura, G., Villegas, N.M., Vogel, T., Zambonelli, F., 2013. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In: Software Engineering for Self-Adaptive Systems. In: Lecture Notes in Computer Science, vol. 9640, Springer, pp. 3–30.

Del Rosso, C., 2006. Continuous evolution through software architecture evaluation: a case study. J. Softw. Maint. Evol.: Res. Pract. 18 (5), 351–383. http://dx.doi.org/10.1002/smr.337, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.337.

Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. Inf. Softw. Technol. 50 (9–10), 833–859. http://dx.doi.org/10.1016/j.infsof.2008.01.006.

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N., 2016. DevOps. IEEE Softw. 33 (3), 94–100. http://dx.doi.org/10.1109/MS.2016.68.

Emam, K.E., 1999. Benchmarking kappa: Interrater agreement in software ProcessAssessments. Empir. Softw. Engg. 4 (2), 113–133. http://dx.doi.org/10.1023/A:1009820201126.

Eramo, R., Tucci, M., Di Pompeo, D., Cortellessa, V., Di Marco, A., Taibi, D., 2021. Replication package "Architectural support for software performancein continuous software engineering: a systematic mapping study". URL https://zenodo.org/record/6298843.

Ferme, V., Pautasso, C., 2017. Towards holistic continuous software performance assessment. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. ICPE '17 Companion. pp. 159–164. http://dx.doi.org/10.1145/3053600.3053636.

Fitzgerald, B., Stol, K.-J., 2017. Continuous software engineering: A roadmap and agenda. J. Syst. Softw. 123, 176–189.

Hasselbring, W., 2018. Software architecture: Past, present, future. In: The Essence of Software Engineering. Springer International Publishing, Cham, pp. 169–184. http://dx.doi.org/10.1007/978-3-319-73897-0_10.

Hasselbring, W., van Hoorn, A., 2020. Kieker: A monitoring framework for software engineering research. Softw. Impacts 5, 100019. http://dx.doi.org/10.1016/j.simpa.2020.100019.

Huang, X., Lin, J., Demner-Fushman, D., 2006. Evaluation of PICO as a knowledge representation for clinical questions. In: AMIA Annu Symp Proc. 2006. pp. 359–363.

Jabbari, R., Ali, N.B., Petersen, K., Tanveer, B., 2016. What is DevOps?: A systematic mapping study on definitions and practices. In: Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, Scotland, UK, May 24, 2016. ACM, p. 12. http://dx.doi.org/10.1145/2962695.2962707.

Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering.

Koziolek, H., 2010. Performance evaluation of component-based software systems: A survey. Perform. Eval. 67, 634–658.

Kudrjavets, G., Thomas, J., Nagappan, N., 2022. The evolving landscape of software performance engineering. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022. EASE '22, pp. 260–261. http://dx.doi.org/10.1145/3530019.3534977.

Laaber, C., 2019. Continuous software performance assessment: Detecting performance problems of software libraries on every build. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 410–414. http://dx.doi.org/10.1145/3293882.3338982.

Martens, A., Koziolek, H., Becker, S., Reussner, R.H., 2010. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proc. of the First Int. Conf. on Perf. Eng. (ICPE '10). pp. 105–116.

Mazkatli, M., Monschein, D., Grohmann, J., Koziolek, A., 2020. Incremental calibration of architectural performance models with parametric dependencies. In: 2020 IEEE International Conference on Software Architecture, ICSA 2020, Salvador, Brazil, March 16-20, 2020. IEEE, pp. 23–34. http://dx.doi.org/10.1109/ICSA47634.2020.00011.

Merseguer, J., Binder, W., Murphy, J., 2017. Guest editorial: Automation in software performance engineering. Autom. Softw. Eng. 24 (1), 71–72. http://dx.doi.org/10.1007/s10515-016-0201-2.

Pahl, C., Jamshidi, P., 2016. Microservices: A systematic mapping study. In: Cardoso, J.S., Ferguson, D., Muñoz, V.M., Helfert, M. (Eds.), CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Volume 1, Rome, Italy, April 23-25, 2016. SciTePress, pp. 137–146. http://dx.doi.org/10.5220/0005785501370146.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: EASE.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18.

Spinner, S., Grohmann, J., Eismann, S., Kounev, S., 2019. Online model learning for self-aware computing infrastructures. J. Syst. Softw. 147, 1–16. http://dx.doi.org/10.1016/j.jss.2018.09.089.

Spinner, S., Walter, J., Kounev, S., 2016. A reference architecture for online performance model extraction in virtualized environments. In: Avritzer, A., Iosup, A., Zhu, X., Becker, S. (Eds.), Companion Publication for ACM/SPEC on International Conference on Performance Engineering, ICPE 2016 Companion, Delft, the Netherlands, March 12-16, 2016. ACM, pp. 57–62. http://dx.doi.org/10.1145/2859889.2859893.

Taibi, D., Lenarduzzi, V., Pahl, C., 2019. Continuous architecting with microservices and DevOps: A systematic mapping study. CoRR abs/1908.10337. arXiv:1908.10337.

Tichy, M., Bosch, J., Goedicke, M., 2017. Editorial. J. Syst. Softw. 123, 173–175. http://dx.doi.org/10.1016/j.jss.2016.09.010, URL https://www.sciencedirect.com/science/article/pii/S0164121216301741.

Weyn, D., Gerostathopoulos, I., Abbas, N., Andersson, J., Biffl, S., Brada, P., Bures, T., Salle, A.D., Lago, P., Musil, A., Musil, J., Pelliccione, P., 2022. Preliminary results of a survey on the use of self-adaptation in industry. In: 2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 70–76. http://dx.doi.org/10.1145/3524844.3528077.

Weyns, D., 2020. An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective. John Wiley & Sons.

Wohlin, C., 2014. 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. http://dx.doi.org/10.1145/2601248.2601268.

Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., Engström, E., do Carmo Machado, I., de Almeida, E.S., 2013. On the reliability of mapping studies in software engineering. J. Syst. Softw. 86 (10), 2594–2610. http://dx.doi.org/10.1016/j.jss.2013.04.076, URL https://www.sciencedirect.com/science/article/pii/S0164121213001234.

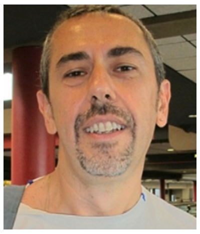
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., 2012. Experimentation in Software Engineering. Springer.

Woodside, M., Franks, G., Petriu, D.C., 2007. The future of software performance engineering. In: Future of Software Engineering (FOSE '07). pp. 171–187. http://dx.doi.org/10.1109/FOSE.2007.32.

Romina Eramo is Assistant Professor in Computer Science at the University of Teramo (Italy). Before moving to the University of Teramo, she has been Assistant Professor at the University of L'Aquila (2018–2022). She had received his Ph.D. in Computer Science at University of L'Aquila in 2011, and she held postdoc positions at the same institution. Her research interests include model-driven engineering, software quality, continuous software engineering, DevOps, and digital twins. She is involved in several program committees of international conferences, reviewing activities and conference organization. She published several articles in journals and proceedings of international events on her topics. She has been working and leading different European and Italian research projects.

Michele Tucci is an Assistant Professor in the Department of Information Engineering, Computer Science and Mathematics at the University of L'Aquila, Italy. From 2021 to early 2023, he was a postdoctoral researcher at the Department of Distributed and Dependable Systems of the Faculty of Mathematics and Physics at Charles University in Prague, Czech Republic. He received his Ph.D. in computer science from the University of L'Aquila, Italy, in 2021, where he was advised by Romina Eramo and Vittorio Cortellessa. His current research interests include performance regression testing, software refactoring, and optimization of software architectures towards quality aspects.

Daniele Di Pompeo is an Assistant Professor (RTD-a) at the University of L'Aquila, Italy. He received his Ph.D. in ICT from the University of L'Aquila in 2019. His research interests include model-based performance analysis, software refactoring, and search-based software engineering. He has been elected as an "Expert of the subject" of Software Quality Engineering since 2019. He is a member of the SPENCER research group. His research activity is currently supported by the European project SoBigData.it.



Vittorio Cortellessa is Full Professor at the Department of Computer Science and Engineering, and Mathematics of University of L'Aquila. He had received his Ph.D. in Computer Science at University of Roma Tor Vergata in 1995. Between 1996 and 1999 he held postdoc positions at the same institution and at European Space Agency. In 2000 and 2001 he has been Research Assistant Professor at the Computer Science and Electrical Engineering Department of West Virginia University. Since 2022 he is at University of L'Aquila. His main research interests are in the areas of Software Performance, Software Reliability, and Model-Driven Engineering. He has served and serves in program committees and editorial boards of conference and journals in the Software Engineering domain. He currently is Co-Chair of the Steering Committee of ACM/SPEC International Conference on Performance Engineering (ICPE).

Antinisca Di Marco is Associate Professor in Computer Science at University of L'Aquila. Her main research topics are Software Quality Engineering, Data Science, Quality (such as fairness, explainability and privacy) in Learning Systems and Bioinformatics. She is involved in several national and international projects on such topics. She is the responsible of the research infrastructure of Territori Aperti project, co-PI of the SoBigData.it project and the director of the INFOLIFE CINI Laboratory node in L'Aquila. Since 2018, she is involved in several actions and projects aiming at improving equal opportunities in STEM (Science, Technology, Engineering and Mathematics). In particular, she is member of the cost action EUGAIN (https://eugain.eu/), and co-ideator and co-coordinator of PinKamP (www.pinkamp.disim.univaq.it).

Davide Taibi is Full Professor at the University of Oulu (Finland) where he head the M3S Cloud research group. His research is mainly focused on Empirical Software Engineering applied to cloud-native systems, with a special focus on the migration from monolithic to cloud-native applications. He is investigating processes and techniques for developing Cloud Native applications and identifying cloud-native-specific patterns and anti-patterns. He has been a member of the International Software Engineering Network (ISERN) since 2018. Before moving to Finland, he has been Assistant Professor at the Free University of Bozen/Bolzano (2015–2017), post-doctoral research fellow at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering - IESE (2013–2014), and research fellow at the University of Insubria (2007–2011).