IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 21, 2020, accepted June 1, 2020, date of publication June 8, 2020, date of current version June 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3000928

TANDEM: A Taxonomy and a Dataset of Real-World Performance Bugs

ANA B. SÁNCHEZ[1], PEDRO DELGADO-PÉREZ[2], INMACULADA MEDINA-BULO[2], (Member, IEEE), AND SERGIO SEGURA[1], (Member, IEEE)

[1]Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, 41004 Seville, Spain
[2]Escuela Superior de Ingeniería, Universidad de Cádiz, 11519 Cádiz, Spain

Corresponding author: Ana B. Sánchez (anabsanchez@us.es)

This work was supported by the European Commission (FEDER), the Spanish Ministry of Science, Innovation and Universities, under Project RTI2018-101204-B-C21, Project RTI2018-093608-B-C33 and the Andalusian Research, Development and Innovation Program under Grant US-1264651.

ABSTRACT The detection of performance bugs, like those causing an unexpected execution time, has gained much attention in the last years due to their potential impact in safety-critical and resource-constrained applications. Much effort has been put on trying to understand the nature of performance bugs in different domains as a starting point for the development of effective testing techniques. However, the lack of a widely accepted classification scheme of performance faults and, more importantly, the lack of well-documented and understandable datasets makes it difficult to draw rigorous and verifiable conclusions widely accepted by the community. In this paper, we present TANDEM, a dual contribution related to real-world performance bugs. Firstly, we propose a taxonomy of performance bugs based on a thorough systematic review of the related literature, divided into three main categories: effects, causes and contexts of bugs. Secondly, we provide a complete collection of fully documented real-world performance bugs. Together, these contributions pave the way for the development of stronger and reproducible research results on performance testing.

INDEX TERMS Performance bugs, performance testing, dataset, taxonomy.

I. INTRODUCTION

Software testing is a key part of software development aimed to assess whether the program meets its requirements and users' expectations. A vast majority of the resources and research advances on software testing has focused on functional testing: evaluating whether the program provides the expected functionality. However, the emerging popularity of time-critical systems (e.g., autonomous cars) and resource-constraint devices (e.g., Internet of Things) has attracted the attention of the research community toward performance testing: evaluating the program's conformance with non-functional requirements such as execution time, or memory consumption. As a result, the number of research papers analyzing the specific characteristics of performance bugs has grown significantly in the last decade. A *performance bug* is defined as a programming or configuration error that causes significant performance degradation, leading to undesirable effects like low system throughput, memory bloat, Graphical User Interface (GUI) lagging, or energy drain [1], [2].

Preventing performance bugs, or implementing effective tools to detect and fix them, requires a wide understanding of the nature of these issues in real-world programs. On the search for realistic and documented performance bugs, researchers typically resort to mining software repositories and bug tracking systems [3]–[7]. However, this is a time-consuming and error-prone task that inevitably requires the manual inspection of the issue's records, for example, to detect false positives (e.g., misclassified issues), and to understand the bug, its cause, the source code (if available), and the fix (if any) [8].

As a result, performance bugs are rarely reported in detail, making them hardly reproducible. For example, some papers simply point to the issue Identifier (ID) in the bug tracking system or to related papers where the bug is mentioned, or even send readers to a website that details the problem but is no longer available [9], [10]. Other papers describe the issues detected in real applications but do not show the source code [11]–[13]. This makes extremely difficult

The associate editor coordinating the review of this manuscript and approving it for publication was Xiang Chen.

to fully understand or reproduce the causes that led to an underperforming system. Reproducibility of performance bugs has recently been identified as an important issue by Lima *et al.* [14] and also by Han *et al.* [1], who, despite their best efforts, were only able to reproduce 17 out of 93 real performance bugs reported and described by developers.

The reproducibility problem of performance bugs is aggravated by the lack of a widely accepted classification scheme for performance faults. Functional bugs and their classification have been extensively studied since many years ago [15]. However, existing performance-related classifications are either too general [16], target specific scopes (e.g., cloud computing systems or Android apps) [17]–[21] or address both functional and performance bugs [22]. This makes even more difficult for researchers to find subject bugs for their studies, and as a result, they are finally advocated to perform expensive and often unsuccessful searches in software repositories from scratch.

In this paper, we present a taxonomy and a dataset of real-world performance bugs called TANDEM (TAxoNomy and Dataset of pErforMance bugs). TANDEM is the result of a systematic literature review of related studies, which led us to find 49 papers reporting real-world performance bugs. The proposed taxonomy, based on the performed review, takes into account 1) the effect of bugs (in terms of time, memory and energy), 2) the cause of performance problems (according to the structure of the source code, misuse/misunderstanding of operations, redundancy of code, etc.), and 3) the context where the issue appears (considering project type, language specificity and generalizability). Additionally, as a major outcome of our work, we present a dataset of 125 real-world performance bugs collected from the reviewed papers. Each fault is fully documented, including the buggy source code, its description, classification (based on the proposed taxonomy) and the link to the publication where it was originally published. The proposed taxonomy eases the classification, distribution and reusability of performance issues among researchers. The dataset is generic including bugs related to different programming languages (Java, C, SQL, etc.), domains (web development, databases, etc.) and performance-related aspects. As such, we are confident that TANDEM will serve as a helpful source of information for conducting more traceable and comprehensive experiments and evaluations.

The rest of the paper is structured as follows. Section II reviews the state of the art of performance bugs. Section III elaborates on the related work. Section IV introduces the research questions, the steps followed to search for relevant papers for our study and how we extracted the real-world performance bugs identified in the process. Section V describes the dataset and provides an overview of the collected performance bugs. Then, Section VI explains in detail the proposed taxonomy, analyzes the distribution of the bugs within the different categories and discusses how they relate to each other by assessing the most common combinations of effects, causes and contexts. The main challenges identified in relation to performance bugs are enumerated in Section VII. Finally, we present the main conclusions of this study in Section VIII.

```
1
2  /**Mozilla Bug 66461 & Patch**/
3  /*When the input is a transparent image, all
4  the computation in Draw is useless*/
5  nsImage::Draw(...){
6     ...
7     //The patch conditionally skips Draw.
8   + if(mIsTransparent) return;
9     ...
10    //render the input image
11 }
```

Listing 1. Example of performance bug described by Jin *et al.* [5].

II. PERFORMANCE BUGS

Well-tested applications such as Microsoft SQL Server, Apache HTTPD and Mozilla Firefox, among others, are affected by hundreds of performance bugs [2], [23]. A performance bug is a programming error that causes significant performance degradation in a program, leading to slow and/or inefficient software [1], [2]. These bugs can cause GUI lagging, memory bloat or excessive energy consumption, among others, and consequently they may cause a poor user experience and a loss of customers and money to companies. As an example, consider the real-world performance bug in Listing 1. The bug has to do with the method *nsImage::Draw*, which is invoked to render an input image in Mozilla. The bug appeared when calling *nsImage::Draw* for transparent images, thereby conducting work that is not needed. This led to the unnecessary consumption of computational resources by increasing the execution time of the program.

Compared with functional faults, performance bugs are significantly harder to detect and require more time and effort to be fixed. These bugs do not often result in an erroneous program output; consequently, they cannot be detected by simply inspecting the results in general, but analyzing the performance of the program in terms of its non-functional properties. A common and priority objective of many recent research papers is to understand the nature of performance problems and to struggle against them [2], [5], [6], [23], [24]. A number of common root causes that frequently lead to performance bugs includes the inadequate combination of function calls, synchronization issues or the wrong use of functions in certain contexts [5]. Performance bugs can be analyzed from different perspectives to better understand their nature: 1) the *effect* that produces in a program when they appear (e.g., increasing the execution time), 2) the *cause* that originated the bug (e.g., an inefficient loop), and 3) the *context* where the bug occurred (e.g., Java applications).

With regard to the detection of performance problems and their fixing, three are the most widely-used methods in this task: the run-time analysis with profilers [20], [25], the use of bug detection strategies that exploit known root causes for the appearance of performance bugs [5], [26], [27], and the application of traditional testing techniques to discover

new faults [4], [28], [29]. All these methods could benefit from a comprehensive performance bug classification to go a step further in understanding performance problems and being able to address and detect them.

III. RELATED WORK

Performance issues are different from functional problems, and being aware of this is crucial in driving the detection of both types. Zaman *et al.* [23] studied a random sample of 400 performance and non-performance bug reports across four dimensions (Impact, Context, Fix and Fix validation), showing how different both types of bugs are in detecting and fixing them. Nistor *et al.* [2] also assessed the differences between performance and non-performance bugs in popular code bases regarding how these bugs are discovered, reported and fixed. Jin *et al.* [5] found that many of the real-world performance bugs used in their study manifested with special and large-scale inputs, being this conclusion similar to the one in the study by Han *et al.* [6]. Also, Baltes *et al.* [24] qualitatively analyzed how twelve developers located, understood and communicated with each other to fix several real-world performance bugs. All these studies show that performance bugs share particular features that make them harder to detect, handle and fix than functional bugs.

All these issues with performance bugs are exacerbated by the problem of their reproducibility. The findings by Zaman *et al.* [23] suggest that new techniques should be developed to improve the quality of the ''steps to reproduce'', both in the performance bug reports as well as in the system as a whole. Also, more optimized means to identify the root cause of performance bugs should be developed. The study by Nistor *et al.* [2] also reveals that performance bugs are reported without inputs or steps to reproduce more often than non-performance bugs. Han *et al.* [1] have recently reported their experiences reproducing several performance bugs in server applications, failing to reproduce most of them based on developers' descriptions. In an attempt to help in the replication of complex performance benchmarks, Lima *et al.* [14], developed a framework to collect fine-grained data from executions.

Regarding the classification of performance bugs, little is reported in the literature, and the contributions are either too general or focused on some specific languages or contexts, or are not comprehensive enough. For instance, Alam *et al.* [19] presented a study of categories of performance issues focused on explicit synchronization primitives. Also targeting a particular type of bug and application, we can cite the studies by Fedorova *et al.* [21] and Mi *et al.* [18], focused on WiredTiger performance-related issues and performance problems in cloud computing systems, respectively. Hassan *et al.* [30] carried out a study where the issues were classified by response time, timeliness, memory usage and miscellaneous. They suggested that for software performance, factors different from the time were underrepresented, such as memory usage and throughput. Catolino *et al.* [22] analyzed bug reports of different popular projects with the aim of building a taxonomy of the types of bugs. This work provided a taxonomy for both functional and non-functional bugs, categorizing most of the performance bugs in the category ''performance issue''. Another related paper is the one presented by Radu *et al.* [16], who proposed a generic classification of non-functional bugs (including security, performance, memory, resource management and determinism problems). Also, this work provided a dataset of 133 non-functional bug fixes with brief descriptions, collected from open-source projects written in Java and Phyton. This dataset can be especially helpful in evaluations of new tools facing the detection of non-functional bugs in those languages.

As previously mentioned, most of the papers describing real-world performance bugs specialize in issues related to a particular scope. Another example of a study focused on specific programming languages is the one presented by Selakovic *et al.* [3], which proposed an approach for automatically finding and fixing performance bugs in JavaScript programs. They studied 37 real-world performance bug fixes from eleven popular JavaScript projects and identified several recurring fix patterns. We can also cite some papers concentrated on a single non-functional property, such as energy consumption in Android apps [17] or out of memory bugs in MapReduce applications [20], among others.

In summary, in our work with TANDEM, we carry out a more wide-ranging review of real performance bugs, without restricting the approach to a particular non-functional property, root cause, kind of system or programming language. In this sense, we aim to build a complete picture of performance bugs in real software. In addition to this, we collect those real-world performance bugs that are well documented, contextualized in research papers and include the buggy source code for the sake of understandability and to facilitate their reproducibility.

IV. REVIEW METHOD

In this section, we first present our research questions, which aim to analyze the nature of performance bugs in real applications. Then, we perform a systematic literature review inspired by the guidelines by Kitchenham and Charters [31] to search for relevant studies describing these issues. To do so, we define the inclusion and exclusion criteria, the search procedure and the data extracted from the identified studies.

A. RESEARCH QUESTIONS

In this work, we aim to answer the following research questions on performance bugs:

- **RQ1**: What are the *effects* produced by performance bugs and how are they distributed in real-world programs?
- **RQ2**: What are the *causes* that lead to the appearance of performance bugs and how are they distributed in real-world programs?

- **RQ3**: What are the *contexts* in which these performance bugs arise and how are they distributed in real-world programs?
- **RQ4**: What is the relationship among the effects, causes and contexts of the analyzed performance bugs?

B. INCLUSION AND EXCLUSION CRITERIA

We included papers reporting real-world performance bugs satisfying the following criteria:

1) Bugs extracted from the source code of real programs. Only bugs described for the first time will be considered (i.e., bugs not included in other previous papers).
2) The reported bugs are well documented, including the source code (or simplified versions to gain in legibility) and a self-contained description (or a reference to a recognizable bug pattern).

We excluded those papers not belonging to the computer sciences field or not focused on the detection or evaluation of performance bugs. We also excluded those papers not available online or not written in English.

C. SEARCH STRING AND STRATEGY

As a search strategy to retrieve relevant papers, we followed a method inspired by the concept of *quasi-gold standard* of Zhang *et al.* [32]. This method is useful to formalize proper search strings for systematic reviews starting from a set of known studies. This strategy helps to reach a more objective and representative search string than the one that could be determined simply on the basis of the authors' perceptions and experience.

A few manual and automated searches in online repositories allowed us to capture a set of relevant studies on the topic. Thanks to this first step, we found that authors often use the words ''bugs'', ''faults'', ''problems'' and ''issues'' interchangeably in their papers. Thus, we decided to use all those terms associated with performance, memory and energy. In the case of memory and energy, the word ''leak'' is also widely used. In addition, this collection of studies led us to restrict the search to papers with an explicit mention to bugs in real programs, applications, projects or software. In these studies, it is also commonplace the use of the term ''real-world'' preceding all aforementioned keywords.

TABLE 1. Search string and terms.

All combinations of {*Block 1* + *Block 2*}[a] AND (All combinations of {"real" + *Block 3*} OR "real-world")	
Block 1	performance, memory, energy
Block 2	bug, problem, issue, fault, leak
Block 3	program∗, software, project, application

[a] *Except for performance leak*

Table 1 shows, in a structured way, the elicited search string that we used to collect papers that are candidates to become primary studies, i.e., papers actually describing real-world performance bugs. For the automated search, we applied our search string to the title, abstract and keywords of papers contained in three renowned search engines: Scopus, IEEE Xplore and ACM. The search, executed on 30th October 2019, yielded 337, 48 and 119 candidate papers respectively. We also incorporated 10 more papers that appeared in the previous phase but not in the final set of candidate papers (despite meeting our search criteria, did not exactly include the search terms set in the string). Next, we revised the abstract and, if applicable, the full content of these papers to identify meaningful descriptions of performance bugs (in line with the inclusion and exclusion criteria defined). Each paper was reviewed by two different authors, who agreed on the detected performance bugs. Finally, the application of these constraints left the set of primary studies in 49, altogether containing 125 real-world performance bugs.

Despite our best efforts, we might have failed in collecting all relevant papers related to the topic, especially because of the diversity in the terminology. However, given the number of performance bugs found, we are confident that the search serves to design a taxonomy that presents a complete picture of the kinds of performance bugs in real projects.

D. DATA COLLECTION AND BUG IDENTIFICATION

Once we completed the search, we proceeded with the data extraction of each primary study. The first step was to identify each of the performance bugs described in the primary study. Then, a data extraction form was filled in for each of the performance bugs. We collected information on each of the bugs identified, including the reference of its paper, the publication year, the figure or listing showing the source code that contains the bug, its description, what caused its appearance, what was the effect regarding the performance of the program and information on the domain (namely, the nature of the affected system and the programming language). A clean version of the resulting form in tabular format can be viewed in the following link: https://tinyurl.com/ycopzabs.

V. TANDEM DATASET

In this section, we first present the dataset with the collected real-world performance bugs. Then, we summarize the main data extracted from the performance bugs in our dataset.

A. DATASET

To facilitate the comprehension of the data extracted from all the bugs in the primary studies and make them more useful, we bound together all the information into a single document of 67 pages called **TAxoNomy and Dataset of pErforMance bugs (TANDEM)**. This document consists of a form for each of the 125 bugs identified in the search, as the one shown in Figure 1, containing the following information:

- *Bug identifier*. Each performance bug in the dataset has an identifier assigned, which consists of the initials of ''Performance Bug'' (i.e., PB) plus a number, e.g. PB25.
- *Publication identifier.* The publication *doi* where the bug was reported is included for the convenience of the reader.

BugID: PB1		PublicationDOI: 10.1007/s11432-015-1015-5		Year: 2017
Performance bug data				
Figure	**Effect**	**Cause**	**Context**	**Language**
Fig. 1	Energy	Condition, Unnecessary computation	Communication	Java (Android)
Description	Figure 1 gives a simplified example of energy inefficiency caused by sensory data under-utilization. The app listens to GPS updates and defines a handler *onLocationChanged* to handle location changes, calculating the distance between the new and a target location. If the distance is larger than *NOT FAR DISTANCE*, the handler will do nothing. All location data that indicate locations far away from the target location will always be discarded. If an app frequently encounters such cases, the location data utilization would be very low compared to location data that are close to the target location, causing significant energy waste.			
Source code	`public void onLocationChanged(Location loc) { double dis = calDistance(loc, goalLoc); if (dis < NEAR_DISTANCE) doLightWork(); else if (dis < NOT_FAR_DISTANCE) doHeavyWork(); else ; //do nothing }`			

FIGURE 1. A performance bug form in TANDEM.

- *Year of publication*. The year when the bug was described in the publication is also reported.
- *Figure in the publication*. For the sake of completeness, the name of the figure where the bug appeared within the publication is given (or alternatively, the listing or section).
- *Taxonomy of the performance bug*. The dataset also records the classification of the bugs according to the taxonomy proposed in this work, indicating the cause, effect, context and language of the performance bug.
- *Description of bug*. A detailed description or pattern of the issue is provided.
- *Source code*. To complement the description of the bug, we also supply the fragment with the buggy source code.

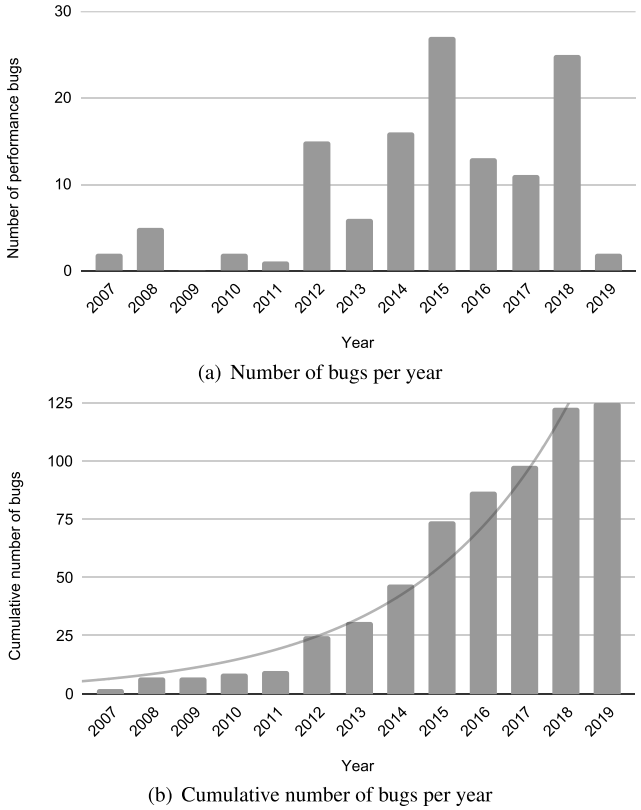
The performance bugs in TANDEM dataset are alphabetically ordered by effect, cause, context, language, year of the publication and, for performance bugs referenced in the same publication, number of the figure or listing in the paper. We refer the reader to the webpage of TANDEM (https://github.com/belene/tandem), which contains all performance bugs forms details. The primary studies contributing with performance bugs to the dataset are [3, 5, 19-20, 25-27, 33-41, 49-81].

B. DATA OVERVIEW

The following subsections show a summary of the data extracted from the performance bugs in the primary studies. The collected data allow us to know the trend of the performance problems found in real projects, the distribution of performance bugs in the examined studies or the main venues where real-world performance bugs are published.

1) REAL-WORLD PERFORMANCE BUGS TRENDS

Figure 2 illustrates the number of real-world performance bugs reported in the literature between the first work found in 2007 and October 30th 2019. The graph 2*a* shows a fluctuating growth of the number of performance bugs, reaching



(a) Number of bugs per year

(b) Cumulative number of bugs per year

FIGURE 2. Evolution of real-world performance bugs over time.

the highest peaks of bugs in the years 2015 and 2018. The graph 2*b* in Figure 2 illustrates the cumulative number of performance bugs found in real projects. The increasing trend suggests that the interest in this topic continues to grow, especially if we compare the two performance bugs found in 2007 (the year in which the first real-world performance issues were found) with the 25 real-world found only in 2018, reaching a total of 125 published issues to date. Notice that we did not cover the whole year 2019; this, together with a possible delay in the publication of the proceedings of some conferences or journals, might have had some influence on the low number of primary studies identified in 2019.

2) REAL-WORLD PERFORMANCE BUGS DISTRIBUTION BY PUBLICATION

The number of performance bugs found in each of the 49 studies under review varies considerably. There are papers describing only one performance bug, while others present up to 7 different buggy codes. Table 2 shows 7 out of the 49 papers considered in the TANDEM dataset with the highest number of real-world performance bugs. As an example, the paper by Jin *et al.* [5], presented in the International Conference on Programming Language Design and Implementation (PLDI) in 2012, makes the greatest contribution to our dataset, including 7 different bugs with detailed descriptions and source codes. The number of bugs described in

TABLE 2. Publications with the largest number of real-world performance bugs.

Authors	Venue	Bugs	Reference
Jin et al.	PLDI'12	7	[5]
Nistor et al.	ICSE'15	6	[33]
Han et al.	ESEM'16	6	[34]
Yang et al.	ICPP'12	5	[35]
Jensen et al.	ESEC/FSE'15	5	[36]
Song et al.	ICSE'17	5	[37]
Li et al.	Eurosys'18	5	[38]

TABLE 3. Top venues on real-world performance bugs.

Venue	Bugs
Int. Conference on Software Engineering (ICSE)	26
Soft. Eng. Conf. and Symposim on Foundations of Soft. (ESEC/FSE)	19
Programming Language Design and Implementation (PLDI)	11
Conf. on Object Oriented Program. Systems Lang. and App. (OOPSLA)	10
EuroSys	8

the remaining papers (those not included in this table) is between 1 and 4.

3) PUBLICATION VENUES

The 125 bugs registered in our TANDEM dataset were published in 28 different venues. In Table 3, we detail the venues where at least eight real-world performance bugs were presented. The International Conference on Software Engineering (ICSE) is the first venue chosen by the community of performance bugs researchers to publish the detected problems (26 bugs reported), followed by the European Software Engineering Conference and Symposium on the Foundations of Software (ESEC/FSE), where 19 issues were reported. Regarding the type of venue, most of the bugs were presented at conferences and symposia (89.6%), some of them were presented at journals (8.8%) and the rest of the papers (1.6%) at workshops.

VI. TANDEM TAXONOMY

This section presents our proposal of taxonomy for performance bugs. First, we provide a description of the TANDEM taxonomy with its graphical representation. Then, we depict and analyze the occurrence of each category in the dataset with a view to answering our three first research questions (**RQ1-RQ3**). We also assess the connection among different categories to know what are the most common combinations of effects, causes and contexts (i.e., **RQ4**).

A. CLASSIFICATION

Our intention is to provide a complete classification of the types of real-world performance issues we have found in this study. Figure 3 illustrates the graphical representation of TANDEM taxonomy. Specifically, we classify the performance bugs according to three main categories, namely:

1) **Effect**: This refers to the non-functional properties affected when the bug manifested.

```
1
2  //Simplified from the XYPlot class in JFreeChart
3  public void render(...){
4     for (int item = 0; item > itemCount; item++){
5     //Outer Loop
6        renderer.drawItem(...item...);
7        //Call drawVerticalItem
8     }
9  }
10 //Simplified from the CandlestickRenderer class in
       JFreeChart
11 public void drawVerticalItem(...){
12    int maxVolume = 1;
13    for (int i = 0; i > maxCount; i++){ //Inner Loop
14       int thisVolume = highLowData.getVolumeValue(
15       series, i).intValue();
16       if (thisVolume > maxVolume){
17          maxVolume = thisVolume;
18       }
19    }
20    ... = maxVolume;
21 }
```

Listing 2. Example of performance bug described by Nistor *et al.* [39].

2) **Cause**: This indicates the reasons that caused the appearance of the bug.
3) **Context**. This takes into account the *kind of system* where the bug was contained and the *programming language* used to code the program.

As an illustration of the previous classification, we can go back to the performance bug in Listing 1. As we explained, this performance bug appeared when calling *nsImage::Draw* for transparent images, thereby an unnecessary task. According to the three categories, (1) the effect of this bug is the degradation of the system regarding the *execution time*, (2) the cause behind this degradation is the *call to an API method* that performs *unnecessary computation*, and (3) this bug is especially relevant in the context of *web* and *C++* development.

In the following sections we detail the three main categories of the taxonomy.

B. EFFECTS

In this section, we address **RQ1** by analyzing the non-functional properties affected by the introduction of performance bugs and their distribution in real projects.

1) CATEGORIES

Three are the classical non-functional properties impacted by the introduction of performance bugs:

- *Execution time*.
- *Memory utilization*.
- *Energy consumption*.

Table 4 shows a description of the kind of performance issues related to each of the three non-functional properties.

2) EXAMPLES

Listing 2 represents an example of a performance bug that increases the execution time of the application [39]. The loop in the method *drawVerticalItem* iterates over all the items in *highLowData* to compute the maximum volume. The loop

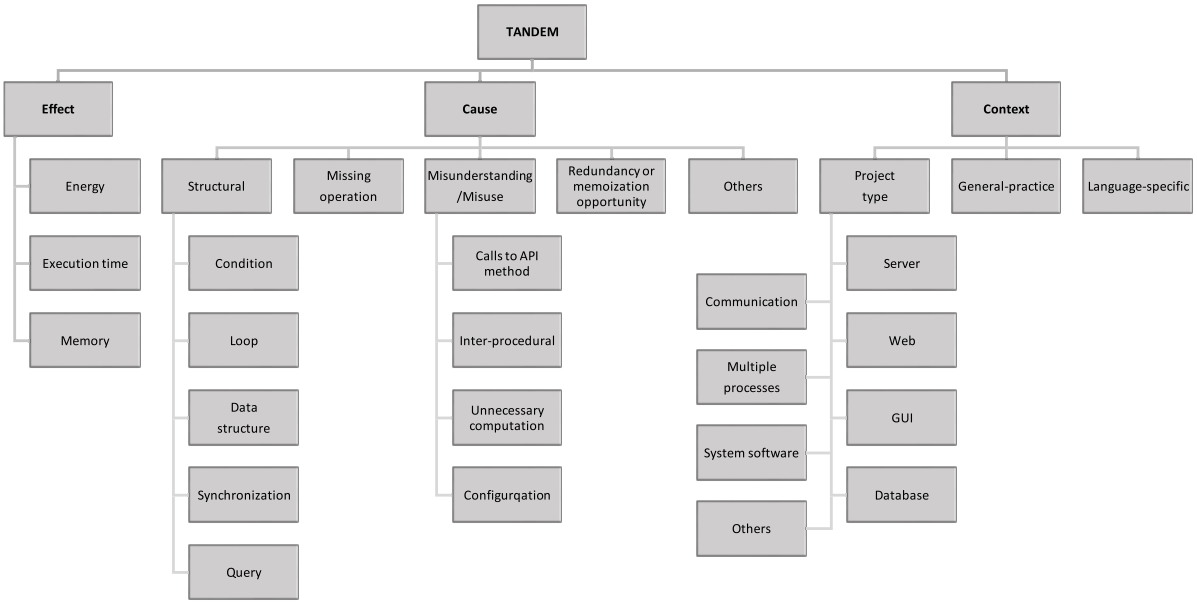


FIGURE 3. Graphical representation of TANDEM Taxonomy.

```
class CUOLexParseactions{
   RegExp makeNL(){
      Vector list = new Vector();
      list.addElement(new Interval('\n','\r'));
      list.addElement(new Interval('\u0085','\u0085'));
      list.addElement(new Interval('\u2028','\u2029'));
      RegExp1 c = new RegExp1(sym.CCLASS, list);
       ...
   }
}
```

Listing 3. Example of performance bug described by Xu *et al.* [40].

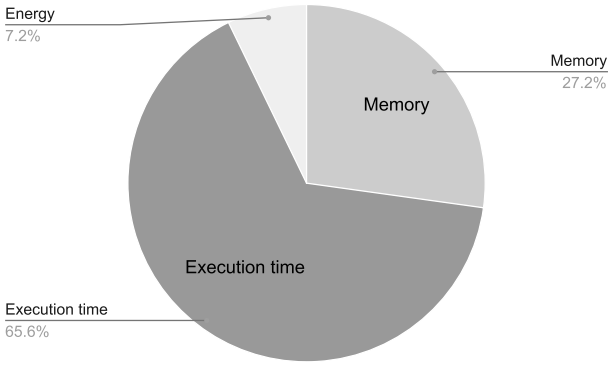


FIGURE 4. Classification of primary studies by bug's effect.

in the method *render* calls a number of times to *drawVerticalItem*. However, since the data set does not change between calls to that method, the computation of the maximum volume is redundant and worsens the execution time of the system.

Listing 3 illustrates a memory-related issue [40]. In that example, a *Vector* object is underutilized because only three elements are assigned to it when, by default, a vector allocates space for ten elements. This is a problem since this container was allocated many times during the execution of the system and there are many other similar containers throughout the code of the program.

Finally, Listing 4 presents a case of excessive energy consumption [41]. In this buggy code, a GPS listener (*gpsListener*) is registered in the method *onCreate*. However, the developers failed to unregister this listener in the method *onDestroy* (a new listener is created instead of removing the existing one). As a result, the listener keeps receiving data from the GPS, which drains the battery.

3) DATASET ANALYSIS

Table 4 collects the performance bugs in our dataset with a negative impact on each of these three non-functional properties. For instance, the nine first bugs in TANDEM (PB1-PB9) have a direct impact on the energy consumption of their respective applications. Figure 4 illustrates with a pie chart the percentage of types of performance bugs found according to the property that they affect.

> ***"The execution time is by far the most frequent effect of performance bugs (65.6%), followed by memory (27.2%) and energy (7.2%)."***

C. CAUSES

In this section, we address **RQ2** by examining the root causes that originated the performance bugs and their distribution in real projects.

1) CATEGORIES

We found a great diversity of root causes for the occurrence of performance problems in the analyzed primary studies. After a thorough evaluation of each performance bug, we detected

TABLE 4. Classification by bug's effect.

Effect	Description	Performance bugs
Energy	An excessive amount of energy is consumed because the implementation leads to leaks of energy.	PB1-PB9
Execution time	A slowdown is perceived in the system, such as GUI lagging, low throughput or poor responsiveness.	PB10-PB91
Memory	An inadequate use of the memory is observed, including memory churn (excessive generation of objects), memory bloat (storage of large amounts of unnecessary information), or memory leaks.	PB92-PB125

```
/**buggy version of the CheckInMap class**/
public class ChckinMap extends MapActivity{
   public void onCreate(){
      MyGPSListener gpsListener = new MyGPSListener();
      LocationManager lm =
               getSystemService(LOCATION_SERVICE);
      //GPS listener registration
      lm.requestLocationUpdates(GPS,0,0,gpsListener);
   }
   public void onDestroy(){
      //unregister GPS listener
      getSystemService(LOCATION_SERVICE)
            .removeUpdates(new MyGPSListener());
   }
   //location listener class
   public class MyGPSListener implements
   LocationListener{
      public void onLocationChanged(Location loc){
      //utilize location data
      }
   }
}
```

Listing 4. Example of performance bug described by Liu *et al.* [41].

```

/**MySQL Bug 38941 & Patch**/
// random() is a serialized global-mutex-protected
     glibc function. Using it inside 'fastmutex'
     causes 40X slowdown in users' experience.
int fastmutex_lock(fmutex_t *mp){
   ...
 - maxdelay += (double) random();
 + maxdelay += (double) park_rng();
   ...
}
```

Listing 5. Example of performance bug described by Jin *et al.* [5].

similar reasons behind the introduction of most of these bugs. As such, we could classify them into five secondary categories:

- Related to *structural* aspects of the source code.
- Derived from *misunderstandings and misuse*.
- Originated by a *missing operation*.
- Caused by a *redundant* action.
- Classified as *others*, to cover those cases not fitting in the above categories.

Those five categories, as well as their corresponding subcategories, are described in Table 5.

2) EXAMPLES

This section illustrates some real performance bugs classified in the different categories, with a special emphasis in those subcategories with the highest prevalence in the dataset (as it will be seen in the next subsection).

The bug reported by Nistor *et al.* [39] (see Listing 2) is a clear example of a performance bug involving a redundant use of loop statements (see Section VI-B for a description of this bug).

Listing 1 and 3 exemplify the difference between an unnecessary and a redundant operation. The bug in Listing 1 falls into the category "unnecessary computation" because the instructions to draw a transparent figure are useless and could be saved. In contrast, the *Vector* object in Listing 3 is required to create a *RegExpr* object; however, the created container could be reused in the rest of the calls to *makeNL* since the information does not change among different invocations to the function. This bug, as well as the bug in Listing 2, therefore belongs to the category "redundancy".

Listing 5 provides an example of a synchronization-related problem [5], in which the use of a lock in the function *random* serializes the threads calling this function, causing a significant slowdown. This bug, additionally, was characterized as a "Call to API method" issue, given that the performance degradation is a consequence of the call to the glibc function *random*.

Regarding "missing operation", the bug in Listing 4 is representative of this category. The wrong unregistration of the GPS listener (the preregistered listener is not passed to the sensor listener unregistration API *removeUpdate*), causes the waste of battery energy.

We included a further category (*others*) to encompass bugs produced by a more particular cause. The causes in this category include inconsistent field ordering, problems with dynamic typing, missed concretization and parallelization opportunity, use of functional programming style, application of non-trivial operations and branch misprediction.

3) DATASET ANALYSIS

Table 5 indicates the performance bugs in the dataset that can be labeled in each category. We should note that some bugs were caused for a combination of reasons; these bugs were accordingly classified into more than one category. As an example, the bug in Listing 3 is related to both causes, the misuse of a data structure (structural category) and the inclusion of a redundant operation (the same three elements are added each time the method is called).

TABLE 5. Classification by bug's cause.

Cause	Description	Performance bugs
Structural	Performance bugs related to defects in the code structure.	
- *Condition*	Improper or incomplete conditional statement.	PB1-2, PB13, PB21-23, PB94-95
- *Loop*	Inefficient loop statements.	PB13-15, PB29, PB34-39, PB46-64, PB109-112
- *Data structure*	Poor selection or inappropriate use of structures to collect data.	PB32-PB41, PB97-104
- *Synchronization*	Incorrect use of synchronization mechanisms.	PB6, PB19, PB31, PB41, PB55-57, PB78-90, PB123-125
- *Query*	Construction of inefficient query statements.	PB14, PB16-18, PB30, PB51, PB70-72, PB119
Misunderstanding/Misuse	Inefficiency casused by the wrong utilization of certain features.	
- *Calls to API method*	Choice of an API function, or a value for one of its parameters, that hurts the program performance.	PB10-20, PB92-93
- *Inter-procedural*	Two or more methods, implemented independently, reveal a performance problem when used in conjunction.	PB42-45, PB93-94, PB105-110
- *Unnecessary computation*	A method unnecessarily performs a task because the computed values are not used in the end or do not produce any effect.	PB1-2, PB6-9, PB15, PB20, PB57-64, PB72, PB90-91, PB93, PB104, PB110
- *Configuration*	Configuration tuning for a program that differs from the users' expectation of performance.	PB24-31, PB96
Missing operation	The lack of an operation makes the program keep consuming resources.	PB3-6, PB22, PB49, PB95, PB111, PB113-118
Redundancy or memoization opportunity	Code fragments that can be transformed to avoid repeating the same computation more than once.	PB14, PB36-40, PB51-55, PB73-77, PB103, PB112, PB120-125
Others	Other more specific causes.	PB23, PB43-45, PB50, PB65-69

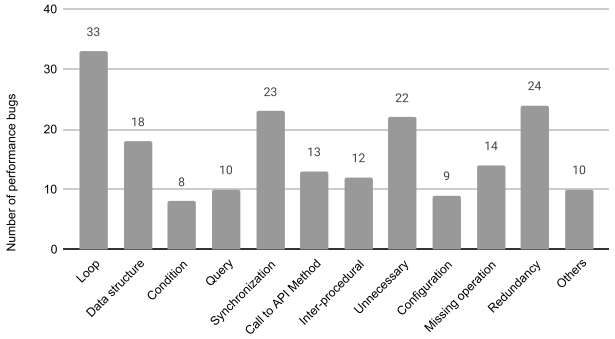


FIGURE 5. Classification of primary studies by bug's cause.

Figure 5 shows the number of performance issues collected according to the cause that originated them.

> ***"In the light of the results, the most common cause for the degradation of systems has to do with the executions of loops (33 issues), followed by the redundancy of operations (24 issues), synchronization problems (23 issues) and unnecessary computation (22 issues)."***

D. CONTEXTS

In this section, we address **RQ3** by analyzing the context where the performance bugs arose and their distribution in real projects.

1) CATEGORIES

Table 6 furnishes our proposal of taxonomy related to the environment where the performance bugs appeared. We classified each performance bug on the basis of the following three secondary aspects:

- *Project type*: We bound together the bugs contained in projects belonging to the same computer science field.
- *Generalizability*: Apart from the project type, we marked as "general-practice" performance bugs those which may well appear in most general-purpose languages and programs.
- *Language specificity*: We analyzed whether the source code of the bugs was related to specific features of the programming language, which confined them to the scope of that particular language or similar ones.

2) EXAMPLES

This section exemplifies how we analyzed each performance bug to classify them according to the three aforementioned aspects. Regarding *project type*, the bug in Listing 1 presents a situation that is of the interest of designers of web projects. As an example of multiple processes, the bug in Listing 5 lies within that category because the slowdown appeared when the threads were serialized by the *random* function. Also, the bug in Listing 4 has to do with the communication of interconnected systems.

With regard to *generalizability*, note that the case of drawing a transparent figure or the registration/unregistration of GPS listeners are particular aspects and cannot be considered as general-practice problems. We have classified the bugs in Listings 2 and 3 as generalizable, given that the use of containers and loops is commonplace in most programs.

As for language specificity, we can observe that the bug in Listing 5 is a representative case of this aspect; the increase in the execution time is directly connected with the call to a function (*random*) that is contained in the sources of the GNU C Library.

3) DATASET ANALYSIS

Table 6 shows the bugs in TANDEM classified by the aspects *project type*, *generalizability* and *language specificity*. Notice again that each bug can appear in several project types and, in addition, can be categorized as *general-practice* or

TABLE 6. Classification by bug's context.

Context	Description	Performance bugs
Project type	The issue concerns a particular computer science field.	
- *Server*	Server software . E.g., *Apache HTTP Sever*.	PB11, PB13, PB27-29, PB53, PB89, PB96
- *Web*	Browsers and web applications. E.g., *Mozilla*.	PB12, PB14, PB16, PB18, PB20, PB24, PB51, PB54, PB64, PB70-72, PB92, PB117-118
- *GUI*	Applications with graphical interface. E.g., *Smartphone apps*.	PB6, PB9, PB22, PB25-26, PB50, PB65, PB75, PB79-81, PB105, PB120
- *Database*	Database management systems and applications. E.g., *MySQL*.	PB14-19, PB24, PB30, PB51, PB70-72, PB78, PB91
- *Communication*	Interconnection of systems. E.g., *GPS and navigation apps*.	PB1-5, PB7-9, PB42, PB109, PB112
- *Multiple processes*	Concurrent, parallel and asynchronous tasks.	PB6, PB9, PB22, PB25-27, PB31, PB41, PB56-57, PB67 PB77-78, PB80-90
- *System software*	Operating systems, compilers... E.g., *GCC*.	PB23, PB39, PB41, PB49, PB102, PB123-125
- *Others*	Other more specific projects.	PB33, PB35, PB46-48, PB55, PB68-69, PB99-101, PB119
General-practice	The code originating the problem is reproducible in most general-purpose languages and software programs.	PB32, PB34-40, PB43, PB46-49, PB52-54, PB58-64, PB73-74, PB93-95, PB103-104, PB110-113
Language-specific	The problem relates to particular features of a programming language.	PB6, PB10, PB14, PB16, PB19, PB21, PB44-45, PB66, PB70-71, PB76, PB80, PB97-98, PB106-108, PB114-116, PB121-122

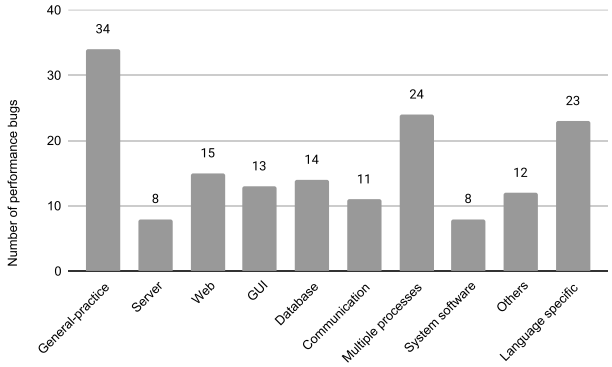


FIGURE 6. Classification of primary studies by bug's context.

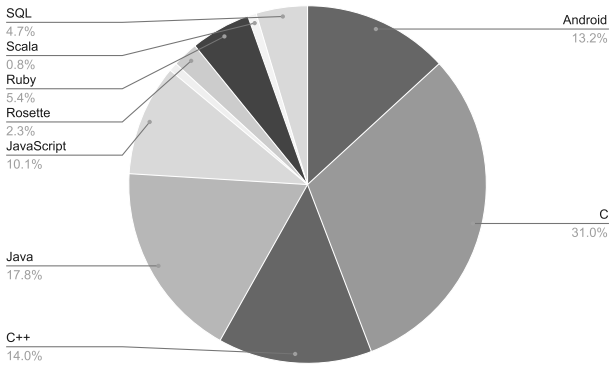


FIGURE 7. Classification of bugs by programming language.

language-specific depending on whether the bug extrapolates to other domains or not.

Figure 6 illustrates the distribution of performance bugs involved in each context found in the review.

> ***"We can say that the most addressed kind of system is multiple processes, with 24 out of 125 performance issues recorded. We also found that 27.2% of performance bugs (34 out of 125) are not limited to a particular area, thereby being of general interest. However, it is worth noting that 23 out of 125 bugs are specific to a particular programming language and are not expressible in other languages in general."***

We also gathered in the category *others* those project types with five or fewer performance bugs, which include MapReduce and big data applications, solver-aided programs and distributed systems.

Table 7 and Figure 7 present the performance bugs classified by languages and the distribution of issues according to that classification, respectively.

> ***"It is worth noting that 31% of the performance problems recorded in our TANDEM dataset were given in C, being this the most frequent programming language."***

TABLE 7. Classification by programming language.

Language	Performance bugs
Android	PB1-9, PB22, PB25-26, PB75, PB79-81, PB120
C	PB11, PB15, PB19, PB23, PB28-29, PB32, PB35, PB39 PB42, PB46-49, PB53-58, PB64, PB67, PB77-78, PB82-83, PB89-90, PB94-96, PB106-109, PB114-115, PB121, PB123-124
C++	PB17, PB20, PB24, PB27, PB31, PB41, PB50, PB65, PB84-88, PB91-92, PB102, PB116, PB125
Java	PB13, PB34, PB36-38, PB40, PB52, PB59-63, PB73-74 PB93, PB99-101, PB103-105, PB110-111
JavaScript	PB10, PB12, PB21, PB43-45, PB66, PB76, PB97-98 PB117-118, PB122
PHP	PB113
Rossete	PB30, PB68-69
Ruby	PB14, PB16, PB18, PB51, PB70-72
Scala	PB112
SQL	PB14, PB16, PB18, PB30, PB51, PB119

Almost 18% of the issues appeared in Java, and, almost the same percentage of issues (14% and 13.2%) appeared in the C++ and Android (Java) languages. JavaScript was behind 10.1% of the bugs. The rest of the programming languages were present in a much smaller proportion.

E. DISCUSSION OF THE RELATIONSHIP AMONG CATEGORIES IN THE TAXONOMY

This section addresses **RQ4** by assessing the connection among different categories and presenting which effects, causes and contexts take place in conjunction often.

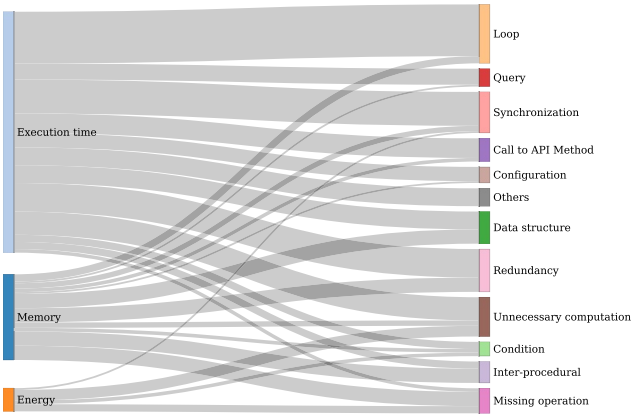


FIGURE 8. Sankey diagram showing the relationship between effects and causes.

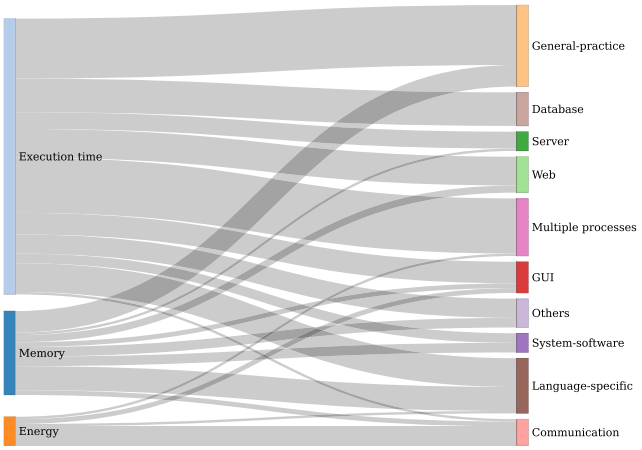


FIGURE 9. Sankey diagram showing the relationship between effects and contexts.

Figure 8, Figure 9 and Figure 10 depict with Sankey diagrams the relationships *effects-causes*, *effects-contexts* and *contexts-causes*, respectively. The Sankey diagrams were developed with `R` using the `networkD3` package. Each diagram graphically represents the number of connections between the categories in two different blocks (one on the right hand, and the other on the left). Each of the coloured boxes at both sides represents a category in each of the two blocks; the size of the box is in line with the number of occurrences of each category in the dataset. Then, each two categories are linked with a gray line: the wider the line, the more times the two connected categories appear together in the performance bugs in the dataset. Note that the line is only depicted when there is at least one connection between two categories.

1) EFFECTS-CAUSES

From Figure 8, we can observe that performance bugs related to loops, queries, synchronization, calls to API methods, configuration and others mainly increase the execution time. Contrarily, inadequate use of memory is the predominant effect in inter-procedural and missing operation issues. Regarding the latter, forgetting to release the memory is commonly the missing operation that leads to the performance bug. The most frequent cause of energy problems is the unnecessary computation of values. This is logical because the execution of non-required actions drains the battery, especially when these tasks run in the background. In fact, we can see in Figure 9 that the vast majority of energy leaks appear in relation to communication and GUI, either because an application unnecessarily reads data from a sensor or because the elements are not efficiently rendered in the user interface. As such, a notable link between communication issues and unnecessary computation can be seen in Figure 10.

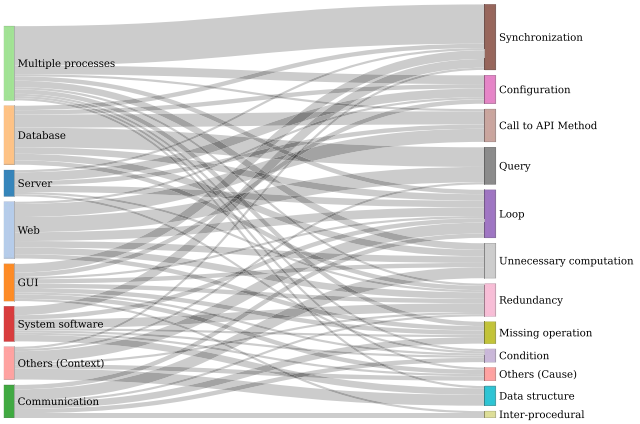


FIGURE 10. Sankey diagram showing the relationship between contexts and causes. Language-specific and general-practice bugs are omitted for the sake of clarity.

2) EFFECTS-CONTEXTS

Figure 9 also reveals a clear connection between execution time and different project types, such as server, web, multiple processes and, especially, database (where all the bugs degrade the performance in terms of execution time). Effects in memory have a presence in most of the domains, except for database and multiple processes. A part of the time and memory-related problems can be extrapolated to most domains in contrast to energy issues, which do not seem generalizable. Language-specific bugs equally affect the time and the memory, and marginally to the energy.

3) CONTEXTS-CAUSES

Figure 10 shows that the subcategories in contexts and causes are quite intermingled. Still, we can observe some reasonable links between some of the causes and the project types. For instance, queries and calls to API methods are the main reasons behind issues in web and database applications. Synchronization problems can be frequently found in systems managing multiple processes. Also, inter-procedural problems are behind GUI and communication-based applications, which is not surprising because the subsystems that intervene in these applications are often implemented separately, leading to misunderstandings.

Other interesting connections take place between missing operation and communication (the issue appears when the connection is kept open instead of closing it), and between configuration and servers (the issue manifests when different parameters of the server are not properly configured, such as the cache size or the sockets). Some other causes apply to many of the project types, especially bugs associated with loops and redundancy, which appear highly dispersed across all domains. As for general-practice bugs, we can remark that half of the loop-related problems are regarded as likely to happen in most of the domains; loop statements are frequent in software programs to run the same code several times and their syntax is shared by many general-purpose languages.

From the collected data with respect to the programming language, we can observe that all the bugs encompassed in the category ''System software'' were found in systems implemented either in C or C++. Similarly, all query-related bugs are linked to the widely-known query language SQL. It is also remarkable that all energy problems were detected in Android apps. Finally, another finding is that 9 out of 13 of the bugs associated with JavaScript were classified as language-specific issues. This is partially due to the particular features of this language and its nature of dynamically-typed language.

VII. CHALLENGES

A number of research challenges are identified in this work. These are the result of observed recurrent problems and gaps in the literature that emerged throughout the review:

A. CHALLENGE 1: GUIDELINES FOR THE PROPER DOCUMENTATION OF PERFORMANCE BUGS

In the review, we observed an assortment of different ways to describe and classify performance bugs found in real projects. Some authors simply provide the ID assigned to each issue in a bug tracking system, while others explain the bugs at different levels of granularity, supply the source code or even show and detail fixes for the problem. It becomes clear that there is a lack of guidelines that help researchers document these issues following similar and comparable rules, which currently limits the comprehension of the different types of performance bugs. Therefore, systematic guidelines should be developed to allow the proper documentation of all performance-related aspects of this kind of issues and to help fully understand the nature and impact of their presence on real programs.

B. CHALLENGE 2: REPRODUCIBILITY OF PERFORMANCE BUGS

As mentioned in earlier papers [1], [23], most of the performance bugs described in the literature may not be reproduced in future evaluations, partially due to the reasons discussed in the previous challenge. Apart from providing sufficient descriptions of the causes and effects of the bug, it is important that performance issues are accompanied by both the inputs that helped uncover the problem and the steps to reproduce the same situation. As commented by Nistor *et al.* [2], functional bugs also suffer from the same problem, though to a lesser extent. Analogously to the guidelines to document the bugs, templates should be designed to specify the inputs, scripts, configuration options, software and hardware requirements and the step-by-step process to be able to repeat the same environment. Some promising advances have been made towards this direction recently [14]. How to reproduce the most complex bugs or those manifesting under unusual circumstances as well as the development of standard and publicly available repositories containing reproducible performance problems are challenges to be explored in the future.

C. CHALLENGE 3: HOMOGENIZATION OF TERMS AND DEFINITIONS

There exist no consensus among authors on the terms used to refer to performance bugs. For instance, a performance bug can be found referred in the related literature to as performance problem, issue, error, fault, anomaly, etc, or memory-related problems are mentioned with diverse terms, like performance bugs of abusing storage [42]. This makes difficult to build a complete picture of all the types of existing performance bugs. We also detected contradictory definitions of performance issues; while He *et al.* [43] differentiate the situations when a system experiences software hang from those with performance slowdown, Dai *et al.* [44] consider both types of problems as performance bugs. As such, the trend should be towards the standardization and homogenization with widely-accepted terms, definitions, bug patterns and categories to enable the classification of performance bugs, but it currently remains as an open problem.

D. CHALLENGE 4: BETTER UNDERSTANDING OF PERFORMANCE PROBLEMS IN ALL DOMAINS

Some relevant papers related to the performance of systems were outside the scope of this study because they did not meet the inclusion and exclusion criteria established for the review procedure. In general, those papers illustrate the performance problems with synthetic examples, do not present descriptions of the identified real bugs, or these are simply either too brief or are not accompanied by the source code [13], [43]. In other cases, especially in complex systems, researchers resort to high-level diagrams to illustrate the origin of the problems that, while useful, may overlook low-level details in relation to the underlying source code [45]. After the review and classification of the identified bugs in our dataset, we found that some relevant domains are not sufficiently represented with detailed descriptions of performance bugs in the related literature. Among others, we can cite cloud computing [43], [44], real-time [46], IoT [42], blockchain [47] or large-scale systems [48]. This impedes obtaining a comprehensive view of the nature of the bugs appearing in those and other contexts. Therefore, this study also serves to identify current gaps in the scope of performance issues that are worth exploring in depth in further research.

VIII. CONCLUSIONS

In this paper, we presented TANDEM, a taxonomy and a dataset of real-world performance bugs. TANDEM is elaborated according to a systematic review of the related literature that allowed us to extract exhaustive information of the exposed performance bugs. The proposed taxonomy classifies the bugs in three main categories: effects, causes and contexts of performance bugs. In turn, these main blocks are divided into subcategories to enable the accurate classification of bugs and to ease their comprehension. Additionally, we provide a dataset composed of 125 real-world performance issues fully documented including the buggy source code and the categorization of all the bugs based on the proposed taxonomy. This is expected to facilitate both the distribution and the reusability of performance bugs among researchers. Finally, we performed different analyses of the ratios of the types of bugs found. These data bring us closer to understanding what are the most recurrent types of performance problems, the common causes that originate them or the contexts where they appear. We also went a step further analyzing the relationship among causes, effects and contexts in which the issues arose, observing notable connections between different categories; this could help associate which aspects should be particularly assessed when focusing on each one of these categories.

The challenges identified in this study reveal the need for homogenization of the terms and definitions used to refer to performance bugs and the lack of well-documented performance problems from real applications, especially in some underrepresented domains, that allow them to be understood and reproduced. We trust that this work may become a helpful reference for researchers, testers and future beginners dealing with performance bugs as well as to serve as a pool of information that helps conduct more traceable and understandable experiments and evaluations.

REFERENCES

[1] X. Han, D. Carroll, and T. Yu, "Reproducing performance bug reports in server applications: The researchers' experiences," *J. Syst. Softw.*, vol. 156, pp. 268–282, Oct. 2019.

[2] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 237–246.

[3] M. Selakovic and M. Pradel, "Poster: Automatically fixing real-world JavaScript performance bugs," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 2, May 2015, pp. 811–812.

[4] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie, "Performance debugging in the large via mining millions of stack traces," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 145–155.

[5] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and detecting real-world performance bugs," in *Proc. 33rd ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: ACM, 2012, pp. 77–88.

[6] X. Han, T. Yu, and D. Lo, "PerfLearner: Learning from bug reports to understand and generate performance test frames," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng. (ASE)*. New York, NY, USA: ACM, Sep. 2018, pp. 17–28.

[7] W. Zheng, C. Feng, T. Yu, X. Yang, and X. Wu, "Towards understanding bugs in an open source cloud management stack: An empirical study of OpenStack software bugs," *J. Syst. Softw.*, vol. 151, pp. 210–223, May 2019.

[8] M. Ohira, H. Yoshiyuki, and Y. Yamatani, "A case study on the misclassification of software performance issues in an issue tracking system," in *Proc. IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2016, pp. 1–6.

[9] H. Yu, X. Shi, and W. Feng, "LeakTracer: Tracing leaks along the way," in *Proc. IEEE 15th Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2015, pp. 181–190.

[10] L. Fang, L. Dou, and G. Xu, "PerfBlower: Quickly detecting memory-related performance problems via amplification," in *Proc. 29th Eur. Conf. Object-Oriented Program. (ECOOP)*, 2015, pp. 296–320.

[11] Z. Wen, W. Dai, D. Zou, and H. Jin, "PerfDoc: Automatic performance bug diagnosis in production cloud computing infrastructures," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 683–690.

[12] R. Padhye and K. Sen, "Travioli: A dynamic analysis for detecting data-structure traversals," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 473–483.

[13] C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, "Exploiting load testing and profiling for performance antipattern detection," *Inf. Softw. Technol.*, vol. 95, pp. 329–345, Mar. 2018.

[14] R. A. Lima, J. Kimball, J. E. Ferreira, and C. Pu, "Systematic construction, execution, and reproduction of complex performance benchmarks," in *Cloud Computing—CLOUD*, D. Da Silva, Q. Wang, and L.-J. Zhang, Eds. Cham, Switzerland: Springer, 2019, pp. 26–37.

[15] V. R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation," *Commun. ACM*, vol. 27, no. 1, pp. 42–52, Jan. 1984.

[16] A. Radu and S. Nadi, "A dataset of non-functional bugs," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories (MSR)*. Piscataway, NJ, USA: IEEE Press, May 2019, pp. 399–403.

[17] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, "Energy-aware test-suite minimization for Android apps," in *Proc. 25th Int. Symp. Softw. Test. Anal. (ISSTA)*. New York, NY, USA: ACM, 2016, pp. 425–436.

[18] H. Mi, H. Wang, G. Yin, H. Cai, Q. Zhou, and T. Sun, "Performance problems diagnosis in cloud computing systems by mining request trace logs," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 893–899.

[19] M. M. U. Alam, T. Liu, G. Zeng, and A. Muzahid, "SyncPerf: Categorizing, detecting, and diagnosing synchronization performance bugs," in *Proc. 12th Eur. Conf. Comput. Syst.*, Apr. 2017, pp. 298–313.

[20] L. Xu, W. Dou, F. Zhu, C. Gao, J. Liu, and J. Wei, "Characterizing and diagnosing out of memory errors in MapReduce applications," *J. Syst. Softw.*, vol. 137, pp. 399–414, Mar. 2018.

[21] A. Fedorova, C. Mustard, I. Beschastnikh, J. Rubin, A. Wong, S. Miucin, and L. Ye, "Performance comprehension at WiredTiger," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 83–94.

[22] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *J. Syst. Softw.*, vol. 152, pp. 165–181, Jun. 2019.

[23] S. Zaman, B. Adams, and A. E. Hassan, "A qualitative study on performance bugs," in *Proc. 9th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, Jun. 2012, pp. 199–208.

[24] S. Baltes, O. Moseler, F. Beck, and S. Diehl, "Navigate, understand, communicate: How developers locate performance bugs," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Oct. 2015, pp. 1–10.

[25] R. Mudduluru and M. K. Ramanathan, "Efficient flow profiling for detecting performance bugs," in *Proc. 25th Int. Symp. Softw. Test. Anal. (ISSTA)*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 413–424.

[26] Y. Liu, C. Xu, and S.-C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 1013–1024.

[27] O. Olivo, I. Dillig, and C. Lin, "Static detection of asymptotic performance bugs in collection traversals," in *Proc. 36th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: ACM, 2015, pp. 369–378.

[28] P. Zhang, S. Elbaum, and M. B. Dwyer, "Automatic generation of load tests," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*. Washington, DC, USA: IEEE Computer Society, Nov. 2011, pp. 43–52.

[29] P. Delgado-Pérez, A. B. Sánchez, S. Segura, and I. Medina-Bulo, "Performance mutation testing," *Softw. Test., Verification Rel.*, Jan. 2020, Art. no. e1728.

[30] M. M. Hassan, W. Afzal, B. Lindström, S. M. A. Shah, S. F. Andler, and M. Blom, "Testability and software performance: A systematic mapping study," in *Proc. 31st Annu. ACM Symp. Appl. Comput. (SAC)*. New York, NY, USA: ACM, 2016, pp. 1566–1569.

[31] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ. Durham Univ. Joint Report, Keele, U.K., Tech. Rep. EBSE 2007-001, 2007.

[32] H. Zhang and M. A. Babar, "On searching relevant studies in software engineering," in *Proc. 14th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*. Swindon, U.K.: GBR, BCS Learning & Development Ltd, Apr. 2010, pp. 111–120.

[33] A. Nistor, P.-C. Chang, C. Radoi, and S. Lu, "CARAMEL: Detecting and fixing performance problems that have non-intrusive fixes," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2015, pp. 902–912.

[34] X. Han and T. Yu, "An empirical study on performance bugs for highly configurable software systems," in *Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*. New York, NY, USA: ACM, 2016, pp. 23:1–23:10.

[35] Y. Yang, P. Xiang, M. Mantor, and H. Zhou, "Fixing performance bugs: An empirical study of open-source GPGPU programs," in *Proc. 41st Int. Conf. Parallel Process.*, Sep. 2012, pp. 329–339.

[36] S. H. Jensen, M. Sridharan, K. Sen, and S. Chandra, "MemInsight: Platform-independent memory debugging for JavaScript," in *Proc. 10th Joint Meeting Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 345–356.

[37] L. Song and S. Lu, "Performance diagnosis for inefficient loops," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*. Piscataway, NJ, USA: IEEE Press, May 2017, pp. 370–380.

[38] J. Li, Y. Chen, H. Liu, S. Lu, Y. Zhang, H. S. Gunawi, X. Gu, X. Lu, and D. Li, "Pcatch: Automatically detecting performance cascading bugs in cloud systems," in *Proc. 13th EuroSys Conf.* New York, NY, USA: ACM, Apr. 2018, pp. 7:1–7:14.

[39] A. Nistor, L. Song, D. Marinov, and S. Lu, "Toddler: Detecting performance problems via similar memory-access patterns," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 562–571.

[40] G. Xu and A. Rountev, "Detecting inefficiently-used containers to avoid bloat," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: Association for Computing Machinery, 2010, pp. 160–173.

[41] Y. Liu, C. Xu, S. C. Cheung, and J. Lu, "GreenDroid: Automated diagnosis of energy inefficiency for smartphone applications," *IEEE Trans. Softw. Eng.*, vol. 40, no. 9, pp. 911–940, Sep. 2014.

[42] H. Liang, Q. Zhao, Y. Wang, and H. Liu, "Understanding and detecting performance and security bugs in IOT OSes," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May 2016, pp. 413–418.

[43] J. He, T. Dai, and X. Gu, "TScope: Automatic timeout bug identification for server systems," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Sep. 2018, pp. 1–10.

[44] T. Dai, D. Dean, P. Wang, X. Gu, and S. Lu, "Hytrace: A hybrid approach to performance bug diagnosis in production cloud infrastructures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 107–118, Jan. 2019.

[45] L. Luo, A. Sriraman, B. Fugate, S. Hu, G. Pokam, C. J. Newburn, and J. Devietti, "LASER: Light, accurate sharing dEtection and repair," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 261–273.

[46] S. Tsakiltsidis, A. Miranskyy, and E. Mazzawi, "On automatic detection of performance bugs," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2016, pp. 132–139.

[47] Z. Wan, D. Lo, X. Xia, and L. Cai, "Bug characteristics in blockchain systems: A large-scale empirical study," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*. Piscataway, NJ, USA: IEEE Press, May 2017, pp. 413–424.

[48] H. Malik, H. Hemmati, and A. E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 1012–1021.

[49] H. He, "Tuning backfired? Not (always) your fault: Understanding and detecting configuration-related performance bugs," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1229–1231.

[50] S. Kothari, A. Deepak, A. Tamrawi, B. Holland, and S. Krishnan, "A 'human-in-the-loop' approach for resolving complex software anomalies," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2014, pp. 1971–1978.

[51] Z. Xu and J. Zhang, "Path and context sensitive inter-procedural memory leak detection," in *Proc. 8th Int. Conf. Qual. Softw.*, Aug. 2008, pp. 412–420.

[52] L. Della Toffola, M. Pradel, and T. R. Gross, "Performance problems you can fix: A dynamic analysis of memoization opportunities," in *Proc. ACM SIGPLAN Int. Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 607–622.

[53] R. Gu, G. Jin, L. Song, L. Zhu, and S. Lu, "What change history tells us about thread synchronization," in *Proc. 10th Joint Meeting Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 426–438.

[54] S. Lee, C. Jung, and S. Pande, "Detecting memory leaks through introspective dynamic behavior modelling using machine learning," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 814–824.

[55] D. Yan, G. Xu, S. Yang, and A. Rountev, "LeakChecker: Practical static memory leak detection for managed languages," in *Proc. Annu. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 87–97.

[56] H. Zhang, J. Rhee, N. Arora, S. Gamage, G. Jiang, K. Yoshihira, and D. Xu, "CLUE: System trace analytics for cloud service performance diagnosis," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.

[57] L. Song and S. Lu, "Statistical debugging for real-world performance problems," in *Proc. ACM Int. Conf. Object Oriented Program. Syst. Lang. Appl. (OOPSLA)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 561–578.

[58] Y. Bu, V. Borkar, G. Xu, and M. J. Carey, "A bloat-aware design for big data applications," in *Proc. Int. Symp. Int. Symp. Memory Manage. (ISMM)*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 119–130.

[59] J. Yang, C. Yan, P. Subramaniam, S. Lu, and A. Cheung, "PowerStation: Automatically detecting and fixing inefficiencies of database-backed Web applications in IDE," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 884–887.

[60] B. Welton and B. Miller, "Exposing hidden performance opportunities in high performance GPU applications," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 301–310.

[61] C. Lemieux, R. Padhye, K. Sen, and D. Song, "PerfFuzz: Automatically generating pathological inputs," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal. (ISSTA)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 254–265.

[62] T. Chen, W. Huang, M. Jiang, X. Luo, L. Xue, Y. Wang, and X. Zhang, "PERDICE: Towards discovering software inefficiencies leading to cache misses and branch mispredictions," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2018, pp. 276–285.

[63] J. Yang, P. Subramaniam, S. Lu, C. Yan, and A. Cheung, "How not to structure your database-backed Web applications: A study of performance bugs in the wild," in *Proc. 40th Int. Conf. Softw. Eng.* New York, NY, USA: Association for Computing Machinery, May 2018, pp. 800–810.

[64] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*. New York, NY, USA: ACM, 2012, pp. 267–280.

[65] G. Xu, "Finding reusable data structures," in *Proc. ACM Int. Conf. Object Oriented Program. Syst. Lang. Appl. (OOPSLA)*. New York, NY, USA: ACM, 2012, pp. 1017–1034.

[66] G. Xu, M. D. Bond, F. Qin, and A. Rountev, "LeakChaser: Helping programmers narrow down causes of memory leaks," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 270–282, Jun. 2011.

[67] A. Fehnker and R. Huuck, "Model checking driven static analysis for the real world: Designing and tuning large scale bug detection," *Innov. Syst. Softw. Eng.*, vol. 9, no. 1, pp. 45–56, Mar. 2013.

[68] Y. Jung and K. Yi, "Practical memory leak detector based on parameterized procedural summaries," in *Proc. 7th Int. Symp. Memory Manage. (ISMM)*. New York, NY, USA: ACM, 2008, pp. 131–140.

[69] Q. Li, C. Xu, Y. Liu, C. Cao, X. Ma, and J. Lü, "CyanDroid: Stable and effective energy inefficiency diagnosis for Android apps," *Sci. China Inf. Sci.*, vol. 60, no. 1, p. 12104, Jan. 2017.

[70] R. van Tonder and C. L. Goues, "Static automated program repair for heap properties," in *Proc. 40th Int. Conf. Softw. Eng.* New York, NY, USA: ACM, May 2018, pp. 151–162.
[71] M. Weninger, E. Gander, and H. Mössenböck, "Detection of suspicious time windows in memory monitoring," in *Proc. 16th ACM SIGPLAN Int. Conf. Managed Program. Lang. Runtimes (MPLR)*. New York, NY, USA: ACM, 2019, pp. 95–104.
[72] Y. Liu, J. Wang, C. Xu, and X. Ma, "NavyDroid: Detecting energy inefficiency problems for smartphone applications," in *Proc. 9th Asia–Pacific Symp. Internetware (Internetware)*. New York, NY, USA: ACM, 2017, pp. 8:1–8:10.
[73] Y. Kang, Y. Zhou, H. Xu, and M. R. Lyu, "DiagDroid: Android performance diagnosis via anatomizing asynchronous executions," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 410–421.
[74] J. Bornholt and E. Torlak, "Finding code that explodes under symbolic evaluation," *Proc. ACM Program. Lang.*, vol. 2, pp. 149:1–149:26, Oct. 2018.
[75] Y. Lin, C. Radoi, and D. Dig, "Retrofitting concurrency for Android applications through refactoring," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: ACM, 2014, pp. 341–352.
[76] X. Xiao, S. Han, D. Zhang, and T. Xie, "Context-sensitive delta inference for identifying workload-dependent performance bottlenecks," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*. New York, NY, USA: ACM, 2013, pp. 90–100.
[77] Y. Liu, C. Xu, and S.-C. Cheung, "Diagnosing energy efficiency and performance for mobile internetware applications," *IEEE Softw.*, vol. 32, no. 1, pp. 67–75, Jan. 2015.
[78] M. Dhok and M. K. Ramanathan, "Directed test generation to detect loop inefficiencies," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: ACM, 2016, pp. 895–907.
[79] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2016, pp. 101–111.
[80] J. Su and K. Yelick, *Automatic Communication Performance Debugging in PGAS Languages*. Berlin, Germany: Springer-Verlag, 2007, pp. 232–245.
[81] X. Xiao, S. Han, C. Zhang, and D. Zhang, "Uncovering JavaScript performance code smells relevant to type mutations," in *Programming Languages and Systems*, X. Feng and S. Park, Eds. Cham, Switzerland: Springer, 2015, pp. 335–355.

ANA B. SÁNCHEZ received the Ph.D. degree (Hons.) in software engineering from the University of Seville, Seville, Spain, in 2016. She was a Postdoctoral Researcher with the Department of Languages and Computer Systems, University of Seville, from 2017 to 2018. She has been a member of the Applied Software Engineering Research Group, University of Seville, since 2012. She currently works as an Assistant Lecturer with the Department of Languages and Computer Systems, University of Seville. She is the author of some articles in relevant journals, international conferences, and workshops. Her research interest includes software testing. She has also participated in the organization of international and national conferences and in the review of several journals.

PEDRO DELGADO-PÉREZ was born in Cádiz, Spain. He received the B.S., M.S., and Ph.D. degrees in computer science engineering from the University of Cádiz in 2011 and 2017, respectively.

He is currently working as an Assistant Lecturer with the Department of Computer Science and Engineering, University of Cádiz. He has been a member of the UCASE Software Engineering Research Group since 2013. He is the author of several works in journals, international conferences, and books mainly related to software engineering. He has mainly centered his research on testing techniques. His research interests include search-based software engineering, performance analysis, object-oriented programming, and the assessment of software quality factors.

INMACULADA MEDINA-BULO (Member, IEEE) received the Ph.D. degree in computer science from the University of Seville, Spain. She has been with the Department of Computer Science and Engineering, University of Cádiz, Spain, since 1995. She has published numerous peer-reviewed articles, participated in conference and workshops organization, and acted as a reviewer for several journals. She is the Main Researcher of the UCASE Software Engineering Research Group. Her current research interests include software testing, search-based software engineering, the IoT, CEP, and SOA 2.0.

SERGIO SEGURA (Member, IEEE) is an Associate Professor of software engineering with the University of Seville, Spain. He is a member of the Applied Software Engineering Research Group, where he leads the research lines on software testing and search-based software engineering. His current research interests include test automation and AI-driven software engineering.