Journal of Software: Evolution and Process

JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.1639

Challenges and opportunities for software change request repositories: a systematic mapping study

Yguaratã Cerqueira Cavalcanti[1,3,4,*,†], Paulo Anselmo da Mota Silveira Neto[1,3], Ivan do Carmo Machado[2,3], Tassio Ferreira Vale[2,3], Eduardo Santana de Almeida[2,3] and Silvio Romero de Lemos Meira[1,3]

[1]*Center for Informatics, Federal University of Pernambuco (CIn/UFPE), Recife, Pernambuco, Brazil*
[2]*Computer Science Department, Federal University of Bahia (DCC/UFBA), Salvador, Bahia, Brazil*
[3]*Reuse in Software Engineering Group (RiSE), Recife, Pernambuco, Brazil*
[4]*Brazilian Federal Organization for Data Processing (SERPRO), Florianópolis, Santa Catarina, Brazil*

ABSTRACT

Software maintenance starts as soon as the first artifacts are delivered and is essential for the success of the software. However, keeping maintenance activities and their related artifacts on track comes at a high cost. In this respect, change request (CR) repositories are fundamental in software maintenance. They facilitate the management of CRs and are also the central point to coordinate activities and communication among stakeholders. However, the benefits of CR repositories do not come without issues, and commonly occurring ones should be dealt with, such as the following: duplicate CRs, the large number of CRs to assign, or poorly described CRs. Such issues have led researchers to an increased interest in investigating CR repositories, by considering different aspects of software development and CR management. In this paper, we performed a systematic mapping study to characterize this research field. We analyzed 142 studies, which we classified in two ways. First, we classified the studies into different topics and grouped them into two dimensions: *challenges* and *opportunities*. Second, the challenge topics were classified in accordance with an existing taxonomy for information retrieval models. In addition, we investigated tools and services for CR management, to understand whether and how they addressed the topics identified. Copyright © 2013 John Wiley & Sons, Ltd.

Received 11 May 2013; Revised 27 August 2013; Accepted 2 October 2013

KEY WORDS: software maintenance; software evolution; software quality assurance; change request repository; bug report; bug tracking

1. INTRODUCTION

In software development, maintenance activities begin as soon as the first software artifacts are built and delivered [1, 2]. Such maintenance activities come at a high cost and are often time consuming [3–6]. Yet, they should be neither avoided nor postponed if the intention is for a software system to remain useful for a long time [7]. Therefore, change request (CR) repositories play a fundamental role in the software maintenance process. Although CR repositories were originally conceived to track requests for modification, they have evolved into a central hub where stakeholders such as managers, developers, and customers coordinate activities and communicate with each other [8].

The CR repositories provide developers with the management of defect fixes and enhancements in software artifacts. In addition, they also enable controlled implementation of new features, either when developing new software, or in ongoing maintenance activities. For example, in a very common

*Correspondence to: Yguaratã Cerqueira Cavalcanti, Center for Informatics, Federal University of Pernambuco (CIn/UFPE), Recife, Pernambuco, Brazil.
†E-mail: yguarata@gmail.com

workflow, when the test team or customers find any software defect or enhancement, a new CR can be reported describing the identified issue. The software team then verifies every incoming CR and a board committee establishes the implementation criteria, that is, it analyzes and decides which issues should be dealt with, the order in which this will be carried out, and it may also assign someone to undertake the task.

Another important benefit of using CR repositories is that software changes can be quickly identified and reported to developers [9]. Besides, they can also help with estimating the cost of software development, impact analysis, traceability, planning, discovery of knowledge, and understanding software, as outlined in our previous work on the CR duplication problem [10]. Furthermore, as CRs hold facts on the software maintenance and evolution of software, they can serve as the documentation of the history of the project.

However, these benefits do not come for free. A series of commonly occurring issues should be addressed, such as the following: duplicate CRs that are opened inadvertently, large numbers of CRs that must be assigned to developers, poorly described CRs, impact analysis of new CRs, and CRs that are wrongly assigned. All of these characteristics, either the benefits or issues, have led to increased interest in investigating CR repositories and to do so by considering different aspects of software development and CR management. On the other hand, there is a shortage of studies that set out to understand this increase, as discussed in this paper. These kinds of studies are essential in order to characterize what topics of investigation have been addressed in the literature and to point to possible new directions.

In this paper, we report on how we conducted a systematic mapping study to identify existing research in the field with regard to CR repositories. We considered a total of 142 studies. For the purpose of classification, we mapped the studies into different *topics* and *research areas*, which we discuss in the context of two dimensions: *challenges* and *opportunities*. Challenges refer to problems faced in CR management, whereas opportunities refer to the advantages provided by CR data for software development. For each of these, we provide a detailed analysis regarding its objectives and proposed approaches, as well as the lessons learned. The challenges dimension also comprises a classification built upon an existing taxonomy for information retrieval (IR) models [11].

In addition, we performed an investigation on the most well-known tools and online services available for CR management. Such investigation set out to understand whether the topics identified in this systematic mapping study are addressed by any of them and, if so, how.

The contributions of this study are to provide a comprehensive view on the state of the art of dealing with CR repositories issues and on pinpointing opportunities and challenges. Through this study, researchers will find lessons learned, open issues, and understand the road ahead with respect to investigating CR repositories. Software development practitioners, in turn, will also find technical references for their day-to-day work, which can be valuable for improving their own processes and tools.

The rest of this paper is organized as follows. We discuss the main concepts regarding CR repositories in Section 2. In Section 3, we outline the research questions and present the research methodology in which the matters concerning how the mapping study was conducted, and the studies classified are explained. In Sections 4, 5, and 6, we report the outcomes of the study with regard to the challenges, opportunities, and tools on CR management, respectively. Then, in Section 7, we discuss the main findings and the open issues regarding CR repositories, as well as the aspects related to the state of the art. The threats to the validity and the related work are discussed in Sections 8 and 9, respectively. Conclusions are drawn and directions for future studies are presented in Section 10.

2. BACKGROUND

The CRs are managed with the support of specialized software systems, which we simply refer to as *CR repositories*. Examples are Bugzilla [12], Mantis [13], RedMine [14], and Trac [15]. When a new CR is created, it is presumed to follow a specific workflow, which is represented as a state machine in the CR repository. It is worth mentioning that most CR repositories enable the state machine to be tailored in order to match a specific need of a software development project. Figure 1 shows a generic state machine adapted from Ihara *et al.* [16], which is representative of many software development projects. The state machine can be divided into three phases: *Untreated Phase*, *Modification Phase*, and *Verification Phase*. These are explained next.

DOI: 10.1002/smr

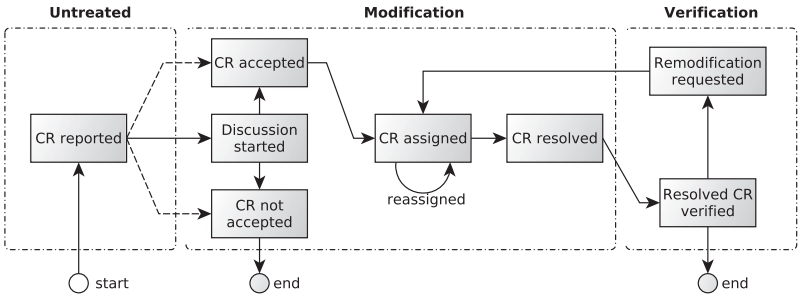

Figure 1. Generic change request (CR) workflow. We added the *CR Not Accepted* state to the original workflow.

First, in the Untreated Phase, a new CR is created in the CR repository. Each CR stores different information fields, which are essential so that the request is properly understood and implemented. For instance, it stores a short and long description of the change, the type of the change (e.g. defect or enhancement), the target component impacted by the change, and the software version. Depending on the CR repository being used, it is possible to define more informational fields when configuring the repository.

In the Modification Phase, the CR can either be accepted for implementation or not, and the discussion concerning CR aspects takes place. Such a discussion is carried out through comments that can be inserted directly in the CR. Note that it is neither necessary to have a discussion before accepting a CR nor for accepting it. There are many reasons for not accepting a CR such as the following: poor descriptions, redundancy, that is, whenever the reported issue refers to an existing CR, or it is not planned to fix the reported CR, etc. Although discussion on every CR proposed is not a requirement, it is often useful to have discussion on whether a CR should be accepted or not, and it often precedes such a decision. However, if a CR is accepted, then it must be assigned to a developer who will perform the modifications requested, as shown in Figure 1.

Assigning a CR starts with finding a suitable developer to deal with the fix or enhancement. This step is very important because finding the appropriate developer is crucial for obtaining the lowest, economically feasible fix time [17]. This developer should be one who has sufficient knowledge in the relevant aspects reported in the CR [18]. The assignment also needs to comply with the developer's workload and the priority of the CR [19]. Thus, due to these characteristics, assigning CRs is a labor intensive and time-consuming task mainly because it is performed manually [20, 21]. In fact, in many projects the number of CRs that are reported and need to be assigned can vary from dozens to hundreds per day [10]. As a result, if assignments are not properly performed, the project schedule can be affected.

After the developer assigned has implemented the CR, it goes on to the Verification Phase, which is the final phase as per Figure 1. This phase encompasses verification that the CR has been properly implemented, which it is common for the quality assurance team to do. Then, if the implementation carried out for the CR needs an additional repair, it returns to the developer who performed the implementation. It is worth mentioning that if a developer is not able to fix a CR, it can be reassigned to others until an appropriate developer is found. When the CR is given as properly fixed, this workflow reaches an end, and the change can be released for production.

3. RESEARCH METHODOLOGY

We employed a process for conducting systematic mapping studies as described in the study by Petersen *et al.* [22]. In particular, we defined a set of research questions then conducted the searches and proceeded with the selection of papers, tools, and services. Next, we built a classification scheme according to the data extracted from the studies. Finally, we analyzed and synthesized the results. By applying this process, the survey has more systematic and formal characteristics, which can make further replications or extensions easy. This section is structured in accordance with the process so that each subsection represents one of its steps.

DOI: 10.1002/smr

3.1. Research questions

The objective of this study is to understand the state of the art on CR repositories research. The focus is to identify the challenges and opportunities in this field. By challenges, we mean the issues faced during CR management, which we wish to characterize according to their impact on software development and techniques used to address them. On the other hand, opportunities mean the advantages that can be taken from the data present in CR repositories to aid software development. In addition, we attempt to understand if existing tools and online services for CR management have addressed any of the challenges that we have identified. Thus, in order to achieve and guide all of these goals, we defined the following research questions:

- *Question 1 – What are the current challenges and opportunities regarding CR repositories and how do they impact software development?* This is the leading research question in this investigation. The aims are the following: first, to collect the types of challenges and opportunities that have emerged by using CR repositories in software projects; secondly, to gain an understanding on the main impact of these problems on the software development life cycle; and thirdly, to identify the approaches that have been proposed to address these issues.
- *Question 2 – Do the tools and online services for CR management address any of the challenges pointed out as a result of the answers to Question 1? If so, how do they address such challenges?* There is a variety of tools and online services that supports CR handling such as Mantis, Bugzilla, SourceForge, and GitHub, which are widely used in and by software companies. Thus, it is important to understand if they implement any solutions to the problems identified in answers to the previous question. Therefore, we want to understand if there has been any technology transfer in this field. By technology transfer, we mean the transfer of techniques and methods from the state of the art to the state of the practice.

3.2. Search in the literature

In order to select the most relevant set of studies and eliminate studies which do not address the research questions, the following criteria were used to form the final set of *studies included*:

- The study must report either theory, or practice, or approaches to CR repository issues;
- The CR artifacts themselves, which were the objects of analysis in the study, must be described in natural language;
- The study must be unique, that is, when a study was published in more than one venue, we only considered the most complete version of the investigation. The remaining studies were only used as an occasional support to complement the findings;
- Any study that did not address any of the topics of the research questions topics must be excluded; and
- Studies concerning tutorial or workshop summaries, poster summaries, keynotes, or studies with no scientific analysis or published in unknown sources must be excluded as well.

We collected the studies in two steps. In the first, we carried out an automated search, which consisted of applying a combination of search terms in the following search engines: *Google Scholar*, *ACM Portal*, *IEEExplorer*, *Citeseer*, *Elsevier*, *Scirus*, *ScienceDirect*, *Scopus*, *ISI Web of Knowledge*, *SpringerLink*, and *Wiley*. The keywords were as follows: *bug report*, *change request*, *modification request*, *defect track*, *software issue*, and *bug tracking*. Therefore, we formulated a search query using these keywords for each of the search engines as shown in Table A.1 in Appendix A. As a second step, we manually trawled a set of representative conferences and journals as listed in Tables B.1 and B.2 in Appendix B.

After conducting the automated and manual searches, we had a set of 1550 studies. Then, this set was filtered according to the aforementioned criteria for including studies, which reduced the total to 321. This difference from the first to the second filter was in part due to the number of duplications that arise when looking at different sources of information such as search engines, conferences, and journals. Finally, the remaining studies were analyzed on the basis of the abstract and reading the full text. This resulted in 142 studies being included in this study.

DOI: 10.1002/smr

Table I. List of the tools and online services selected.

Tools		Online Services	
Bugzilla	http://www.bugzilla.org	SourceForge	http://www.sourceforge.net
MantisBT	http://www.mantisbt.org	Launchpad	http://www.launchpad.net
Trac	http://trac.edgewall.org	Code Plex	http://www.codeplex.com
Redmine	http://www.redmine.org	Google Code	http://code.google.com
Jira	http://www.atlassian.com	GitHub	http://www.github.com

3.3. Tools selection and analysis

There is a wide range* of CR repository systems; however, they have almost the same concept for handling CRs. Thus, as it would be hard to analyze each of them to answer *Question 2*, we selected a subset considering the best known tools disregarding whether these were open source or commercial software. Additionally, we also considered CR repository tools available online, that is, provided as a Software as a Service, such as SourceForge. Table I lists all the tools we considered in the study.

The analysis of each tool was performed as per the challenges and opportunities identified in *Question 1*. Thus, for each tool, we analyzed whether there was any method or technique that addressed it. We analyzed the tools by looking at their documentation such as development documents and user manuals, and available extensions (such as plug-ins) in those tools, which have an extensible architecture. If none of the previous approaches was enough, we conducted the analysis by using the tools.

3.4. Classification scheme

We provide two classification schemes for the studies considered. The first scheme organizes the studies in accordance with the challenges and opportunities that they address. The second one is the classification based on the taxonomy for IR models [11] and is used to classify the challenges identified. In particular, we used this scheme, because all the challenges are addressed with the support of some IR model. As such, the taxonomy provides an easy way to understand, which kind of methods and techniques for IR have been used. Next, we discuss both schemes in detail.

3.4.1. Classification as per challenges and opportunities. First, we categorized the studies using the process defined by Petersen *et al*. [22]. This process is performed in two steps: (1) looking for keywords and concepts that identify the contributions of the study by analyzing the title, keywords, and abstracts; and (2) thereafter, all the keywords are combined to build a high level understanding set of categories with which the papers should be classified. Petersen *et al*. [22] also state that should the abstracts be poor, the introduction and conclusion sections must be analyzed. By applying the process, we created the classification scheme presented in Figure 2, which we will discuss next.

As can be seen, the classification scheme is divided into two dimensions: (i) challenges, which are those related to research with the objective to improve CR management (left side of Figure 2) and (ii) opportunities, which are those related to research that use data from CR repositories to understand or improve other aspects of software development (right side of Figure 2). Furthermore, we divided both categories into *research areas* and *topics*, as follows:

- *CR classification*. The research in this area investigates the core activities of CR management, i.e., assigning CRs to appropriate developers, identifying duplicate CRs and CR types, prioritizing CRs, and clustering similar CRs.
- *CR effort estimation*. This research area concerns investigations for the purpose of estimating the effort necessary for fixing a CR, measuring the impact for a given CR, or even predicting the number of CRs that are expected for a software project.
- *CR data*. In this research area, we grouped work that analyzed data from CR repositories, automatically or manually, for quality assessment of CR attributes, or for information extraction, to improve the quality of CRs.

*In http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems, it is possible to see more than 50 tools and some comparison among them.

DOI: 10.1002/smr

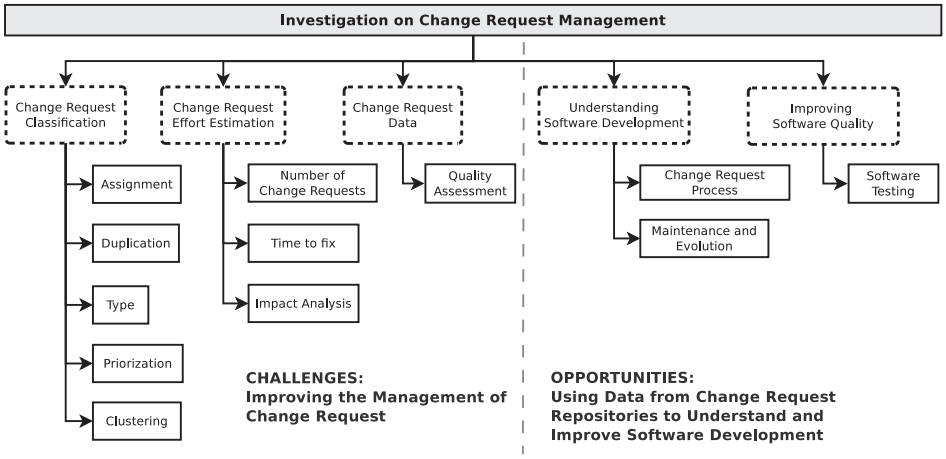

Figure 2. Classification scheme as per challenges and opportunities. The dashed boxes on the top are the research areas, and the solid boxes below them are the topics. The scheme is further divided by a dashed vertical line, which separates the challenges from the opportunities.

- *Understanding software development*. Research concerned with using data from CRs to understand aspects related to software development, such as the process involved in managing CRs and characteristics of software maintenance and evolution.
- *Improving software quality*. This research area conducts investigations that use data from CRs to improve software quality, such as aiding testing and verification activities.

Additionally, Table II shows the distribution of the studies per topic. To a certain extent, such a distribution indicates the maturity of the research in each of these topics. Nevertheless, it does not necessarily indicate the importance that each topic has in CR management. Additionally, the studies

Table II. List of papers according to the classification scheme.

	Research Area	**Topic**	**Studies**	*Total*
Challenges	Change request classification	*Assignment*	[17, 18, 20, 21, 23–46]	28
		Duplication	[9, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]	20
		Type	[66, 67, 68]	3
		Prioritization	[69, 70, 71, 72, 73, 74, 75, 76]	8
		Clustering	[77, 78, 79]	3
	Change request effort estimation	*Number of change requests*	[63, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]	13
		Time to fix	[19, 89, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107]	18
		Impact analysis	[55, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120]	14
	Change request data	*Quality assessment*	[12, 18, 22, 24, 56, 81, 83, 88, 91, 117, 121, 122, 123, 124]	15
Opportunities	Understanding software development	*Change request process*	[8, 96, 125, 126, 127, 128, 129, 130, 131, 132]	10
		Maintenance and evolution	[123, 133, 134]	3
	Improving software quality	*Software testing*	[135, 136, 137, 138]	4

DOI: 10.1002/smr

that addressed more than one topic were classified repeatedly in each topic, that is, since Hosseini *et al*. [19] investigated the topics of *CR assignment* and *time to fix*, their study is counted under both two topics.

3.4.2. Classification as per information retrieval techniques. Most of the papers we found provide approaches for the topics presented in Figure 2 by using IR models and techniques. The IR models solve a user's problem as regards to an information need, represented by a set of terms (query), where a list of the most relevant documents must be retrieved from the whole collection of documents [139]. In order to facilitate the understanding of the existing research on the use of IR models and techniques, we classified the studies using an adapted version of the taxonomy proposed by Canfora and Cerulo [11]. The adaptation was necessary because the original taxonomy did not consider all the techniques that we found nor the different sources of information (repositories) considered in the approaches. Table III presents the taxonomy applied. It comprises three dimensions:

- *Representation* – the techniques used to represent queries and documents in the IR model. As we will explain in the following sections, both queries and documents are representations of the contents of a CR;
- *Reasoning* – the classes of algorithms used to rank the retrieved documents. Although Table III only shows the classes, in the following sections we cite each algorithm used in the approaches; and
- *Repository* – the type of repository in which the documents are queried and retrieved. Therefore, we can see that CR repositories can be queried using other kinds of information, present in other repositories, for improving the results.

Specifically, the extension we made consisted of adding the new dimension of repository and new types of techniques for *reasoning with learning*. Regarding the repository dimension, all studies used data from some of these information sources: CR repositories, such as Bugzilla and Mantis; Commit logs, which are items of information stored in version control systems (such as Subversion and Concurrent Version System); and source code. Additionally, each item of the dimensions of representation and reasoning represents a class of different algorithms that can be implemented. However, although we list the algorithms used in each approach found in the literature, it is beyond the scope of this paper to provide a description of their operation.

It is important to mention that when some topic does not have enough studies that would justify the use of the taxonomy table, we only briefly describe them in the text; this is the case for the topics of *type*, *clustering* and *number of CRs*.

4. CHALLENGES: IMPROVING THE MANAGEMENT OF CHANGE REQUEST

In this section, we present the analysis of the studies with respect to the challenges regarding CR repositories. As we previously introduced, these challenges are the problems faced in CR management. We included all studies that present approaches or provide any kind of analysis on CR repository data to ease the management of CRs. Next, we discuss the research areas and their topics.

Table III. Taxonomy for the classification of the information retrieval models and techniques used in each approach. This is an extension of the taxonomy created by Canfora and Cerulo [11].

Representation						**Reasoning**													**Repository**		
Query			Document			With logic			With uncer-tainty		With learning										
Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Genetic Algorithms	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code

CVS, Concurrent Version System; SVN, Subversion.

DOI: 10.1002/smr

4.1. Change request classification

This research area was created to accommodate studies, which proposed approaches or analysis regarding the classification of CRs. As we describe later, this accommodates many topics, such as CR assignment, duplicate CRs analysis, classification of CR types, prioritization of CRs, and clustering.

4.1.1. Change request assignment. The assignment of CRs, also called *triage*, concerns the objective of finding the most suitable developer to handle a given CR. Assigning a CR to the appropriate developer is crucial for obtaining the lowest, economically feasible time to fix [17]. Such a developer is the one who has enough knowledge in the relevant aspects reported in the CR [18]. In addition, the assignment also needs to be performed in accordance with the developer's workload and the CR priority [19]. Thus, considerable knowledge of the project is needed as is the communication ability to negotiate with developers and stakeholders. Based on these characteristics, the activity of assigning CRs to developers is labor intensive and consumes a considerable amount of, especially because it is performed manually [20, 21]. Depending on the project being developed, the number of new CRs can vary from dozens to hundreds in a single day [10]. Thus, the greater the number of CRs that are opened, the more complex the problem becomes.

4.1.1.1. The change request assignment as an information retrieval problem. Most research we found addressed the problem of CR assignment by using IR models and techniques with the objective of providing automated solutions. When applying IR models to the CR assignment problem, the content of an incoming CR is used to query the database of CRs already fixed. Then, a list of potential developers to be assigned is retrieved from the CRs in the query results and suggested to the analyst. The analyst, in turn, selects the desired developer from the suggested list.

Some approaches are proposed on the basis of the hypothesis that the most suitable developer for a new CR is the one who has already solved similar CRs in the past [17, 20, 21, 24, 30, 34, 40]. While other approaches consider that the suitable developer can be found by looking into past CRs and information from version control systems [23, 28, 33, 36] or source code [35]. In general, all these approaches use IR models to automatically suggest a list of potential appropriate developers for a new incoming CR, thus leveraging the assignments of CRs.

We used the taxonomy for IR models to map out the papers we found, as shown in Table IV. New methods for reasoning with learning found in the papers had to be added to the taxonomy, so the papers could be fully mapped. As to the representation of text in the IR models, Table IV shows that all approaches represent the query (incoming CR to be assigned) using a keyword-based approach, while the representation for the documents (information about developers) is given by using vector space techniques. Specifically for the vector space techniques, all studies implemented the Term Frequency–Inverse Document Frequency (TF-IDF), and some of them also tested the Latent Semantic Indexing (LSI) [23, 24, 33, 35, 72], N-Gram [37], and the Alphabet Frequency Matrix [39]. Finally, Table V shows the reasoning algorithms implemented in the papers according to their category (with and without learning).

4.1.1.2. What kind of repositories should I use?. In Table IV, most papers only used CR repositories for their approaches to CR assignment, while just a few used CR repositories in conjunction with information from version control systems. In the former, the research based its approaches on the assumption that the list of appropriate developers could be obtained from similar, already solved CRs, which are retrieved by comparing the similarity among CR text descriptions. For the latter case, the assumption relies on the belief that CRs and commit descriptions (on version control systems) are linked, formally or according to project conventions. The formal linkages are provided by specific tools, such as Rational ClearCase or Jazz. The linkage by project convention can be achieved, that is, when developers are used to inserting the CR identification in the commit description when they submit source code changes to the repository.

In this context, Anvik and Murphy [25] investigated which strategy was most appropriate for determining the implementation expertise needed by a project in order to solve the problem of CR assignments. They concluded that the approach is one that combines CR and source code repositories is the best one for recommending experts. According to the authors, such an approach does not retrieve all the relevant developers; however, it retrieves a subset of relevant developers with fewer false positives. On the other hand, a similar study [140] found that the linkage, by project convention,

DOI: 10.1002/smr

Table IV. Research on change request (CR) assignment mapped to an adaptation of the taxonomy created by Canfora and Cerulo [11].

	Representation						Reasoning													Repository		
	Query (new CR)			Document (fixed CRs)			With logic			With uncertainty		With learning										
Approach	Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space*	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Genetic Algorithms	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code
Lucca *et al.* [17]	✓				2			✓		✓				✓	✓	✓					✓		
Cubranic and Murphy [30]	✓				2												✓				✓		
Anvik *et al.* [20]	✓				2									✓							✓		
Canfora and Cerulo [28]	✓				2					✓											✓	✓	
Bettenburg *et al.* [26]	✓				2									✓			✓				✓		
Ahsan *et al.* [24]	✓				1,2							✓		✓	✓		✓				✓		
Ahsan *et al.* [23]	✓				1,2									✓	✓	✓	✓				✓	✓	
Jeong *et al.* [21]	✓				2				✓								✓				✓		
Lin *et al.* [34]	✓				2									✓							✓		
Matter *et al.* [36]	✓				2			✓													✓	✓	
Rahman *et al.* [40]	✓				2										✓						✓		
Tan *et al.* [42]	✓				2			✓													✓		
Xuan *et al.* [43]	✓				2												✓				✓		
Aljarah *et al.* [18]	✓				2												✓				✓		
Chen *et al.* [29]	✓				2			✓													✓		
Nasim *et al.* [39]	✓				4							✓		✓	✓		✓		✓		✓		
Tamrawi *et al.* [41]	✓				2						✓										✓		
Zou *et al.* [46]	✓				2												✓				✓		
Bhattacharya *et al.* [27]	✓				2				✓					✓	✓		✓				✓		
Kagdi *et al.* [33]	✓				1			✓													✓	✓	
Linares-Vásquez *et al.* [35]	✓				1			✓													✓		✓
Nagwani and Verma [38]	✓				2			✓													✓		
Moin and Neumann [37]	✓				3			✓													✓		
Xuan *et al.* [44]	✓				2				✓					✓			✓				✓		
Zhang and Lee [45]	✓				2				✓					✓							✓		

*** Types of Vector Space representation for documents:**
(1) Latent Semantic Indexing (2) TF-IDF (3) N-Gram (4) Alphabet Frequency Matrix

between CRs and version control systems is very prone to errors, because it needs to be carried out manually. If errors occur when creating these linkages, the repositories would become biased. Such a bias could potentially have an impact on the performance of IR models [20, 140].

4.1.1.3. Do change request reassignments matter?. The other concern regarding CR assignment is about the reassignments. These may be viewed as a symptom, which indicates that the first assignments are not fully effective, that is, the CRs were wrongly assigned at first and thus must be reassigned until they reach the appropriate developers. Jeong *et al*. [21] argued that such reassignments are time consuming, thus affecting the project schedule, and proposed a method based on Tossing Graphs to minimize the number of reassignments.

However, in a more recent study [141], we identified that CR reassignments are useful when the CRs need to be addressed by different teams, such as those CRs that need to be fixed either by the test team or the requirements team. In this case, the reassignments are necessary to ensure that a CR will reach all those responsible. Due to these different characteristics, the reassignment of CRs can either be useful or harmful, which is also supported by Guo *et al*. [31], and each case must be addressed accordingly.

4.1.1.4. Modeling change request assignment activity with Lotka's law. For the scientific productivity context, Lotka's law says that as the frequency of publications increases, the number of authors who publish articles with that frequency is supposed to be predictable. Similarly, Canfora and Cerulo [28] identified that the distribution of the CR assignments follows actually the Lotka's law. In others to investigate this, they used Zpif's equation, which produces a curve similar to the

DOI: 10.1002/smr

Table V. Algorithms used in different proposals for automatic change request assignment.

Class	Algorithms	Papers
With learning	Bayesian Network	[18, 21, 27, 39, 41]
	Bayesian Networks (Updateable)	[41]
	Naive Bayes	[21, 23, 24, 26, 27, 30, 39, 41, 43, 44, 46]
	Naive Bayes (Discriminative Multinominal, Multinomial Updateable, Multinomial, Updateable, Complement)	[39]
	Naive Bayes (with Training Set Reduction)	[46]
	Semi-supervised Naive Bayes with expectation-maximization	[43]
	C4.5	[27, 41]
	Classification And Regression Trees	[17]
	Decision Table	[24]
	Random Forest	[24]
	REPTree	[24]
	Tree J48	[24, 39]
	k-Nearest Neighbor	[17, 23, 45]
	Label Power Set	[23]
	Radial Basis Function Network	[24]
	Multi Layer Perceptron	[39]
	Simple Logistic Regression	[39]
	Support Vector Machine	[17, 20, 23, 24, 26, 27, 39, 41, 44, 45]
	Support Vector Machine (Polynomial kernel function, degree=2) (RBF kernel function)	[27]
	Fuzzy set	[41]
Without learning	Probabilistic IR model	[17, 28]
	Vector Space IR model	[17, 29, 33, 35–38, 42]
	Greedy search	[40]
	Auction algorithm	[19]
	Tossing Graphs	[21, 27, 29]

RBF, radial basis function; IR, information retrieval.

one shown in Figure 3. There are two cut levels in this figure, which divide the frequency of assignments into three groups: the first group is supposed to hold a small number of developers to whom the most CRs are assigned; the second group, which holds more developers than the first

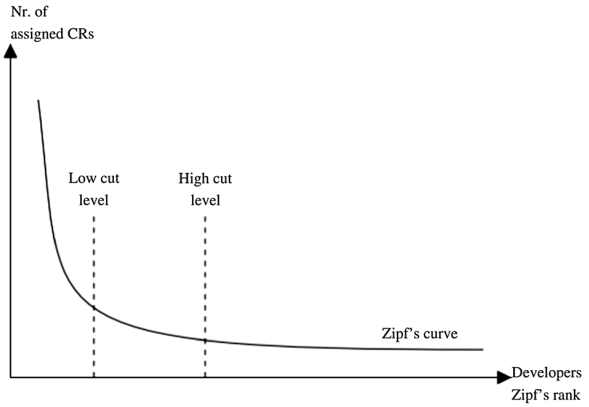

Figure 3. The curve produced by Zpif's equation [28].

DOI: 10.1002/smr

group, has a reasonable number of CRs assigned; and the third group has the highest number of developers to whom few CRs have been assigned.

In summary, the application of Lotka's law reveals that a small set with the most active developers is generally responsible for most of the tasks in a software project, that is, they fix a greater number of CRs. Similar results in this respect are also presented in [142] for both open source and proprietary software.

Although only Canfora and Cerulo [28] used such a law to model the distribution of assignments, other studies can be used to confirm the adequacy of Lotka's law for this matter. For instance, Crowston and Scozzi [125] had already identified this behavior in some open source projects in which most of the CRs are resolved by a small group of developers. Additionally, Hosseini *et al.* [19] plotted a real example of the CR assignment distribution for the Eclipse project as shown in Figure 4. As expected from Lotka's law, we can see a high concentration of CRs for a small number of developers on the left side of the figure forming a curve, which is indeed very similar to Zipf's equation.

Finally, considering the automated methods for CR assignments, Tamrawi *et al.* [41] observed that this behavior has also implications in the calibration of these methods. For example, a recommendation tool could rely only on a small set with the most active developers to perform the recommendations, which in turn, would require less computational resources to run the algorithms.

4.1.2. Analysis of duplicate change requests. Duplicate CRs are those that request a change, which has already been reported in existing CRs. Duplicate CR analysis, in turn, is the activity where someone has to decide whether a CR is a duplicate or not. When a duplicate is identified, it must be linked to the master CR, which was the one to request the change first. Many studies have shown that between 10% and 30% of the requests in a CR repository comprise duplicate CRs and that the presence of duplicates can impact software maintenance activities [9, 10, 57]. Due to the large number of duplicates, it is often necessary to designate people to perform an analysis of incoming CRs in order to avoid duplicates ending up going to developers [9, 52, 56, 59, 62]. If these duplicate CRs are not identified, then this will consume time and effort unnecessarily [51], and the costs for the whole life cycle of software maintenance may be adversely affected [10, 131].

The identification of duplicate CRs is not an easy task. Actually, in order to avoid (before submitting) or identify a duplicate, it is necessary to have a good knowledge of past CRs or to manually perform searches in the whole CR repository [9, 59, 61, 131]. Both strategies can be time consuming and do not avoid the misidentification of duplicates [52]. Additionally, if CRs are misidentified (false-positives), important software issues could be ignored [131]. Needless to say, it is clear that such tasks become more challenging with the increasing number of CRs that are constantly submitted to the repository [62]. For instance, in a previous study [10], we identified that more than 10 min is spent on average to take such a decision on open source projects, and, in

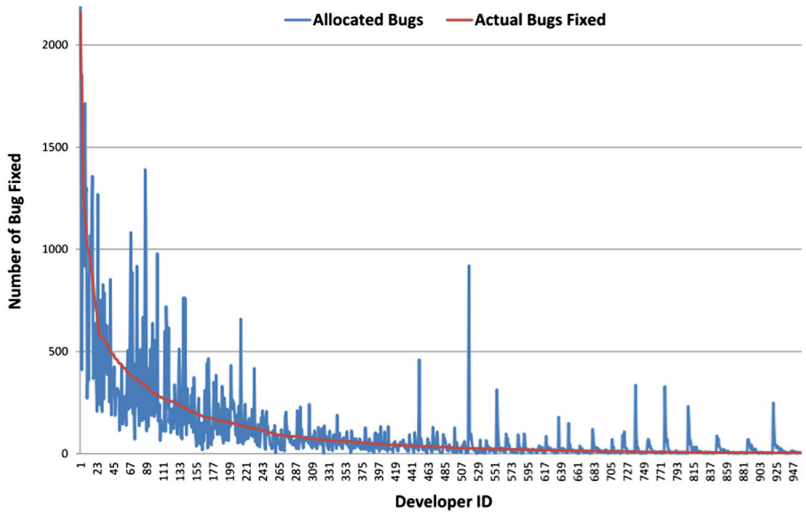

Figure 4. Statistics on change request assignments producing a curve similar to Zipf's equation [19].

DOI: 10.1002/smr

private projects, this amount of time can exceed 20 min. Runeson *et al*. [57] noted that about 30 min are spent on average to analyze a single CR in Sony Ericsson.

4.1.2.1. Duplicate change request analysis as an information retrieval problem. Due to the time and effort necessary to identify duplicates, IR techniques could aid in this challenge. Indeed, IR techniques could improve the productivity of the team that makes the analysis of incoming CRs [58], or even avoid the submission of duplicates [51]. In this context, most studies found in the literature applied IR techniques that aim to identify or avoid duplicates. Table VI summarizes the studies found according to the IR taxonomy adapted from [11]. In these IR models, new CRs are regarded as the queries for the model, whereas the existing CRs in the repository are the documents, and only the most relevant of which must be retrieved. Thus, the main objective of the proposed approaches is to compute similarity among CRs.

All studies presented in Table VI applied keyword-based techniques to represent the queries while the documents are represented with the vector space model (VSM). For the VSM techniques, Sureka and Jalote [60] tested the N-Gram model, and Kaushik and Tahvildari [52] compared the models TF-IDF, log IDF, log entropy, TF entropy, LSI, random projections, and Latent Dirichlet Allocation (LDA), whereas all the others only used the TF-IDF model. The LDA approach is also used in [55]. As to the reasoning of the IR models, most studies implemented algebra models, whereas very few investigated reasoning techniques with uncertainty or learning. The approaches which use probabilistic theories (uncertainty) implemented the similarity functions Kullback–Leibler [63] and BM25F [55, 59, 61], whereas support vector machine (SVM) [58, 61, 64] and learn to rank [53, 65] were used for reasoning with learning. For the approaches concerning algebra models, all studies tested similarity functions such as cosine, whereas Runeson *et al*. [57] also tested the Jaccard and Dice functions.

Table VI. Research on duplicate change requests (CRs) analysis mapped to an adaptation of the taxonomy created by Canfora and Cerulo [11].

	Representation						**Reasoning**													**Repository**		
	Query (input CR)			Document (CR)			With logic			With uncertainty		With learning											
Approach	Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space*	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Genetic Algorithms	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code
Anvik *et al.* [9]	✓				4			✓													✓		
Hiew [50]	✓				4			✓													✓		
Runeson *et al.* [57]	✓				4			✓													✓		
Jalbert and Weimer [51]	✓				4			✓	✓												✓		
Wang *et al.* [62]	✓				4			✓													✓		
Cavalcanti [47]	✓				4			✓													✓		
Nagwani and Singh [54]	✓				4			✓													✓		
Sun *et al.* [58]	✓				4			✓						✓							✓		
Sureka and Jalote [60]	✓				7			✓													✓		
da Cunha *et al.* [48]	✓				4			✓													✓		
Prifti *et al.* [56]	✓				4			✓													✓		
Sun *et al.* [59]	✓				4					✓											✓		
Wu *et al.* [63]	✓				4			✓		✓											✓		
Zhang and Lee [64]	✓				4			✓						✓							✓		
Nguyen *et al.* [55]	✓				5					✓											✓		
Kaushik and Tahvildari [52]	✓				1-6			✓													✓		
Tian *et al.* [61]	✓				4					✓				✓							✓		
Liu *et al.* [53]	✓				4															✓	✓		
Zhou and Zhang [65]	✓				4															✓	✓		

*** Types of Vector Space representation for documents:**
(1) Log-IDF (2) Log-Entropy (3) Latent Semantic Indexing (4) TF-IDF (5) Latent Dirichlet Allocation (6) TF-Entropy (7) N-Gram

CVS, Concurrent Version System; SVN, Subversion; TF-IDF, Term Frequency–Inverse Document Frequency.
[a]Types of vector space representation for documents: (1) Log IDF, (2) log entropy, (3) Latent Semantic Indexing, (4) TF-IDF, (5) Latent Dirichlet Allocation, (6) TF entropy, and (7) N-gram.

DOI: 10.1002/smr

4.1.2.2. Two information retrieval strategies for the duplication problem. The IR approaches identified in the literature can be divided into two main types [52, 61]: (1) removal of duplicates, as in [51]; and (2) identification of duplicates, which holds most research. In the first type, the objective is to prevent duplicate CRs from entering the repository and avoiding the extra time and effort required to identify them later.

On the other hand, the second type does not care if duplicates enter the repository. The objective is to suggest a list of possible duplicates for a new CR, and maybe to group them. Research on the second type bases its approaches on the premise that duplicate CRs are not always bad and can actually provide additional useful information [26]. However, such a premise cannot always be true because the additional information present in duplicate CRs are inserted by accident and, additionally, the costs of identifying and grouping duplicates can outweigh their benefit [49]. Therefore, it is ideal that new approaches try to balance these two types. It might avoid extra time on CR analysis and also support developers with additional information [52, 56].

4.1.2.3. Being careful with machine learning approaches. In most CR repositories, there are groups of duplicate CRs. Each group consists of a master CR, which generally is the first CR opened and its children, which are the duplicate CRs. In order to apply machine learning algorithms, it is desirable to have large groups of duplicates. Thus, it is possible to train the algorithms in each group.

However, as we observed in our previous paper [10], in all CR repositories that we analyzed most of the groups comprise only a master CR and one child, and thus are groups, which do not have enough data to train algorithms. As a consequence, before applying learning approaches, it is necessary to analyze the CR repository to understand the size of the groups of duplicates.

4.1.2.4. Reasons for the duplication problem. Some studies point out reasons for the duplication problem or factors that can leverage it. Bettenburg *et al*. [26] for instance, highlighted the following reasons: (a) *lazy or inexperienced users* (*reporters*); (b) *poor search features* provided by the CR repository tool; (c) *multiple failures, one defect*, which means that different failures are reported in different CRs but the failures were generated by the same defect; and (d) *intentional or accidentally resubmission* of the CRs. In our previous paper [10], we confirmed the findings of Bettenburg *et al*. [26] in which they say that inexperienced reporters are supposed to submit more duplicate CRs. Specifically, we divided the reporters into three profile groups: frequent reporter, average reporter, and sporadic reporter. Our findings indicate that sporadic reporters have an 80% chance of submitting a duplicate CR. However, the findings of Davidson *et al*. [49] point that the profile is not so determinant, but agree that skilled reporters are less likely to submit duplicates.

4.1.3. Classifying the change request type. When a CR is submitted to the repository it must be classified according to its type, which in turn will determine the kind of maintenance that should be performed, that is, corrective, perfective, preventive, or adaptive [1]. Most CR repositories enable a CR to be classified as a request for fixing a defect or implementing an enhancement. However, CRs are not always concerned with software maintenance activities, and some of them are often about requests for help, architectural discussions, and legal and licensing issues (a common case for open source projects) [66, 143]. Such a diversity in the CR types end up leading to misclassifications [66, 143]. For instance, a CR that is actually about a defect could be misclassified as a new feature by an inexperienced developer. Such misclassification can produce project delays, because misclassified CRs could be assigned to wrong developers or even receive less attention than they deserve.

Herzig *et al*. [143] also pointed out that misclassifications have a severe impact on IR models for defect prediction (further discussed in Section 4.2.3). This is, if the prediction models rely on data from the CR repository, the predictions can be biased if all the CRs are not correctly classified. In a study that analyzed 7000 CRs, Herzig *et al*. [143] found that CR classifications are unreliable: 33.8% of the CRs classified as defects were actually not about corrective maintenance. They also analyzed some defect prediction IR models existing in the literature, and found that 39% of source code files were wrongly marked as defective and that 16–40% of the top 10% of source code files predicted as defect-prone was wrong.

4.1.3.1. The classification of change request type as an information retrieval problem. In this context, IR models can be applied to such a problem if we think that new incoming CRs could be classified based on existing similar CRs, previously classified. Two studies investigated this

DOI: 10.1002/smr

approach. Antoniol *et al*. [66] implemented an IR model using VSM with TF-IDF to represent the CRs (queries and documents) and compared three machine learning algorithms to provide the classification mechanism: decision tree, Naive Bayes, and logistic regression. Kadar *et al*. [68] provided an IR model in which VSM was also used. However, they implemented VSM using TF-IDF and the cosine similarity function in conjunction with LDA and a machine learning algorithm for multinomial logistic regression. However, neither study considered the problem of misclassifications of CRs.

4.1.4. Prioritization of change requests. Each CR reported in the repository needs to be prioritized so it can be correctly scheduled in the maintenance plan. This prioritization is also important because developers can allocate their time and effort accordingly [71, 75]. Usually, the prioritization of CRs will determine how urgent they should be addressed [73]. Therefore, most CR management systems provide specific fields in the CR form to be selected in accordance with the priority. In Bugzilla, for instance, the prioritization of a CR is determined by two fields: the *priority* field itself, which specifies how urgent it is (e.g., for business goals) and the *severity* field, which specifies the degree of impact of the CR on the system (e.g., if it crashes the application) [71]. However, the CR priority can also be determined according to its subject. For example, Gegick *et al*. [70] state that if a CR has to do with some security concern in the software, so it has a high priority. Indeed, if security problems are not fixed as soon as possible, they can cause serious damage to the project [70].

4.1.4.1. Prioritization of change requests as an information retrieval problem. Very similar to the challenge of classifying a CR's type as earlier presented, IR models are also likely to foresee the CR priority. For example, when a new CR is opened, existing CRs, already prioritized, can be queried by using textual information from the new CR, which will retrieve a set of similar CRs. Then, the priority of the new CR is calculated on the basis of the priority of the retrieved CRs [70, 72–76]. Table VII shows the approaches classified according to their IR model characteristics. For the vector space representation, binary and term count was used in [74] and TF-IDF in all other approaches, except by Gegick *et al*. [70], which used log entropy and TF entropy. Additionally, all reasoning algorithms implement learning techniques, as follows: SVM [69, 72, 74, 76], decision tree [69, 76], k-nearest neighbor [69, 74], Naive Bayes [69, 72–74], and Naive Bayes Multinomial [69].

4.1.5. Automatic change requests clustering. In the previous categories, all the approaches had distinct objectives regarding the CR management process. Some of them were interested in classifying CRs according to the appropriate developer, identifying duplicates, prioritizing CRs, or

Table VII. Research on prioritization of change requests (CRs) mapped to an adaptation of the taxonomy created by Canfora and Cerulo [11].

	Representation						**Reasoning**														**Repository**		
	Query (input CR)			Document (CR)			With logic			With uncertainty		With learning												
Approach	Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space*	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Genetic Algorithms	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code
Menzies and Marcus [75]	✓				2,4								✓								✓		
Gegick *et al.* [70]	✓				2																✓		
Lamkanfi *et al.* [73]	✓				2												✓				✓		
Sureka and Indukuri [76]	✓				2									✓	✓						✓		
Lamkanfi *et al.* [74]	✓				2,3									✓		✓	✓				✓		
Kanwal and Maqbool [72]	✓				2									✓			✓				✓		
Chaturvedi and Singh [69]	✓				2								✓	✓	✓	✓	✓				✓		
*** Types of Vector Space representation for documents:** (1) Latent Semantic Indexing (2) TF-IDF (3) Binary (4) InfoGain																							

CVS, Concurrent Version System; SVN, Subversion; TF-IDF, Term Frequency–Inverse Document Frequency.
[a]Types of vector space representation for documents: (1) Latent Semantic Indexing, (2) TF-IDF, (3) binary, and (4) InfoGain,

DOI: 10.1002/smr

detecting the correct maintenance type of a CR. However, we identified some work that also aimed to classify CRs but without a specific defined objective.

Kumar Nagwani [77] proposed a means of classifying CRs according to clusters of labels that are automatically created (from CRs textual descriptions) by a clustering algorithm. Then, such labels are mapped to predefined CR categories, thus classifying each CR cluster. The CR categories are generated as per the software domain and characteristics, such as logical, GUI, data type, security, memory, and analysis. [77]. Similarly, Rus *et al*. [78] also proposed an approach to clustering CRs; however, they did not categorize the CR clusters, and Santana *et al*. [79] analyzed the performance of different algorithms for CR clustering.

4.1.5.1. The automatic clustering of change requests as an information retrieval problem. Unlike previous classification types, the classification of clustering does not have a new incoming CR as a query. Thus the task of retrieving relevant documents is not necessary. The main objective is only to group the similar CRs together through clustering algorithms. As to the IR techniques, both studies applied machine learning approaches and VSM for document representation. Kumar Nagwani [77] tested his algorithm against SVM, naive Bayes and Weka's Classification using clustering, while Rus *et al*. [78] tested the k-means, normalized cut, and size regularized cut algorithms.

4.2. Effort estimation for change requests

In this research area, we included studies that aim to estimate the effort required to fix a CR. We identified three ways to measure this effort: determining the time to fix new CRs, determining the initial artifacts impacted by new CRs, and forecasting the number of incoming CRs.

4.2.1. Determining the time to fix of change requests. This topic concerns estimating the time necessary to address the change required in the request. It is particularly difficult to make an accurate estimate due to the kinds of tasks involved, such as debugging, concept location in source code and other related software artifacts [105, 144]. Furthermore, the degree of complexity involved in these tasks often varies among different CRs and the fixing of CRs can also induce new CRs, thus hampering even more the accuracy of such estimates [96, 144]. Indeed, the developers' estimation of the time to fix a CR is very biased and is underestimated or overestimated most of the time [145].

However, accurate estimation of time to fix is important for project planning because it helps to allocate resources more effectively [89, 95, 102] and also to improve the prediction of costs and time needed for future releases [105]. Accurate time to fix can also be an indicator of software quality. For instance, software artifacts taking too much time to be fixed may have some structural problems [100]. Currently, the approaches for predicting CR fixing time, which we found in the literature, can be divided into statistical prediction models and IR-based models, as discussed next.

4.2.1.1. Change request fixing time as statistical prediction models. These models are presented in Table VIII. Such models rely on features of already solved CR, such as *priority*, *severity*, *reporter reputation*, *number of comments*, *CR state transitions*, *CR dependencies*, and *attachments*. Thus, when new CRs are opened, their features are used to query the model which in turn will predict the fixing time according to the features of existing CRs. However, these models have been questioned by Bhattacharya and Neamtiu [94], where they argued that such CR features are not so useful for predicting fixing time in case of open source projects.

4.2.1.2. Change request fixing time as an information retrieval problem. Information retrieval models are used to predict CR fixing time by using textual similarity, as shown in Table VIII. Therefore, a new CR is treated as an IR query and relevant existing CRs must be retrieved for such a query. Then, the time spent to fix the similar CRs can be used to predict the fixing time for the new one. The approaches have used TF-IDF [97, 105] and LSI [103] for text representation and the cosine similarity function for the reasoning with algebra.

Finally, the algorithms for reasoning with learning, applied in both types of models, include the following: 0-R [19, 95, 102], 1-R [19, 95, 102], naive Bayes [19, 89, 92, 98, 101], multilayer perceptron [98], Bayesian networks [95, 102], C4.5 [19, 89, 95, 102], J48 [98], PART [89], k-nearest neighbor [97, 98, 105], linear regression [93, 99], SVM [98], and logistic regression [19, 95, 96, 102].

DOI: 10.1002/smr

Table VIII. Research on predicting fixing time of change requests (CRs) mapped to an adaptation of the taxonomy created by Canfora and Cerulo [11].

	Representation						Reasoning														Repository		
	Query (input CR)			Document (CR)			With logic			With uncertainty		With learning												
Approach	Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space*	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Association Rules	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code
IR-based Models																							
Weiss *et al.* [105]	✓				1			✓								✓					✓		
Hewett and Kijsanayothin [98]	✓				1							✓		✓	✓	✓	✓				✓		
Hassouna and Tahvildari	✓				1			✓								✓					✓		
Raja [103]	✓				2			✓								✓					✓		
Statistical Prediction Models																							
Song *et al.* [89]															✓		✓	✓			✓		
Hooimeijer and Weimer [99]																			✓		✓		
Panjer [102]															✓		✓		✓		✓		
Anbalagan and Vouk [93]																			✓		✓		
Raja *et al.* [104]																			✓		✓		
Bougie *et al.* [95]															✓		✓		✓		✓		
Guo *et al.* [96]																			✓		✓		
Lamkanfi and Demeyer [101]																	✓				✓		
Abdelmoez *et al.* [92]																	✓				✓		
Hosseini *et al.* [19]															✓		✓		✓		✓		
Xuan *et al.* [106]												✓			✓				✓		✓		
Zimmermann *et al.* [107]																			✓		✓		

*** Types of Vector Space representation for documents:**
(1) TF-IDF (2) Latent Semantic Indexing

IR, information retrieval; CVS, Concurrent Version System; SVN, Subversion; TF-IDF, Term Frequency–Inverse Document Frequency.
[a]Types of vector space representation for documents: (1) TF-IDF and (2) Latent Semantic Indexing.

4.2.2. Determining the impact of change requests. Impact analysis is the activity in which the software artifacts impacted by a maintenance task must be identified [109, 111]. It is a key maintenance activity in software development, because it helps in cost estimation, resource planning, testing, propagation of change, managing ripple effects, and traceability [109, 114]. In the context of CR repositories, maintenance tasks are driven by CRs. Specifically, when a developer is assigned to fix a CR, he/she must search for requirement documents, architectural documents, source code, testing projects, and so on, which is labor intensive and takes too much time [108, 113]. The literature about impact analysis in software development is very broad, in which many artifacts are involved. Thus, our work focused on studies where the CRs are the primary unit of work for the impact analysis.

4.2.2.1. Impact of change requests as an information retrieval problem. Due to the intensive amount of time and work required in impact analysis, tools that can effectively help developers in this aspect are very important [55, 113]. Therefore, many studies investigated CR models to narrow the search space of software artifacts for impact analysis, as shown in Table IX. For these models, the new CR is also regarded as the input query. However, the relevant documents are not past CRs as in other issues. Instead the software artifacts (documents, source code, etc.) impacted by the new CR must be retrieved. They used TF-IDF [109–113, 116, 117, 119, 120], LSI [108, 114, 115], and LDA [55, 117] for document representation, and algorithms for reasoning, including the following: SVM [55, 108, 113, 117, 119], Naive Bayes [119], J48 [108], PageRank [113], and cosine similarity function [55, 109, 110, 113–116, 120].

4.2.3. Predicting the number of future change requests. In addition to determining the impact and fixing time of CRs, the prediction of the number of CRs, which are likely to be reported is another important resource for software projects. Such prediction aims to forecast the number of CRs to be reported in future releases, and it is useful for planning maintenance and evolution activities, such as supporting rational and timely resource allocation, or for improving software development processes [82, 85, 89, 90]. The literature can be divided into machine learning approaches and mathematical models.

DOI: 10.1002/smr

Table IX. Research on predicting impacted artifacts of change requests (CRs) mapped to an adaptation of the taxonomy created by Canfora and Cerulo [11].

	Representation						Reasoning													Repository		
	Query (input CR)			Document (CR)			With logic			With uncertainty		With learning											
Approach	Keyword-based	Pattern-based	Structural	Stream of Characters	Vector Space*	Structural	Logic	Algebra	Graph Theories	Probability Theories	Fuzzy Set Theories	Neural Network	Symbolic Learning	Support Vector Machines	Decision Trees/Table	Lazy Learning	Bayesian Statistics	Association Rules	Regression Analysis	Learn to Rank	CRs (e.g. Bugzilla)	Commit Log (e.g. CVS, SVN)	Source Code
Antoniol *et al.* [109]	✓				1			✓		✓											✓	✓	
Canfora and Cerulo [111]	✓				1					✓											✓	✓	
Canfora and Cerulo [112]	✓				1					✓											✓	✓	
Chen *et al.* [113]	✓				1			✓	✓					✓							✓	✓	
Torchiano and Ricca [118]	✓				4						✓										✓	✓	✓
Ahsan and Wotawa [108]	✓				2			✓						✓	✓		✓				✓	✓	
Gethers *et al.* [114]	✓				2			✓													✓	✓	✓
Nguyen *et al.* [55]	✓				3									✓			✓				✓		✓
Kagdi *et al.* [115]	✓				2			✓													✓	✓	✓
Somasundaram and Murphy [117]	✓				1,3					✓				✓							✓		
Wang *et al.* [119]	✓				1									✓			✓				✓		
Bangcharoensap *et al.* [110]	✓				1			✓													✓	✓	✓
Nguyen *et al.* [116]	✓				1			✓													✓	✓	✓
Zhou *et al.* [120]	✓				1			✓													✓		✓
*** Types of Vector Space representation for documents:** (1) TF-IDF (2) Latent Semantic Indexing (3) Latent Dirichlet Allocation (4) Simple term-by-document matrix																							

CVS, Concurrent Version System; SVN, Subversion; TF-IDF, Term Frequency–Inverse Document Frequency.
[a]Types of Vector Space representation for documents: (1) TF-IDF, (2) Latent Semantic Indexing, (3) Latent Dirichlet Allocation, and (4) simple term-by-document matrix.

With respect to machine learning approaches, Song *et al*. [89] and Radlinski [88] applied association rule mining and Bayesian networks, respectively. Based on mathematical models, Auto-Regressive Integrated Moving Average models [82–85, 91], Cox survival analysis [90], and Weibull distribution [63] are used. Kim *et al*. [86] tested two prediction models, similar to those in [88, 89], by applying a technique to reduce possible noise present in the data used. In both types of approaches, the prediction relied solely on the CR data, with the exception of the studies of Wedel *et al*. [90], which also used data from version control systems.

There are also other ways of predicting that CRs might emerge. For example, instead of predicting the number of CRs, what is predicted is which software parts are likely to be buggy [87]. With this information, managers can better allocate the test effort, that is, ensuring that buggy software parts are given the proper attention. On the other hand, as this approach does not forecast emergent CRs, it is not so useful for planning future resource allocation for post-release maintenance and evolution.

4.3. Change request data

This research area is related to studies with the objective of analyzing CRs data, such as descriptions and users' comments. We identified only one topic in this area, which is the quality assessment of CR data, described next.

4.3.1. Quality assessment of change request data. As presented in previous topics, the information present in CRs data is fundamental to approaches, mainly to those which rely on machine learning techniques. This way, improving the quality of data in CRs can be seen as a starting point for addressing the other problems related to CR repositories. For instance, we have shown earlier in this article that some approaches have integrated data from CR repository and version control systems. However, many researchers pointed out that such integration is not straightforward, owing to noises present in such data [140, 146–148]. An example of noise is a CR, which is not explicitly linked with source code files which were changed during the CR fix [140]. As a result of noise presence, the results from some models, such as those built for CR assignment and effort estimation, can be very adversely affected [86].

DOI: 10.1002/smr

This situation alerts to the fact that it is important to perform some investigation on the data it is intended to be used before building these models, as well as to provide methods that detect and eliminate noise in data from CR repositories, as performed by Kim *et al.* [86]. Other studies investigated CR data to understand how people describe software issues [121, 149] and identify what type of information is essential to a CR [122, 137, 149, 150], which would be useful to improve CR repositories systems or even to train new CR reporters. It is also possible to provide automated methods, using techniques such as machine learning and natural language processing, to predict if the CRs data have enough quality [151, 152], thus identifying poor quality CRs or even suggesting improvements.

Finally, some studies addressed the challenge of extracting relevant information from CRs. Rastkar *et al.* [153] argued that there is a need to reduce the amount of text that the developer has to read in each CR, thus speeding up some tasks related to CR management. Therefore, they provide a reduction technique based on conversation-based generators. Additionally, Bettenburg *et al.* [154] proposed an approach to extract structural information from CRs, such as stack traces, source code, and enumerations. Such information can be used to improve the performance of machine learning algorithms, and, as structural information are components of good quality CRs [122, 150], it can be also used to predict the quality of CRs, as in [151, 152].

5. OPPORTUNITIES: USING DATA FROM CHANGE REQUEST REPOSITORIES TO UNDERSTAND AND IMPROVE SOFTWARE DEVELOPMENT

In this section, we present the opportunities regarding CR repositories. While the problems of CR management were presented in the challenges category, in the previous section, the opportunities are related to studies, which aimed at improving software development through the use of information contained in CR data. This concerns two research areas: *understanding software development* and *improving software quality*. Next, we discuss these two areas and their topics.

5.1. Understanding software development

This includes studies that addressed CR repositories to understand the aspects of the CR management process and software maintenance and evolution.

5.1.1. Change request management process. The CR repositories play an important role in software development, as a common place where activities are coordinated. Indeed, in many software development projects, the CRs are actually the primary units of work [25]. Although most research has investigated such repositories in the context of open source projects, this perception also holds independently of the team size, project type (open or private), or the geographical distribution of developers. Thus, understanding the characteristics of the CR management process can provide insights, which in turn are useful to feed the development of new tools and processes. Specifically, it is possible to investigate data present in CRs to identify concepts, phenomena and relationships related to software management activities [131].

In this context, research on *CR networks* investigates the connections created when developers assert duplication, dependencies, or reference relationships among different CRs [127, 131]. Sandusky and Gasser [131] identified that 65% of the CRs in their study had one or more of these three kinds of relationships. CR networks add virtual structure to the repository, helping to reduce the complexity involved in CR management. Additionally, they also represent the social relationships and interactions, demonstrating the social process inherent in CR management, as pointed out by Bertram *et al.* [8]. For example, negotiation is the kind of interaction which, in the study by Sandusky and Gasser [130], was present in 27% of the CRs analyzed. The information on the interactions can also help to organize the teams in such a way that development activities can be performed more efficiently [129, 132].

Considering the interactions carried out in CR repositories, Crowston and Scozzi [125] investigated coordination practices in open source projects. Their findings point out that the process for CR fixing is quite short and emerges spontaneously, and consists of the following: submitting the CR, analyzing,

DOI: 10.1002/smr

fixing, and closing it. Additionally, the CR assignment in open source projects is not formal, meaning that the developers who have the competencies assign the CRs to themselves. This behavior is different from private projects, where CR assignments are activities that are formally coordinated. Another interesting aspect concerns participation in CR management: in open source projects, only a few developers are responsible for the vast majority of CR reporting and fixing [8, 126].

Guo *et al*. [96] also investigated which aspects of a CR determine its chance of being fixed. In summary, the likelihood of a CR being fixed is proportional to the reputation and interpersonal skills of the reporter, the amount of people interested in the CR, the number of reassignments, and the perceived business impact of the CR, and the textual quality of the CR. Ohira *et al*. [128] conducted a similar research, in which they investigated how patterns of CR fixing impact CR resolution time. They found that: (a) when the reporter is also the developer who analyzes the CRs, the resolution time is 17–47% faster; and (b) when the developer who analyzes a CR assigns it to himself, the assignment takes 48–58% longer. However, such a developer can fix the CR around two times faster than if the CR were to be assigned to another developer. Finally, they concluded that the worst pattern is when the CR reporter is different from the CR analyzer, who in turn is different from the CR fixer.

5.1.2. Software maintenance and evolution. We identified a set of work that investigated data from CR repositories to understand characteristics of the activities for software maintenance and evolution. As mentioned earlier, this is possible because the CR repositories are a live record of these activities. For instance, Burch and Kungs [133] identified that software maintenance occurs in four distinct stages: first, the maintenance is performed for user support; second, maintenance is performed for repairing; third, enhancements take place; and, in the last stage, all of these three stages have low frequency, and the software is to be replaced.

Gupta *et al*. [134] analyzed CRs of a telecom system, which is developed incrementally. They identified a small number of CRs that target requirements, while functionality is enhanced and improved in each release, mainly with respect to the quality attributes. Adaptive and preventive changes also represented a small portion of the CRs. The organization initiates more CRs than the customers or changing environments do. In addition, there is an increasing tendency to accept CRs, and reused and non-reused components have equal change-proneness. Similarly, Edwards [123] found close relationships between the most used screens of the system, the core of business rules, and the CRs reported by customers. That is, these parts of the software are supposed to have more CRs reported.

5.2. Improving software quality

This is a one-topic research area. It investigates CR repositories aiming to improve software quality by using CR data. Next, we discuss the topic on software testing.

5.2.1. Software testing. Because CR descriptions contain important information for software testers (such as steps to reproduce an error, business rules, and pre and post conditions) [155], some studies identified a strong potential in using CR data for improving the software testing process. Thus, for the *Software Testing* category, we grouped studies which did so. Although only four works were found for this challenge, we created a separate topic because there is more room for future research on this topic.

Regarding the techniques used in this category, basically the proposed methods depend on manual analysis of the CR data to extract information for software testing activities. For example, Elcock and Laplante [135] presented an approach to test software that does not have formal requirements, such as open source projects. The main idea of the concept is to analyze development artifacts (CRs and help files, in this case) to serve as input for generating test cases. By using these artifacts, it is possible to reconstruct behavioral requirements, enabling the design and implementation of test cases.

Moritz [136] systematically investigated the CRs reported by customers in order to improve system testing. She described how the outcomes of this study were used by the test team to reduce the number of defects found by customers. In this same context, Yu and Kerong [138] studied CR data from Apache, Eclipse, and Firefox to provide a knowledge representation of software faults. Thereafter,

DOI: 10.1002/smr

an algorithm to transform CRs in software test case is presented, thus avoiding extra effort to construct test cases from scratch.

Finally, Wang *et al*. [137] measured and improved the software testing process based on improvements in the CR management process. According to the authors' study, many CRs could not be fixed due to unclear descriptions. Such a situation produced a communication gap between software testers and developers. Thus, using education actions, the testers were introduced to good practices for CR reporting, which, in turn, led to a significant improvement in the efficiency of testing.

6. TOOLS AND ONLINE SERVICES FOR CHANGE REQUEST MANAGEMENT

As stated in Section 3, we analyzed 10 CR repositories, among standalone software and services, to answer *Question 2*: *Do the tools and online services for CR management address any of the challenges pointed out as a result of the answers to Question 1? If so, how do they address such challenges?*

We carried out the analysis with standalone tools, which need to be installed and configured before being used, and with preconfigured online services available on the Internet. The standalone tools used were as follows: Bugzilla, MantisBT, Trac, Redmine, and Jira. For the online services we used: SourceForge, Launchpad, Code Plex, Google Code, and GitHub. These systems were analyzed regarding the challenges identified in this study, as given in Figure 2 in Section 3.4, namely:

- *CR classification*: assignment of CRs, identification of duplicate CRs, prediction of the CR type, and clustering of similar CRs;
- *CR effort estimation*: predicting the number of CRs, predicting the time to fix CRs, and predicting the impacted artifacts of CRs; and
- *CR data analysis*: quality assessment of CR data.

However, it is worth mentioning that the chances are that the topics which we identified as challenges were not considered as such by the teams of these tools and services. It is even possible that the teams are not aware of these challenges. If so, this article can be used as a guide for the new features of these tools and services. Additionally, this analysis did not consider those topics regarding the opportunities that were identified in the study. We believe that the concept of opportunities with respect to CR repositories is very subjective and could be considered a bias in such analysis.

6.1. Analysis of results

In Table X, we summarize our findings on the analysis of tools and online services. Next, we detail such results.

Regarding the CR classification problem, our analysis identified that there is no tool or online service which provides automatic methods to support the assignment of new CRs, predicts the type of CRs, or clusters similar CRs. For the analysis of duplicate CRs, all tools and services only provide common search features, such as searches using keywords and custom filters. On the other hand, Bugzilla and Trac sites are aware of the duplication problem, which led them to provide some tips for submitters with the objective of avoiding duplicates.

In addition, the Launchpad tool has a feature that addresses a problem not present in our findings: the identification of possible important CRs. According to the Launchpad site, the feature, called 'Bug Heat',

> Is a calculated estimate of its (CRs) importance, using factors such as how many people are subscribed, whether it is a security issue, how many people have marked the bug as affecting them, and so on. [156]

We did not make any analysis of the effectiveness and efficiency of Bug Heat, because it would be beyond the scope of this study.

Regarding the effort estimation problem, all tools and services provide some traceability by linking CRs to commit logs of version control systems. These links are made by inserting the number of the CR in the commit log message and vice versa. The presence of the traceability link is an important feature for the automated methods found in the literature regarding effort estimation. However, no

DOI: 10.1002/smr

Table X. Analysis of the selected tools and online services.

Tools/services	Reference	Challenges addressed	Techniques (how the challenge is addressed)
Bugzilla	—	1 – Duplicate CR analysis	1 – Common search features; also provides a Web page with the most submitted CRs and warns to look at it before submitting a new CR
		2 – Quality of CR data (in part)	2 – Provides additional fields for CR data
MantisBT	—	1 – Duplicate CR analysis	1 – Common search features
		2 – Quality of CR data (in part)	2 – Provides additional fields for CR data
Trac	—	1 – Duplicate CR analysis	1 – Similar to Bugzilla
Redmine	—	1 – Duplicate CR analysis	1 – Common search features
Jira	—	1 – Duplicate CR analysis	1 – Common search features
SourceForge	—	1 – Duplicate CR analysis	1 – Common search features
Launchpad	Revell [156]	1 – Duplicate CR analysis	1 – Common search features; also sorts the CRs in a search result according to the number of comments
Code Plex	—	1 – Duplicate CR analysis	1 – Common search features
Google Code	—	1 – Duplicate CR analysis	1 – Common search features
		2 – Quality of CR data (in part)	2 – Provides additional fields for CR data
GitHub	—	1 – Duplicate CR analysis	1 – Common search features

CR, change request.

tool/service provides any way for predicting the effort automatically, maybe because such traceability practice is a workaround, which is very susceptible to mistakes [146, 148].

For the problem of quality assessment of CRs data, we considered in our analysis the attributes defined according to the study by Bettenburg *et al*. [155], which surveyed software developers and aimed at identifying which attributes make a good CR description. This survey was performed by interviewing developers and other people involved with CR management. According to the results, they found that descriptions containing *steps to reproduce*, *stack traces*, and *test cases* are helpful for developers. Thus, as it is considered a tool/service that promotes quality in CR data it should explicitly provide means that lead to CR submitters specifying these attributes.

Thus, in our analysis, only the MantisBT tool provides a specific field in its CR form to define the attribute *steps to reproduce*. The Google code service and the Bugzilla tool provide by default a template in their CR description fields indicating that submitters should provide the attributes *steps to reproduce* and *stack traces*. Therefore, neither tools nor services provide functionalities to achieve all the CR quality attributes identified by Bettenburg *et al*. [155].

7. DISCUSSION AND OPEN ISSUES

This section presents the observations extracted from the mapping study outcomes.

7.1. A taxonomy for investigating change request repositories

One of the contributions of this study is the classification scheme based on research areas and topics, which in turn are organized into challenges and opportunities. Actually, this scheme forms a taxonomy to map the studies about CR repositories. Mapping these studies into this taxonomy, as we did in this article, shows the research maturity and the open problems of each topic.

In addition, it is worth mentioning that there are two research areas in this taxonomy with only one topic each: CR data and improving software quality. An alternative approach could be to represent only the

DOI: 10.1002/smr

research areas, namely, it is not necessary to have topics. Indeed, if we were sure that these research areas would not expand into more topics, such a representation would fit better. However, we expect that more topics may well appear with regard to these areas due to the increasing interest in investigating CR repositories. Thus, new topics on these areas can be easily mapped out to our taxonomy.

7.2. Main findings on the approaches to solve the challenges of change request management

There is a variety of approaches applied to the diversity of challenges identified on CR repositories, which were evidenced when we mapped them to the taxonomy of IR models. These approaches are basically suggestion techniques for a given search space (software repositories), and we can divide them into: *semiautomated* techniques, which need the presence of a human to take the final decision on the suggestions and *automated* techniques, for which the first element of the suggestion list is picked. For both, semi-automated and automated approaches, there are different techniques being used for text processing, data mining, natural language processing, as well as statistical based techniques for machine learning.

In addition, although this study investigates the challenges on CR repositories, in many cases, we noticed that different software repositories are jointly used to build the approaches, such as version control systems (such as Subversion and Concurrent Version System) and mailing lists. Indeed, depending on the challenge being addressed, it is mandatory to integrate different types of repository. For instance, in order to provide automated impact analysis (discussed in Section 4.2.2), it is necessary to extract information from both CR and source code repositories.

In other cases, the presence of another type of repository is optional to improve the accuracy of the approach. This is the case of the CR assignment approaches (Section 4.1.1), where the suggestions (of developers) for the assignment can be made using the CR repository alone, or it can be combined with source code repository to boost the suggestions. Although many different approaches have been proposed over time, there are still many possibilities that can be explored. This is clear in the taxonomy for IR models provided for each challenge. Such a taxonomy showed that there is room for more combinations of techniques and software repositories. For the repositories, those for staff and project management, requirements, architectural documents, test cases, and so on could be explored.

Regarding the techniques, a means to leverage current results is to mix up the approaches used in different challenges. For instance, we observed that most of the approaches for CR assignment use machine learning techniques, whereas duplicate CRs analysis does not. Therefore, we hypothesize that applying machine learning techniques to duplicate analysis could lead to better results. Nevertheless, first, it is necessary to analyze if the CR data considered is suitable for such a technique, as we observed in Section 4.1.2.

7.3. The lack of contextual information in the approaches

This study only identified research that defined approaches with different IR models and that reported their results based on off-line tests with data from historical repositories. It means that it is difficult to understand if they are suitable for the dynamics of real software development projects. Therefore, understanding the key aspects of the challenges is another important issue, which should be addressed in future research, such as that carried out in [31, 141].

For instance, considering the CR assignment challenge, it is necessary to understand the heuristics used by CR assigners to assign software developers to a CR [141]. This could be performed by answering the following questions: Do they consider the developer work load before assigning? Do they consider the developer knowledge on the CR issue or his/her engagement on the project? Do the priority or political aspects influence their decision?

Similar questions could be set for each challenge identified in this work. The answers to these questions would help the investigation of new and more effective approaches.

Additionally, it is important to observe that most of the approaches found in the literature cannot be seamlessly applied to all projects. This is a direct implication of the contextual information needed by each approach. The contextual information varies in accordance with the process used for software development. Thus, as many software projects use tailored processes, the approaches for CR repository challenges need to be tuned according to the software

DOI: 10.1002/smr

project. Indeed, the configuration of the approaches according to the heuristics of the project will ensure better results when addressing the challenges [52].

7.4. Missing attributes in change request classification approaches

As we identified, the research studies on the topic of CR classification are centered on assignment of CRs, duplicate analysis, CR type guessing, CR prioritization, and clustering. However, the classification approaches can be extended to classify CRs according to other attributes that currently need to be filled manually, such as the following: operating system type, software version, severity, target component, and hardware type. This new classification would be especially useful for helping less experienced reporters when submitting new CRs, because the information required in these attributes may not be intuitive for them. Thus, there is room for research addressing other types of classification and new techniques.

7.5. Stop fixing the symptoms

Some studies mention that the CR descriptions have not been written properly in most of the projects investigated [122, 157]. Actually, poorly described CRs may well be the reason for many of the challenges identified. For instance, it is possible that poor descriptions may facilitate the submission of duplicate CRs and hamper the activities of classifying CRs, estimating the efforts necessary to fix a CR, analyzing the impact of a CR, and even fixing the CR. Thus, improving of the quality of CRs descriptions in the first place is likely to help in addressing these challenges.

In the same way, some CR repositories issues, such as the challenges identified in this study, might be related to a poorly defined process for CR management. For instance, a process that does not consider a step for searching for existing CRs before opening a new one could facilitate the appearance of duplicate CRs. Thus, CR management processes can be designed considering these challenges in order to address or avoid them.

If we consider this point of view, in which poor CR descriptions and processes are the root causes for the challenges, current approaches (i.e., those using IR models) are actually fixing the symptoms. Then, it is necessary to investigate the real impact of poor quality CRs and CR management processes on these challenges. If such an impact is significant, maybe it is worth investing in the quality of the CR management process as a whole rather than in investigating approaches that only fix these symptoms. A good starting point in this direction would be to investigate the best practices applied in projects where the challenges related to CR management are well mitigated or not present, as conducted by Stojanov *et al*. [158].

7.6. Difficulty in assessing the approaches

The vast majority of the approaches uses the measures *precision* and/or *recall* [139] to evaluate their results, and few of them have used the concept of *accuracy* where it is not clear how it fits in with precision and recall. Additionally, approaches have been tested using different historical data from software repositories obtained from open source and/or private projects. That is, those studies do not use a common measuring approach or a common database for evaluation, which is strictly necessary to compare the results of different studies. Thus, any attempt to compare such results would be biased.

Therefore, we believe that there is a need to construct a uniform *corpus* [139] with software data (CRs, source code, emails, etc.) so as to evaluate approaches. This corpus would enable researchers to test their methods by using the same corpus as other researchers, which would eliminate the bias in research that evaluate its methods using different data. Furthermore, by using the same corpus, the comparison of methods based on results from the literature would be possible. The Predictive Models in Software Engineering [124] are a step forward in this direction, because accepted papers must propose repeatable models.

7.7. The state of the art is still far from the state of the practice

In this study, we analyzed tools and online services for CR repositories, regarding the implementation of approaches addressing the challenges identified. As observed in the analysis, the knowledge from

DOI: 10.1002/smr

ideas implemented in the state of the art has not been transferred to the state of the practice. This aspect came from the fact that no tool or service has implemented any of the approaches proposed. We consider two possible explanations for this situation: first, we can believe that practitioners are not very enthused by the current results demonstrated by the approaches proposed in the state of the art; second, it is possible that the impact of CR management issues on software development are not perceived by practitioners, or they consider it is not worth investing in.

We believe that it is necessary to perform some investigation to assess the truth of these hypotheses. If the first explanation is true, then more research on better approaches to address the CR management issues is to be valued. If the second one is true, then, it will be necessary to put more research effort into determining the real impact of these issues on software development and to argue for undertaking more investigations in this direction being considered worthwhile. Actually, the second explanation must be investigated in the first place, because we have few studies about the impact of these issues on software development.

8. THREATS TO VALIDITY AND LIMITATIONS

This mapping study was performed following a defined process [22], which allows its replication and extension. Similar results can be achieved if the same process is carefully followed. However, some aspects and external factors that can influence the results, such as human errors, are beyond the capabilities of the process. Thus, we recognized some threats to the validity of this study, as presented next. Together with each threat, we briefly describe the mitigation strategy used for this study.

- *Research questions*: It is possible that the defined research questions might not completely cover the field of CR repositories. However, some discussions with project members and Software Maintenance experts were carried out in order to validate the questions. Thus, even we did not consider the optimum set of questions, we attempted to address the most frequently asked and the open issues in the field, from both the practitioner's and researcher's points of view.
- *Publication bias*: It is not possible to guarantee that all relevant primary studies were selected and some relevant studies cannot be chosen during the search process. This threat was mitigated by following references in the primary studies, in a technique called *snowballing* [22]. However, because of the volume of papers collected, we believe that research in the field is well represented. In other words, the possible missing papers would not have significant impact on the results of this mapping study.
- *Search conducted*: As the digital databases do not work with search rules that are compatible with each other, all search strings were adapted and calibrated for each digital database. Nevertheless, we did not know all the rules that digital databases use to search for a document. This was mitigated by running the search in 13 digital databases, as detailed in Section 3.
- *Data extraction and classification*: During the extraction process, the studies were classified on the basis of our judgment. However, despite double checking, some studies could have been classified incorrectly. Additionally, sometimes it was difficult to classify the studies, according to research areas and topics, due to the narrowness among challenges and opportunities. In order to mitigate this, the classification process was repeated by each author of this study and the results were jointly discussed in order to reach a consensus.

A specific limitation of this work is the fact that it did not make any comparison considering the effectiveness and efficiency of the approaches identified in the papers. We believe that this comparison deserves a specific research topic.

9. RELATED WORK

Kagdi *et al*. [159] conducted a literature review on approaches for mining software repositories concerning software evolution and maintenance. The result was a taxonomy based on four dimensions: the type of repository mined (*what*), the purpose (*why*), the proposed method (*how*),

DOI: 10.1002/smr

and the evaluation method (*quality*). However, their taxonomy does not provide an extensive understanding about the investigations on CR repositories, mainly for two reasons. First, although they surveyed much work and made a consistent taxonomy, this was carried out as long ago as in 2006 and since then many other works have been published, considering new topics and approaches. Second, according to their exclusion criteria for studies, they were very concerned about studies that approached evolutionary changes of software artifacts by investigating multiple software repositories. As a consequence, many studies that used data from a single repository were beyond their scope, such as those which relied only on data from CR repositories.

On the other hand, our systematic mapping study narrowed the focus to investigations on CR repositories, so that we could provide an extensive overview of the state of the art on this topic. The dimensions proposed by Kagdi *et al*. [159] were used to guide the analysis of the studies that we considered in this article. Besides, we also analyzed some tools and services to understand the technology transfer when investigating CR repositories. Another difference from Kagdi *et al*. [159] is that their taxonomy considers the techniques and methods for mining software repositories as first-class entities, whereas our analysis focused on the challenges and opportunities related to CR repositories. Thus, we moved from the details of the *How* to the details of the *Why* dimension, which results in a taxonomy of the research areas of CR repositories. In addition, it is worth mentioning that, although we changed the focus of the taxonomy, we did not neglect the technical aspects of the approaches identified in each topic of the taxonomy.

10. CONCLUDING REMARKS AND FUTURE WORK

The CR repositories play a fundamental role in software development projects. They can facilitate the communication between developers, testers, and users and also work as a coordination mechanism for many stakeholders and the software team [8, 160]. As these repositories become more prevalent in software projects, there is an increasing concern in issues related to CR management. Thus, in this paper, we conducted a systematic mapping study [22] to leverage investigation of the state of the art on CR repositories and analyzed 142 studies. We identified a set of issues that were classified in accordance with two classification schemes.

In the first classification scheme, we divided the issues into two main categories: opportunities and challenges. These categories were further divided into research areas, which in turn comprise different topics. This scheme actually forms a taxonomy for investigation on CR repositories.

On the other hand, the second scheme classified the topics related to challenges in accordance with a taxonomy for IR models [11], because these topics provide approaches to aid CR management by using IR models. The taxonomy mapped these approaches as per the IR algorithms, techniques for query/document representation and kinds of repositories.

Apart from describing the state of the art on investigating CR repositories, this paper pointed out the achievements of each topic, as well as the open issues that must be considered in new research. Actually, through the classification schemes and the analysis we provided, it is easy to understand how the topics have been investigated, which is fundamental to drive the development of new approaches. Thus, the information gathered in this study is useful for both researchers and practitioners. Researchers can find lessons learned, open issues, and the road ahead as to investigating CR repositories. Practitioners will also find technical references for their day-to-day work, which can be valuable for improving their own processes and tools.

For future research, we are planning to combine the evidence found in this study and evidence identified in controlled experiments and industrial software maintenance projects to define hypotheses, which will be the basis of designing new methods, processes, and tools. Specifically, there is a plan for building a framework in which it is possible to take advantage of the opportunities inherited from using CR repositories and to address the problems faced when using them.

DOI: 10.1002/smr

APPENDIX A: SEARCH STRINGS

Table A.1 Search string per search engine.

Search engine	Search string
Google Scholar	Bug report OR track OR triage “change request” issue track OR request OR software OR “modification request” OR “defect track” OR “software issue” repositories maintenance evolution
ACM Portal	Abstract: ”bug report” or Abstract:”change request” or Abstract:”bug track” or Abstract:”issue track” or Abstract:”defect track” or Abstract:”bug triage” or Abstract: ”software issue” or Abstract: ”issue request” or Abstract: ”modification request”) and (Abstract:software or Abstract:maintenance or Abstract:repositories or Abstract:repository
IEEExplorer (1)	((((((((((”Abstract”: ”bug report”) OR ”Abstract”:”change request”) OR ”Abstract”:”bug track”) OR ”Abstract”:”software issue”) OR ”Abstract”:”issue request”) OR ”Abstract”:”modification request”) OR ”Abstract”:”issue track”) OR ”Abstract”:”defect track”) OR ”Abstract”:”bug triage”) AND ”Abstract”:software)
IEEExplorer (2)	((((((((((”Abstract”: ”bug report”) OR ”Abstract”:”change request”) OR ”Abstract”:”bug track”) OR ”Abstract”:”software issue”) OR ”Abstract”:”issue request”) OR ”Abstract”:”modification request”) OR ”Abstract”:”issue track”) OR ”Abstract”:”defect track”) OR ”Abstract”:”bug triage”) AND ”Abstract”:maintenance)
IEEExplorer (3)	((((((((((”Abstract”: ”bug report”) OR ”Abstract”:”change request”) OR ”Abstract”:”bug track”) OR ”Abstract”:”software issue”) OR ”Abstract”:”issue request”) OR ”Abstract”:”modification request”) OR ”Abstract”:”issue track”) OR ”Abstract”:”defect track”) OR ”Abstract”:”bug triage”) AND ”Abstract”:repositories)
IEEExplorer	((((((((((”Abstract”: ”bug report”) OR ”Abstract”:”change request”) OR ”Abstract”:”bug track”) OR ”Abstract”:”software issue”) OR ”Abstract”:”issue request”) OR ”Abstract”:”modification request”) OR ”Abstract”:”issue track”) OR ”Abstract”:”defect track”) OR ”Abstract”:”bug triage”) AND ”Abstract”: repository)
Citeseer Library	(abstract: ”bug report” OR abstract:”change request” OR abstract:”bug track” OR abstract:”issue track” OR abstract:”defect track” OR abstract:”bug triage” OR abstract: ”software issue” OR abstract: ”issue request” OR abstract: ”modification request”) AND (abstract:software OR abstract:maintenance OR abstract:repositories OR abstract: repository)
Elsevier	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software OR maintenance OR repositories OR repository)
Scirus	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software maintenance OR repositories OR repository) ANDNOT (medical OR aerospace)
ScienceDirect	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”issue request” OR ”modification request”) AND LIMIT-TO(topics, ”software”)
Scopus	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software maintenance OR repositories OR repository)
Wiley	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software maintenance OR repositories OR repository)
ISI Web of Knowledge	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software maintenance OR repositories OR repository) ANDNOT (medical OR aerospace)
SpringerLink	(”bug report” OR ”change request” OR ”bug track” OR ”issue track” OR ”defect track” OR ”bug triage” OR ”software issue” OR ”issue request” OR ”modification request”) AND (software maintenance OR repositories OR repository) ANDNOT (medical OR aerospace)

DOI: 10.1002/smr

APPENDIX B: LIST OF CONFERENCES AND JOURNALS

Table B.1 List of conferences in which the searches were performed.

Acronym	Conference
APSEC	Asia Pacific Software Engineering Conference
ASE	IEEE/ACM International Conference on Automated Software Engineering
CSMR	European Conference on Software Maintenance and Reengineering
ESEC	European Software Engineering Conference
ESEM	International Symposium on Empirical Software Management and Measurement
ICSE	International Conference on Software Engineering
ICSM	International Conference on Software Maintenance
ICST	International Conference on Software Testing
InfoVis	IEEE Information Visualization Conference
KDD	ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
MSR	Working Conference on Mining Software Repositories
OOPSLA	Object-Oriented Programming, Systems, Languages and Applications
QSIC	International Conference On Quality Software
SAC	ACM Symposium on Applied Computing
SEAA	EUROMICRO Conference on Software Engineering and Advanced Applications
SEDE	19th International Conference on Software Engineering and Data Engineering
SEKE	International Conference on Software Engineering and Knowledge Engineering

Table B.2 List of journals in which the searches were conducted.

Journal title
ACM Transactions on Software Engineering and Methodology
Automated Software Engineering
Elsevier Information and Software Technology
Elsevier Journal of Systems and Software
Empirical Software Engineering
IEEE Software
IEEE Computer
IEEE Transactions on Software Engineering
International Journal of Software Engineering and Knowledge Engineering
Journal of Software: Evolution and Process
Software Quality Journal
Journal of Software
Software Practice and Experience Journal

ACKNOWLEDGEMENTS

Thanks to the Brazilian Federal Organization for Data Processing (SERPRO[†]) for the support on the execution of this research and to the JSEP's reviewers for the valuable feedback. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES[‡]), funded by CNPq grants 305968/2010-6, 559997/2010-8, and 474766/2010-1; FACEPE grants 573964/2008-4 and APQ-1037-1.03/08; and FAPESB.

REFERENCES

1. IEEE Computer Society. Software Engineering Body of Knowledge (SWEBOK). IEEE Press, EUA: Washington, 2004.
2. Pigoski TM. Practical Software Maintenance: Best Practices for Managing your Software Investment. John Wiley & Sons, Inc.: New York, NY, USA, 1996.
3. Eastwood A. Firm fires shots at legacy systems. *Computing Canada* 1993; **19**(2):17.
4. Erlikh L. Leveraging legacy system dollars for e-business. *IT Professional* 2000; **2**(3):17–23.
5. Huff F. Information systems maintenance. *The Business Quarterly* 1990; **1**(55):30–32.
6. Moad J. Maintaining the competitive edge. *Datamation* 1990; **4**(36):61–62.

[†]http://www.serpro.gov.br
[‡]http://www.ines.org.br

DOI: 10.1002/smr

7. Sommerville I. Software Engineering (8th edn). Addison Wesley: Boston, 2007.
8. Bertram D, Voida A, Greenberg S, Walker R. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. *Proc. 2010 ACM Conf. on Computer Supported Cooperative Work (CSCW'2010)*. ACM, 2010; 291–300.
9. Anvik J, Hiew L, Murphy GC. Coping with an open bug repository. *Proc. 2005 OOPSLA workshop on Eclipse technology eXchange (OOPSLA'2005)*. ACM, 2005; 35–39.
10. Cavalcanti YC, da Mota Silveira Neto PA, Lucrédio D, Vale T, de Almeida ES, de Lemos Meira SR. The bug report duplication problem: an exploratory study. *Software Quality Journal* 2011; Online first.
11. Canfora G, Cerulo L. A taxonomy of information retrieval models and tools. *Journal of Computing and Information Technology* 2004; **12**(3):175–194.
12. Bugzilla. 1998. Available from: https://www.bugzilla.org [Accessed on 11 May 2013].
13. Mantis Bug Tracker. 2000. Available from: https://www.mantisbt.org [Accessed on 11 May 2013].
14. Redmine. 2006. Available from: https://www.redmine.org [Accessed on 11 May 2013].
15. The Trac project. 2003. Available from: https://trac.edgewall.org [Accessed on 11 May 2013].
16. Ihara A, Ohira M, Matsumoto K. An analysis method for improving a bug modification process in open source software development. *Proc. Joint Int. and Annu. ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*. ACM, 2009; 135–144.
17. Lucca GAD, Penta MD, Gradara S. An approach to classify software maintenance requests. *Proc. 18th IEEE Int. Conf. on Software Maintenance (ICSM'02)*. University of Sannio, Italy, IEEE, 2002.
18. Aljarah I, Banitaan S, Abufardeh S, Jin W, Salem S. Selecting discriminating terms for bug assignment: a formal analysis. *Proc. 7th Int. Conf. on Predictive Models in Software Engineering*. ACM, 2011.
19. Hosseini H, Nguyen R, Godfrey MW. A market-based bug allocation mechanism using predictive bug lifetimes. *Proc. 16th European Conf. on Software Maintenance and Reengineering (CSMR'2012)*, 2012; 149–158.
20. Anvik J, Hiew L, Murphy GC. Who should fix this bug? *Proc. 28th Int. Conf. on Software engineering (ICSE'2006)*, vol. **2006**, 2006; 361–370.
21. Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. *Proc. 7th joint meeting of the European software engineering Conf. and the ACM SIGSOFT Symp. on The foundations of software engineering (ESEC/FSE'2009)*, 2009; 111–120.
22. Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. *EASE '08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. University of Bari, Italy, 2008.
23. Ahsan SN, Ferzund J, Wotawa F. Automatic classification of software change request using multi-label machine learning methods. *Proc. 2009 33rd Annu. IEEE Software Engineering Workshop (SEW'2009)*. IEEE, 2009a; 79–86.
24. Ahsan SN, Ferzund J, Wotawa F. Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine. *Proc. 2009 4th Int. Conf. on Software Engineering Advances (ICSEA'09)*. IEEE, 2009b; 216–221.
25. Anvik J, Murphy GC. Determining implementation expertise from bug reports. *Proc. 4th Int. Workshop on Mining Software Repositories (MSR'2007)*, 2007.
26. Bettenburg N, Premraj R, Zimmermann T, Kim S. Duplicate bug reports considered harmful… really? *Proc. 24th IEEE Int. Conf. on Software Maintenance (ICSM'2008)*, 2008a; 337–345.
27. Bhattacharya P, Neamtiu I, Shelton CR. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software* 2012; **85**(10): 2275–2292.
28. Canfora G, Cerulo L. Supporting change request assignment in open source development. *Proc. ACM Symp. on Applied Computing (SAC'2006)*, vol. **2**, 2006b; 1767–1772.
29. Chen L, Wang X, Liu C An approach to improving bug assignment with bug tossing graphs and bug similarities. *Journal of Software* 2011; **6**(3):421–427.
30. Cubranic D, Murphy GC. Automatic bug triage using text categorization. *Proc. 16th Int. Conf. on Software Engineering & Knowledge Engineering (SEKE'2004)*, 2004; 92–97.
31. Guo P, Zimmermann T, Nagappan N, Murphy B. Not my bug! and other reasons for software bug report reassignments. *Proc. ACM 2011 Conf. on Computer Supported Cooperative Work*. ACM, 2011; 395–404.
32. Jain V, Rath A, Ramaswamy S. Field weighting for automatic bug triaging systems. *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2012)*. IEEE, 2012; 2845–2848.
33. Kagdi H, Gethers M, Poshyvanyk D, Hammad M. Assigning change requests to software developers. *Journal of Software: Evolution and Process* 2012a; **24**(1):3–33.
34. Lin Z, Shu F, Yang Y, Hu C, Wang Q. An empirical study on bug assignment automation using Chinese bug data. *Proc. 2009 3rd Int. Symp. on Empirical Software Engineering and Measurement (ESEM'2009)*. IEEE, 2009; 451–455.
35. Linares-Vásquez M, Hossen K, Dang H, Kagdi H, Gethers M, Poshyvanyk D. Triaging incoming change requests: bug or commit history, or code authorship? *Proc. of IEEE Int. Conf. on Software Maintenance (ICSM'12)*, 2012.
36. Matter D, Kuhn A, Nierstrasz O. Assigning bug reports using a vocabulary-based expertise model of developers. *Proc. 2009 6th IEEE Int. Working Conf. on Mining Software Repositories (MSR'2009)*. IEEE, 2009; 131–140.
37. Moin A, Neumann G. Assisting bug triage in large open source projects using approximate string matching. *Proc. 7th Int. Conf. on Software Engineering Advances (ICSEA'2012)*, 2012.
38. Nagwani N, Verma S. Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. *Proc. 2011 9th Int. Conf. on ICT and Knowledge Engineering*. IEEE, 2012; 113–117.

DOI: 10.1002/smr

39. Nasim S, Razzaq S, Ferzund J. Automated change request triage using alpha frequency matrix. *Proc. 2011 Frontiers of Information Technology (FIT'2011)*. IEEE, 2011; 298–302.
40. Rahman MM, Ruhe G, Zimmermann T. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. *Proc. 2009 3rd Int. Symp. on Empirical Software Engineering and Measurement (ESEM'2009)*, 2009; 439–442.
41. Tamrawi A, Nguyen T, Al-Kofahi J, Nguyen T. Fuzzy set and cache-based approach for bug triaging. *Proc. 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of software engineering (SIGSOFT/FSE'2011)*. ACM, 2011; 365–375.
42. Tan S, Hu S, Chen L. A framework of bug reporting system based on keywords extraction and auction algorithm. *Proc. Fifth Annu. ChinaGrid Conf. (ChinaGrid'2010)*. IEEE, 2010; 281–284.
43. Xuan J, Jiang H, Ren Z, Yan J, Luo Z. Automatic bug triage using semi-supervised text classification. *Proc. 22nd Int. Conf. on Software Engineering & Knowledge Engineering (SEKE'2010)*, 2010.
44. Xuan J, Jiang H, Ren Z, Zou W. Developer prioritization in bug repositories. *Proc. 34th Int. Conf. on Software Engineering (ICSE2012)*, 2012b; 25–35.
45. Zhang T, Lee B. How to recommend appropriate developers for bug fixing? *Proc. 36th Annu. Int. Computer Software & Applications Conf. (COMPSAC2012)*, 2012.
46. Zou W, Hu Y, Xuan J, Jiang H. Towards training set reduction for bug triage. *Proc. 2011 IEEE 35th Annu. Computer Software and Applications Conf. (COMPSAC'2011)*. IEEE, 2011; 576–581.
47. Cavalcanti YC. BTT–towards a bug triage tool. *Master's thesis*, Universidade Federal de Pernambuco, 2009.
48. da Cunha CEA, Cavalcanti YC, da Mota Silveira Neto PA, de Almeida ES, de Lemos Meira SR. A visual bug report analysis and search tool. *Proc. 22nd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'2010)*, 2010; 742–747.
49. Davidson J, Mohan N, Jensen C. Coping with duplicate bug reports in free/open source software projects. *Proc. IEEE Symp. on Visual Languages and Human-Centric Computing (VL/HCC'2011)*. IEEE, 2011; 101–108.
50. Hiew L. Assisted detection of duplicate bug reports. *M.sc. Thesis*, The University of British Columbia, 2006.
51. Jalbert N, Weimer W. Automated duplicate detection for bug tracking systems. *Proc. 38th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN'2008)*, 2008; 52–61.
52. Kaushik N, Tahvildari L. A comparative study of the performance of IR models on duplicate bug detection. *Proc. 16th European Conf. on Software Maintenance and Reengineering (CSMR'2012)*. IEEE, 2012; 159–168.
53. Liu K, Tan HBK, Chandramohan M. Has this bug been reported? *Proc. ACM SIGSOFT 20th Int. Symp. on the Foundations of Software Engineering (FSE'2012)*. ACM, 2012; 1.
54. Nagwani NK, Singh P. Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs. *Proc. Int. Conf. on Advances in Computing, Communication and Control (ICAC3'2009)*, 2009; 202–207.
55. Nguyen A, Nguyen T, Al-Kofahi J, Nguyen H, Nguyen T. A topic-based approach for narrowing the search space of buggy files from a bug report. *Proc. 26th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'2011)*. IEEE, 2011; 263–272.
56. Prifti T, Banerjee S, Cukic B. Detecting bug duplicate reports through local references. *Proc. 7th Int. Conf. on Predictive Models in Software Engineering (PROMISE'2011)*. ACM, 2011.
57. Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. *Proc. 29th ACM/IEEE Int. Conf. on Software Engineering (ICSE'2007)*, 2007; 499–510.
58. Sun C, Lo D, Wang X, Jiang J, Khoo S-C. A discriminative model approach for accurate duplicate bug report retrieval. *Proc. 32nd ACM/IEEE Int. Conf. on Software Engineering (ICSE'2010)*. ACM, vol. **1**, 2010; 45–54.
59. Sun C, Lo D, Khoo S-C, Jiang J. Towards more accurate retrieval of duplicate bug reports. *Proc. 2011 26th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'2011)*. IEEE, 2011; 253–262.
60. Sureka A, Jalote P. Automatic duplicate bug report detecting using character n-gram-based features. *Proc. 17th Asia Pacific Software Engineering Conf. (APSEC'2010)*, 2010.
61. Tian Y, Sun C, Lo D. Improved duplicate bug report identification. *Proc. 16th European Conf. on Software Maintenance and Reengineering (CSMR'2012)*, 2012; 385–390.
62. Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. *Proc. 30th ACM/IEEE Int. Conf. on Software engineering (ICSE'2008)*. ACM, 2008; 461–470.
63. Wu L, Xie B, Kaiser GE, Passonneau RJ. Bugminer: software reliability analysis via data mining of bug reports. *Proc. 22nd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'2011)*, 2011; 95–100.
64. Zhang T, Lee B. A bug rule based technique with feedback for classifying bug reports. *Proc. IEEE 11th Int. Conf. on Computer and Information Technology (CIT'2011)*. IEEE, 2011; 336–343.
65. Zhou J, Zhang H. Learning to rank duplicate bug reports. *Proc. 21st ACM Int. Conf. on Information and knowledge management (CIKM'2012)*. ACM, 2012.
66. Antoniol G, Ayari K, Penta MD, Khomh F, Guéhéneuc Y. Is it a bug or an enhancement? A text-based approach to classify change requests. *Proc. 2008 Conf. Center for Advanced Studies on Collaborative Research: meeting of minds (CASCON'08)*, 2008.
67. Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. *Technical Report Section XII*, Universität des Saarlandes, Saarbrücken, Germany, 2012a.
68. Kadar C, Wiesmann D, Iria J, Husemann D, Lucic M. Automatic classification of change requests for improved it service quality. *Proc. 2011 Annu. SRII Global Conf. (SRII'2011)*. IEEE, 2011; 430–439.

DOI: 10.1002/smr

69. Chaturvedi K, Singh V. Determining bug severity using machine learning techniques. *Proc. CSI 6th Int. Conf. on Software Engineering (CONSEG'2012)*. IEEE, 2012; 1–6.
70. Gegick M, Rotella P, Xie T. Identifying security bug reports via text mining: an industrial case study. *Proc. 7th IEEE Working Conf. on Mining Software Repositories (MSR'2010)*, 2010; 11–20.
71. Herraiz I, German DM, Gonzalez-Barahona JM, Robles G. Towards a simplification of the bug report form in eclipse. *Proc. 2008 Int. working Conf. on Mining software repositories (MSR'2009)*, 2008; 145–148.
72. Kanwal J, Maqbool O. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology* 2012; **27**:397–412. 10.1007/s11390-012-1230-3
73. Lamkanfi A, Demeyer S, Giger E, Goethals B. Predicting the severity of a reported bug. *Proc. 7th IEEE Working Conf. on Mining Software Repositories (MSR'2010)*, 2010; 1–10.
74. Lamkanfi A, Demeyer S, Soetens Q, Verdonck T. Comparing mining algorithms for predicting the severity of a reported bug. *Proc. 2011 15th European Conf. on Software Maintenance and Reengineering (CSMR'011)*. IEEE, 2011; 249–258.
75. Menzies T, Marcus A. Automated severity assessment of software defect reports. *Proc. 24th IEEE Int. Conf. on Software Maintenance (ICSM'2008*), 2008; 346–355.
76. Sureka A, Indukuri KV. Linguistic analysis of bug report titles with respect to the dimension of bug importance. *Proc. Third Annu. ACM Bangalore Conf.* (*COMPUTE'2010*). ACM, 2010; 1–6.
77. Kumar Nagwani N. CLUBAS: An algorithm and Java based tool for software bug classification using bug attributes similarities. *Journal of Software Engineering and Applications* 2012; **05**(06):436–447.
78. Rus V, Nan X, Shiva SG, Chen Y. Clustering of defect reports using graph partitioning algorithms. *Proc. 21st Int. Conf. on Software Engineering & Knowledge Engineering* (*SEKE'2009*), 2009; 442–445.
79. Santana A, Silva J, Muniz P, Araújo F, de Souza RMCR. Comparative analysis of clustering algorithms applied to the classification of bugs. *Lecture Notes in Computer Science* 2012; **7667**:592–598.
80. Chen I-X, Li C-H, Yang C-Z. Mining co-location relationships among bug reports to localize fault-prone modules. *IEICE Transactions on Information and Systems* 2010; **93**:1154–1161.
81. Ekanayake J, Tappolet J, Gall H, Bernstein A. Time variance and defect prediction in software projects. *Empirical Software Engineering* 2012; **17**:348–389.
82. Goulão M, Fonte N, Wermelinger M, Abreu FB. Software evolution prediction using seasonal time analysis: a comparative study. *Proc. 16th European Conf. on Software Maintenance and Reengineering* (*CSMR'2012*), 2012; 213–222.
83. Herraiz I. A statistical examination of the evolution and properties of libre software. *Proc. IEEE Int. Conf. on Software Maintenance* (*ICSM'2009*), 2009; 439–442.
84. Jung H-W, Lim Y, Chung C-S. Modeling change requests due to faults in a large-scale telecommunication system. *Journal of Systems and Software* 2004; **72**(2):235–247.
85. Kenmei B, Antoniol G, Penta MD. Trend analysis and issue prediction in large-scale open source systems. *Proc. 12th European Conf. on Software Maintenance and Reengineering* (*CSMR'2008*), 2008; 73–82.
86. Kim S, Zhang H, Wu R, Gong L. Dealing with noise in defect prediction. *Proc. 33rd Int. Conf. on Software Engineering* (*ICSE'2011*). IEEE, 2011; 481–490.
87. Ostrand TJ, Weyuker EJ. A tool for mining defect-tracking systems to predict fault-prone files. *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2004*), vol. **4**, 2004; 85–89.
88. Radlinski L. Predicting defect types in software projects. *Polish Journal of Environmental Studies* 2009; **18**(3):311–315.
89. Song Q, Shepperd MJ, Cartwright M, Mair C. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering* 2006; **32**(2):69–82.
90. Wedel M, Jensen U, Gohner P. Mining software code repositories and bug databases using survival analysis models. *Proc. 2nd Int. Symp. on Empirical Software Engineering and Measurement* (*ESEM'2008*), 2008; 282–284.
91. Zhang C, Joshi H, Ramaswamy S, Bayrak C. A dynamic approach to software bug estimation. Advances in Computer and Information Sciences and Engineering, Sobh T (ed.). Springer: Netherlands, 2008; 108–113.
92. Abdelmoez W, Kholief M, Elsalmy F. Bug fix-time prediction model using naïve Bayes classifier. *Proc. 22nd Int. Conf. on Computer Theory and Applications* (*ICCTA 2012*), 2012; 167–172.
93. Anbalagan P, Vouk MA. On predicting the time taken to correct bug reports in open source projects. *Proc. IEEE Int. Conf. on Software Maintenance* (*ICSM'2009*), 2009; 523–526.
94. Bhattacharya P, Neamtiu I. Bug-fix time prediction models: Can we do better? *Proc. 8th Working Conf. on Mining Software Repositories* (*MSR'2011*). ACM, 2011; 207–210.
95. Bougie G, Treude C, German DM, Storey M-AD. A comparative exploration of FreeBSD bug lifetimes. *Proc. 7th IEEE Working Conf. on Mining Software Repositories* (*MSR'2010*), 2010; 106–109.
96. Guo PJ, Zimmermann T, Nagappan N, Murphy B. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. *Proc. 32nd ACM/IEEE Int. Conf. on Software Engineering* (*ICSE'2010*), vol. **1**, 2010; 495–504.
97. Hassouna A, Tahvildari L. An effort prediction framework for software defect correction. *Information and Software Technology* 2010; **52**(2):197–209.
98. Hewett R, Kijsanayothin P. On modeling software defect repair time. *Empirical Software Engineering* 2009; **14**(2):165–186.
99. Hooimeijer P, Weimer W. Modeling bug report quality. *Proc. 22nd IEEE/ACM Int. Conf. on Automated software engineering* (*ASE'2007*), 2007; 34–43.

DOI: 10.1002/smr

100. Kim S, Whitehead EJ Jr. How long did it take to fix bugs? *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2006*), 2006; 173–174.
101. Lamkanfi A, Demeyer S. Filtering bug reports for fix-time analysis. *Proc. 16th European Conf. on Software Maintenance and Reengineering* (*CSMR'2012*). IEEE, 2012; 379–384.
102. Panjer LD. Predicting eclipse bug lifetimes. *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2007*), 2007.
103. Raja U. All complaints are not created equal: text analysis of open source software defect reports. *Empirical Software Engineering* 2012; 1–22. 10.1007/s10664-012-9197-9
104. Raja U, Hale DP, Hale JE. Modeling software evolution defects: a time series approach. *Journal of Software Maintenance and Evolution: Research and Practice* 2009; **21**(1):49–71.
105. Weiss C, Premraj R, Zimmermann T, Zeller A. How long will it take to fix this bug? *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2007*), 2007.
106. Xuan J, Hu Y, Jiang H. Debt-prone bugs: technical debt in software maintenance. *International Journal of Advancements in Computing Technology* 2012a; **4**(19):453–461.
107. Zimmermann T, Nagappan N, Guo PJ, Murphy B. Characterizing and predicting which bugs get reopened. *Proc. 34th Int. Conf. on Software Engineering* (*ICSE'2012*, *SEIP Track*), 2012; 1074–1083.
108. Ahsan SN, Wotawa F. Impact analysis of scrs using single and multi-label machine learning classification. *Proc. 4th Int. Symp. on Empirical Software Engineering and Measurement* (*ESEM'2010*), 2010.
109. Antoniol G, Canfora G, Casazza G, Lucia AD. Identifying the starting impact set of a maintenance request: a case study. *Proc. Conf. on Software Maintenance and Reengineering* (*CSMR'2000*), 2000; 227–230.
110. Bangcharoensap P, Ihara A, Kamei Y, Matsumoto K-i. Locating source code to be fixed based on initial bug reports - a case study on the eclipse project. *Proc. 4th Int. Workshop on Empirical Software Engineering in Practice*. IEEE, 2012; 10–15.
111. Canfora G, Cerulo L. Impact analysis by mining software and change request repositories. *Proc. 11th IEEE Int. Software Metrics Symp*. (*METRICS'05*). IEEE, 2005.
112. Canfora G, Cerulo L. Fine grained indexing of software repositories to support impact analysis. *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2006*), 2006a; 105–111.
113. Chen I-X, Yang C-Z, Lu T-K, Jaygarl H. Implicit social network model for predicting and tracking the location of faults. *Proc. 2008 32nd Annu. IEEE Int. Computer Software and Applications Conf.* (*COMPSAC'2008*). IEEE, 2008; 136–143.
114. Gethers M, Dit B, Kagdi H, Poshyvanyk D. Integrated impact analysis for managing software changes. *Proc. of 34th IEEE/ACM Int. Conf. on Software Engineering* (*ICSE'12*), 2012; 2–9.
115. Kagdi H, Gethers M, Poshyvanyk D (2012b). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering* **18**(5): 933–969.
116. Nguyen AT, Nguyen TT, Nguyen HA, Nguyen TN. Multi-layered approach for recovering links between bug reports and fixes. *Proc. ACM SIGSOFT 20th Int. Symp. on the Foundations of Software Engineering* (*FSE'2012*). ACM, 2012.
117. Somasundaram K, Murphy GC. Automatic categorization of bug reports using latent dirichlet allocation. *Proc. 5th India Software Engineering Conf.* (*ISEC'2012*). ACM, 2012; 125–130.
118. Torchiano M, Ricca F. Impact analysis by means of unstructured knowledge in the context of bug repositories. *Proc. 4th Int. Symp. on Empirical Software Engineering and Measurement* (*ESEM'2010*), 2010.
119. Wang D, Zhang H, Liu R, Lin M, Wu W. Predicting bugs' components via mining bug reports. *Journal of Software* 2012; **7**(5):1149–1154.
120. Zhou J, Zhang H, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. *Proc. 34th Int. Conf. on Software Engineering* (*ICSE'2012*), 2012; 14–24.
121. Ko AJ, Myers BA, Chau DH. A linguistic analysis of how people describe software problems. *Proc. Visual Languages and Human-Centric Computing* (*VLHCC'2006*). IEEE, 2006; 127–134.
122. Laukkanen E, Mantyla M. Survey reproduction of defect reporting in industrial software development. *Proc. 2011 Int. Symp. on Empirical Software Engineering and Measurement* (*ESEM'2011*). IEEE, 2011; 197–206.
123. Edwards HK. System utilization and changes in implemented information systems: a case study. *IAENG International Journal of Computer Science* 2007; **33**(1): 12 (online publication).
124. Menzies T, Caglayan B, Kocaguneli E, Krall J, Peters F, Turhan B. The PROMISE repository of empirical software engineering data, 2012. Available from: http://promisedata.googlecode.com [Accessed on 11 May 2013].
125. Crowston K, Scozzi B. Coordination practices within floss development teams the bug fixing process. *Proc. 1st Int. Workshop on Computer Supported Activity Coordination* (*CSAC'2004*). INSTICC Press, 2004; 21–30.
126. Davies J, Zhang H, Nussbaum L, German DM. Perspectives on bugs in the debian bug tracking system. *Proc. 9th Working Conf. on Mining Software Repositories* (*MSR'2010*), 2010; 86–89.
127. Nie C, Zeng D, Zheng X, Wang F-Y, Zhao H. Modeling open source software bugs with complex networks. *Proc. IEEE Int. Conf. on Service Operations and Logistics and Informatics* (*SOLI'2010*). IEEE, 2010; 375–379.
128. Ohira M, Hassan A, Osawa N, Matsumoto K-i. The impact of bug management patterns on bug fixing: a case study of Eclipse projects. *Proc. of 28th IEEE Int. Conf. on Software Maintenance* (*ICSM'2012*), 2012.
129. Rosso CD. Comprehend and analyze knowledge networks to improve software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 2009; **21**(3):189–215.
130. Sandusky RJ, Gasser L. Negotiation and the coordination of information and activity in distributed software problem management. *Proc. 2005 Int. ACM SIGGROUP Conf. on Supporting Group Work* (*GROUP'2005*), 2005; 187–196.

DOI: 10.1002/smr

131. Sandusky RJS, Gasser L. Bug report networks: varieties, strategies, and impacts in a f/oss development community. *Proc. 1st Int. Workshop on Mining Software Repositories* (*MSR'2004*), 2004.
132. Valetto G, Helander ME, Ehrlich K, Chulani S, Wegman MN, Williams C. Using software repositories to investigate socio-technical congruence in development projects. *Proc. Int. Workshop on Mining Software Repositories* (*MSR'2007*), 2007.
133. Burch E, Kungs H-J. Modeling software maintenance requests: a case study. *Proc. 1997 Int. Conf. on Software Maintenance* (*ICSM'1997*), 1997; 40–47.
134. Gupta A, Slyngstad OPN, Conradi R, Mohagheghi P, Ronneberg H, Landre E. An empirical study of software changes in statoil asa - origin, priority level and relation to component size. *Proc. Int. Conf. on Software Engineering Advances* (*ICSEA'2006*). IEEE, 2006.
135. Elcock A, Laplante PA. Testing software without requirements: Using development artifacts to develop test cases. *Innovations in Systems and Software Engineering* 2006; **2**(4):137–145.
136. Moritz E. Case study: How analysis of customer found defects can be used by system test to improve quality. *Proc. of 31st Int. Conf. on Software Engineering* (*ICSE'2009*), 2009; 123–129.
137. Wang D, Wang Q, Yang Y, Li Q, Wang H, Yuan F. Is it really a defect? An empirical study on measuring and improving the process of software defect reporting. *Proc. Int. Symp. on Empirical Software Engineering and Measurement* (*ESEM'2011*). IEEE, 2011; 434–443.
138. Yu L, Kerong B. Knowledge representation of software faults based on open bug repository. *Proc. Int. Conf. on Computer Design and Applications* (*ICCDA'2010*), vol. **2**, 2010; 93–96.
139. Baeza-Yates RA, Ribeiro-Neto B. Modern information retrieval. Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1999.
140. Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P. Fair and balanced? Bias in bug-fix datasets. *Proc. ACM SIGSOFT Symp. on the Foundations of Software Engineering* (*FSE'2009*), 2009; 121–130.
141. Cavalcanti YC, Neto PADMS, Machado IDC, de Almeida ES, de Lemos Meira SR. Towards understanding software change request assignment: a survey with practitioners. *Proc. 17th Int. Conf. on Evaluation and Assessment in Software Engineering* (*EASE'2013*), 2013.
142. Caglayan B, Bener A. Issue ownership activity in two large software projects. *ACM SIGSOFT Software Engineering Notes* 2012; **37**(6):1.
143. Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. *Technical Report*, Universität des Saarlandes, Saarbrücken, Germany, 2012b.
144. Hewett R, Kulkarni A, Stringfellow C, Andrews AA. Software defect data and predictability for testing schedules. *Proc. 18th Int. Conf. on Software Engineering & Knowledge Engineering* (*SEKE'2006*), 2006; 499–504.
145. Knab P, Fluri B, Gall HC, Pinzger M. Interactive views for analyzing problem reports. *Proc. IEEE Int. Conf. on Software Maintenance* (*ICSM'2009*). IEEE, 2009.
146. Ayari K, Meshkinfam P, Antoniol G, Penta MD. Threats on building models from CVS and Bugzilla repositories: the Mozilla case study. *Proc. 2007 Conf. of the Center for Advanced Studies on Collaborative Research* (*CASCON'2007*). ACM, 2007; 215–228.
147. He Z, Shu F, Yang Y, Zhang W, Wang Q. Data unpredictability in software defect-fixing effort prediction. *Proc. 10th Int. Conf. on Quality Software* (*QSIC'2010*). IEEE, 2010; 220–226.
148. Schugerl P, Rilling J, Charland P. Mining bug repositories–a quality assessment. *Proc. 2008 Int. Conf. on Computational Intelligence for Modelling Control and Automation* (*CIMCA'2008*). IEEE, 2008b; 1105–1110.
149. Breu S, Premraj R, Sillito J, Zimmermann T. Information needs in bug reports: improving cooperation between developers and users. *Proc. 2010 ACM Conf. on Computer supported cooperative work* (*CSCW'2010*). ACM, 2010; 301–310.
150. Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C. What makes a good bug report? *IEEE Transactions on Software Engineering* 2010; **36**(5):618–643.
151. Linstead E, Baldi P. Mining the coherence of GNOME bug reports with statistical topic models. *Proc. 2009 6th IEEE Int. Working Conf. on Mining Software Repositories* (*MSR'2009*). IEEE, 2009; 99–102.
152. Schugerl P, Rilling J, Charland P. Enriching se ontologies with bug report quality. *Proc. 4th Int. Workshop on Semantic Web Enabled Software Engineering* (*SWESE'2008*), 2008a.
153. Rastkar S, Murphy GC, Murray G. Summarizing software artifacts: a case study of bug reports. *Proc. 32nd ACM/IEEE Int. Conf. on Software Engineering* (*ICSE'2010*). ACM, 2010; 505–514.
154. Bettenburg N, Premraj R, Zimmermann T, Kim S. Extracting structural information from bug reports. *Proc. 5th Working Conf. on Mining Software Repositories* (*MSR'2008*), 2008b; 27–30.
155. Bettenburg N, Just S, Schroter A, Weiss C, Premraj R, Zimmermann T. What makes a good bug report. *Proc. 16th Int. Symp. on Foundations of Software Engineering* (*FSE'2008*), 2008c; 308–318.
156. Revell M. Feeling the heat, 2010. Available from: http://blog.launchpad.net/bug-tracking/bug-heat [Accessed on 11 May 2013].
157. Sun J. Why are bug reports invalid? *Proc. IEEE 4th Int. Conf. on Software Testing, Verification and Validation* (*ICST'2011*). IEEE, 2011; 407–410.
158. Stojanov Z, Dobrilovic D, Jevtic V. Identifying properties of software change request process: Qualitative investigation in very small software companies. *Proc. IEEE 9th Int. Symp. on Intelligent Systems and Informatics* (*SISY'2011*). IEEE, 2011; 47–52.
159. Kagdi H, Collard M, Maletic J. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 2007; **19**(2):77–131.
160. Bertram D. The social nature of issue tracking in software engineering. *Master's thesis*, University of Calgary, Calgary, Alberta, 2009.

DOI: 10.1002/smr

AUTHORS' BIOGRAPHIES

Yguaratã Cerqueira Cavalcanti is a graduate of computer science from Federal University of Alagoas in 2007, received his MS degree in Computer Science from Federal University of Pernambuco in 2009, and is currently a PhD candidate in Computer Science at Federal University of Pernambuco. He is a system development analyst at the Brazilian Federal Organization for Data Processing (SERPRO). He is member of the Reuse in Software Engineering Research (RiSE) group, ACM, and Institute of Electrical and Electronics Engineers Computer Society. His primary interest is software engineering, with an emphasis on software maintenance and reuse.

Paulo Anselmo da Mota Silveira Neto has a Bachelor of Computer Science degree from Catholic University of Pernambuco (UNICAP), specialist in software engineering from University of Pernambuco (UPE), Master of Science degree in computer science (software engineering) from Federal University of Pernambuco (UFPE). Nowadays, he is a PhD candidate in computer science at Federal University of Pernambuco and member of the RiSE group, which has executed research regarding to Software Product Lines (SPL) Testing, SPL Architecture Evaluation, Test Selection Techniques, and Regression Testing. He is also participating on important research projects in software engineering area, as the National Institute of Science and Technology for Software Engineering (I.N.E.S.).

Ivan do Carmo Machado received his MS degree in computer science from Federal University of Pernambuco, Brazil, in 2010. He is currently a PhD candidate in computer science at Federal University of Bahia, Brazil, where he is also a member of the RiSE group. His primary research interest is software engineering, with an emphasis on software testing, variability management, software product line engineering, and empirical software engineering. He is a member of the Association for Computing Machinery and the Brazilian Computer Society.

Tassio Ferreira Vale received his MS degree in computer science from Federal University of Pernambuco, Brazil, in 2010. He is currently a PhD candidate in computer science at Federal University of Bahia, Brazil, where he is also a member of the RiSE group. His primary research interest is software engineering, with an emphasis on software testing, variability management, software product line engineering, and empirical software engineering. He is a member of the Association for Computing Machinery and the Brazilian Computer Society..

Eduardo Santana de Almeida is an assistant professor at Federal University of Bahia and head of the Reuse in Software Engineering Labs. He has more than 100 papers published in the main conferences and journals related to software engineering, and has chaired several national and international conferences and workshops. His research areas include methods, processes, tools, and metrics to develop reusable software. Contact him at esa@dcc.ufba.br.

DOI: 10.1002/smr

Silvio Romero de Lemos Meira has a degree in electronic engineering from the Instituto Tecnológico de Aeronáutica (1977), Masters in Computer Science from Universidade Federal de Pernambuco (1981), and PhD in computer science at University of Kent at Canterbury (1985). He is currently a professor at Universidade Federal de Pernambuco. He has experience in computer science, with emphasis on software engineering, working on the following topics: software reuse, information systems, open source, social networking, performance, and quality metrics in software engineering.

DOI: 10.1002/smr