

# A Systematic Framework for Evaluating Bug-assignment Research

Ali Sajedi-Badashian\*, Eleni Stroulia

*Department of Computing Science, University of Alberta, Canada*

---

## Abstract

Bug assignment is the task of ranking candidate developers in terms of their potential competence to fix a bug report. Numerous methods have been developed to address this task, relying on different methodological assumptions and demonstrating their effectiveness with a variety of empirical studies with numerous data sets and evaluation criteria. Despite the importance of the subject and the attention it has received from researchers, there is still no unanimity on how to validate and comparatively evaluate bug-assignment methods and, often times, methods reported in the literature are not reproducible.

In this paper, we first report on our systematic review of the broad bug-assignment research field. Next, we focus on a few key empirical studies and review their choices with respect to three important experimental-design parameters, namely the evaluation metric(s) they report, their definition of who the real assignee is, and the community of developers they consider as candidate assignees.

The substantial variability on these criteria led us to formulate a systematic experiment to explore the impact of these choices. We conducted our experiment on a comprehensive data set of bugs<sup>1</sup> we collected from 13 long-term open-source projects, using a simple Tf-IDf similarity metric. Based on our experiment, we argue that MAP is the most informative evaluation metric, the developer community should be defined as “all the project members”, and the real assignee should be defined as “any developer who worked toward fixing a bug”.

---

**Keywords:** Bug-assignment, bug report assignment, change request assignment, software engineering, research evaluation, reproducibility

## 1. Introduction

Bug-assignment (BA) is an important problem for the software-engineering industry. As a key task of software development as well as quality-assurance process, it aims at identifying the most appropriate developer(s) to fix a given bug. Typically, BA methods consider the developers’ previous bug assignments and other activities as indicators of their expertise and rank the developers’ relevance to the bug in question using a variety of heuristics.

Previous BA research involves a number of challenging but well-addressed questions, related to *how* to implement a more accurate BA method, including “how to gather evidence for a developer’s expertise in software projects?”, “how to relate different pieces of information to a bug report to assign it to a developer”, “how to utilize similarity measures to match a bug report with a developer?”, “how to use other clues or heuristics to connect a bug report to a candidate developer as the potential assignee”, “how to take into account the developers’ workload?” and so on. These types of questions are addressed in almost all the previous BA studies including the following studies.

The problem has already received substantial attention over the past 15 years (Jeong et al., 2009; Bhattacharya and Neamtii, 2010; Aljarah et al., 2011; Linares-Vázquez et al., 2012; Shokripour et al.,

---

\*Corresponding author

Email addresses: [alisajedi@ualberta.ca](mailto:alisajedi@ualberta.ca) (Ali Sajedi-Badashian), [stroulia@ualberta.ca](mailto:stroulia@ualberta.ca) (Eleni Stroulia)

<sup>1</sup>The data set, source code, documentations and detailed output results are available at:  
<https://github.com/TaskAssignment/MSBA-outline>

2013; Nguyen et al., 2014; Liu et al., 2016; Xia et al., 2015). Despite the vast attention of researchers, BA is still an expensive, time-consuming task in software projects (Saha et al., 2015). Between 50% to 90% of software development cost is regarding maintenance (Bhattacharya et al., 2012; Seacord et al., 2003), and a great deal of maintenance work is dealing with bugs. Large projects receive hundreds of bug reports daily (Tian et al., 2016), which get recorded in their issue-tracking tools. In Eclipse and Mozilla, it takes about 40 and 180 days respectively to assign a bug to a developer (Bhattacharya et al., 2012). In ArgoUML and PostgreSQL, the median time-to-fix for bugs in some projects is around 200 days (Kim and Whitehead Jr, 2006). In Eclipse, 24% of the bugs are re-assigned to another developer, before getting fixed (Baysal et al., 2009). Even the fixed bugs, in both Eclipse and Mozilla, have been re-assigned at least once in more than 90% of the cases (Jonsson et al., 2016; Bhattacharya et al., 2012).

This indicates the need for more accurate BA methods for big projects (Zhang et al., 2016c,a) to help automate the BA task in the issue-tracking tools. Currently, no issue-tracking tool automates this task, which is still manual or, at best, semi-automated. Despite the long record of related research in the field, it is hard to apply the proposed BA research in practice (Jie et al., 2015). Even it is a question whether the previously introduced BA approaches can be applied in large proprietary projects with acceptable performance or not (Jonsson et al., 2016). Still the industry needs more practical methods, with higher standards (Bhattacharya et al., 2012).

A study can be useful in industry if it undergoes through a realistic validation framework, and, is reproducible. The term “Reproducible research”, introduced by Jan Claerbout, tries to infuse standards for publications in computing science (Fomel and Claerbout, 2009). The idea is that the main product of a research is not only the paper, but also the full contents and materials that may be used to build upon the research and reproduce the results. These materials include but are not limited to source code and data sets used in computational science experiments (Fomel and Claerbout, 2009; Schwab et al., 2000). In a *Science* paper, Roger Peng mentioned the potential of serving as “*a minimum standard for judging scientific claims*” an important characteristic of reproducibility (Peng, 2011). He indicated the need for *data*, *meta-data* and *code* being linked to each other and to the corresponding publications as prerequisites for *full reproducible* research. Posing specific assumptions, conditions or dependencies in some of the previous researches hinders this.

Regarding BA research, one study may depend on very detailed data or sophisticated meta-data that rarely are accessible. Or it may pose biased conditions or filtering (e.g., to remain a few developers or small number of bug reports) in favor of itself. These make the publication of future high-quality studies harder since the comparison against those studies cannot be fair. To summarize, we found the following evaluation-related problems;

1. There is no generally agreed-upon definition of the “real assignee” (i.e., who the best developer for a given bug really is) against which to validate BA methods (i.e., the notion of golden truth). The broader the definition of real assignee is, the easier the prediction will be.
2. The rule of thumb for defining the group of developers who are considered as candidate assignees differs a lot from case to case. The bigger the community is considered, the more difficult the BA task becomes.

In addition, there is no unanimity in evaluation and reporting metrics which are used in assessment of the proposed methods. Needless to say, many of the previously used evaluation measures do not reflect the effectiveness of the proposed method properly. Compounded by the fact that many research publications do not share their data or code, and the experimental data sets vary substantially in their size and complexity, the above issues can become critical reproducibility problems. And if these problems are not addressed correctly in a BA study, the usefulness of the study can be questioned, or even disproved in further research.

In this work, we systematically examine the above-mentioned evaluation-related factors. We propose a practical framework to be able to fairly evaluate goodness of a method and compare it against as many methods reported in the literature as possible. Rather than discussing *how* to establish a new BA method, we cover two “*what questions*”. These questions are the main research questions of this paper:

Research Questions:

1. **What is the best definition of “real assignee”?**

What is the best “golden truth” (gold standard) for cross-validating against a developer recommendation? How can we judge if a recommended developer is a good fit for fixing a given bug or not? And what are the exact criteria for this judgment (to identify the real assignees for a given bug) in a project?

2. **What is the best definition for “developer community” from which the bug-assignment methods recommend appropriate developers?**

Which developers in the project should be considered as “developer community” (considered as the pool of candidate assignees), in order to have a fair evaluation? Are the members of this community normally affected by the definition of real assignee? Does the definition of this community affect the results (reported based on evaluation measures)? And does *filtering* this community bias the results?

Answering the above questions helps to establish a baseline for fair comparison of the results in further BA research. To answer to the above questions, we have curated an extensive data set, including bug reports from thirteen open-source projects, their meta-data and textual information and their assignee(s), according to the different definitions of assignee discussed in this paper. We extracted the information of these projects using Github APIs. This data set is used for an experiment in support of the arguments we provide. We also publish this data set online for further research.

The **contributions** of this paper are as follows: First, in a systematic review, we summarize key methods in the literature and discuss them from different aspects. Then, we investigate the current BA evaluation measures, and provide arguments towards selecting the best evaluation measure for BA research. After that, we identify two important dimensions of variability in the evaluation of BA methods and put forward a framework that argues for special choices in these dimensions. We motivate our framework by demonstrating how these aspects affect the reported results with a study, a big data set and a standard similarity metric, *tf-idf*.

The remainder of this paper is organized as follows. We perform the systematic review in Sections 2 and 3. Section 4 discusses the evaluation metrics used for evaluating BA research. Section 5 introduces the common definitions of the two dimensions or variability, “real assignee” and “developer community”. After Section 6 which describes the experiment setup, in Sections 7, we discuss the two research questions of the study regarding those two dimensions of variability. Section 8 reflects on some implications and discussions about the provided framework. Finally, Section 9 concludes the paper.

## 2. A survey on Bug-assignment Research

In the context of our own research in the area of BA, we experienced lack of a comprehensive review on the field investigating on different aspects of the problem. In this section, we discuss such a review. We first explain our survey methodology and discuss the main objectives, methods, metrics and information used for bug-developer matching in previous research.

### 2.1. Survey protocols and process

This section presents the process we followed for doing the survey. We conducted the systematic review according to the guidelines mentioned in (Kitchenham and Charters, 2007; Budgen and Brereton, 2006; Weidt and Silva, 2016).

The first goal is to select all the papers that propose a new BA method, for further studies to get a proper intuition about the area and proposed solutions. We study the BA objectives, formulations and methods. Later, we will focus on a subset of these research and discuss their methods more in-depth.

We performed a publication search on four research databases; ACM<sup>1</sup>, IEEE<sup>2</sup>, Sciencedirect<sup>3</sup> and Springer<sup>4</sup>. We searched for *bug assignment*, “*bug assignment*”, *bug triaging* and “*bug triaging*” in each of the databases. For each search, we captured the top 100 returned results (which were usually returned in the first 4 or 5 pages). We reviewed all the 100 results of each search and considered them in our survey if they were introducing a BA approach. The selection criteria were as follows:

*Inclusion criteria:*.

1. Papers must be published in peer-reviewed conferences or journals;
2. Papers must describe a new BA-related methodology; and
3. Papers must follow the formulation of “ranking developers for the given bugs”, or similar formulations.

*Exclusion criteria:*.

1. Papers that are only tool-development or poster papers are excluded;
2. Papers that are about bug-triaging rather than BA are removed; and
3. Papers that investigate “challenges of bug-assignment and reassignment”, rather than BA are excluded.

Considering the above criteria, we reviewed all the mentioned results. First, we analyzed the title and abstract. If we found the paper related to the topic, then, we surveyed the paper in more detail to make sure it fits with inclusion and exclusion criteria. We did not consider any filtering on the venue (conference or journal) since all the selected databases support legitimate scientific conferences and journals.

We eliminated studies targeting general purpose task-assignment (Helming et al., 2010; Imtiaz and Ikram, 2017), applying BA methods in code reviewer recommendation (Yu et al., 2016), bug fix time prediction (Zhang et al., 2013; Akbarinasaji et al., 2017), tool-development for BA with no reported accuracy (Bortis and Hoek, 2013) and bug-triaging related studies other than BA, including bug report de-duplication (Wang et al., 2008; Banerjee et al., 2016), bug report classification Zhou et al. (2016), bug localization (Chaparro, 2017; Wang and Lo, 2016; Sisman et al., 2017), component-assignment (recommending a component for a given bug report) (Somasundaram and Murphy, 2012; Yan et al., 2016) and investigating problems of assignment and re-assignment of bugs (Baysal et al., 2012; Cavalcanti et al., 2014b).

Finally, we found 74 BA-related studies, dating from 2004 to the end of 2017, some of which were extensions or journal versions of the other ones. The 74 extracted papers targeted the problem using different wordings; bug-triaging, change request assignment, anomaly request assignment, issue-assignment, and more prevalently bug-assignment (BA). We summarized all these papers and extracted the methods each study utilizes. The details of the techniques used in those papers are explained in Appendix A (Table 6).

## 2.2. Bug-assignment Tasks and Objectives

The most prevalent formulation of BA is as follows: “Given a new bug report, identify a ranked list of developers, whose expertise (based on their record of contributions to the project) qualifies them to fix the bug” (Matter et al., 2009; Bhattacharya and Neamtiu, 2010; Hu et al., 2014; Shokripour et al., 2015; Khatun and Sakib, 2016); this is the formulation most researchers adopted and used in their studies. Also, in this paper, we investigate our research questions considering this formulation. However, there are other formulations that a small portion of research in BA targets and we briefly mention them here.

---

<sup>1</sup><https://dl.acm.org/dl.cfm>

<sup>2</sup><http://ieeexplore.ieee.org/Xplore/home.jsp>

<sup>3</sup><https://www.sciencedirect.com/>

<sup>4</sup><https://link.springer.com/>

Jonsson et al. (Jonsson et al., 2012; Jonsson, 2013) introduced the problem of “team bug-assignment” in which bugs are redirected to one of several available teams. This formulation was motivated by the specific process of their industrial partners, which is not the case in most software projects. In this formulation, teams are typically stable over time, which effectively makes the concept of “team” very similar to that of a “developer” in other research. In fact, by assigning bugs to the teams their method makes the BA task easier. In other words, the goal of assigning bugs to teams is to decrease the number of candidate assignees, which may be quite large in a large open-source project. Given that different teams are likely to work on distinct sets of modules over time, their expertise is likely non-overlapping, again simplifying the overall BA task.

Instead of maximizing the expertise of the candidate assignee, some methods try to minimize the time-to-fix the bugs (Park et al., 2011; Nguyen et al., 2014; Park et al., 2016). For example, Nguyen et al. (Nguyen et al., 2014) propose the construction of a topic model from previously assigned bugs. For an incoming bug, their method predicts the resolution time for each developer (by calculating log normal distribution of combination of three factors; fixer, topic and severity) and ranks the candidate assignees for the new bug report based on this time estimate.

More recently, Liu et al. described a method that takes into account both objectives, i.e., expertise maximization and time-to-fix minimization (Liu et al., 2016).

Finally, Karim et al. (Karim et al., 2016) proposed a more pragmatic multi-objective problem formulation, assigning multiple bugs to several developers at the same time. Their method assumes competency of developers in different areas (packages) as a function of number of times each developer changed a file in each area in previous attempts for fixing bugs. Using an Estimation By Analogy (EBA) method to obtain an effort estimation for a new bug report, they devised a function of the average time of previous similar bugs as the needed effort for the new bug report. Having estimations of the competency of developers in different areas and the needed effort in each area for the new bug report, they considered two Eclipse sub-projects and developed objective functions to estimate the completion time of “fixing a bug by a developer”. Their method aims at maximizing expertise, while also minimizing the total developers’ cost and total bug-fixing time, using a genetic algorithm. Similarly, Rahman et al. (Rahmana et al., 2012), Hosseini et al. (Hosseini et al., 2012) and Khalil et al. (Khalil et al., 2017) proposed multi-objective approaches for BA.

### 2.3. Bug-Assignment Methodologies

Previous research in BA approached the problem in different manners (details are shown in Table 6 in Appendix A). Older approaches used machine learning techniques that just tried to predict the next assignees by investigating the “relations” between the developers and the previously fixed bug reports and their technical terms. The performance of these approaches depends on the number of previously fixed bugs and they may perform poorly in small projects (Shokripour et al., 2012).

Then other methods like fuzzy (Tamrawi et al., 2011a) and statistical (Aljarah et al., 2011) approaches appeared. These methods utilized an expertise perspective for the developers regarding keywords. Based on which developer fixed which bug, they gathered an expertise profile for the developers to make a decision upon arrival of new bug reports.

Some researchers used deeper file and meta-data information to relate developers to the new bugs. Examples are location-based techniques (Hossen et al., 2014; Kagdi and Poshyvanyk, 2009; Shokripour et al., 2013). They first predict or find the location of bugs (i.e., methods or classes). Then, based on the available relations between developers and those locations, they predict the best developer who can work on those objects again. Location-based methods usually require bulky Version Control System (VCS) information (e.g., all the changed files in all project branches and commits). That can be the reason why in most methods that use this technique, the authors only run their experiment on a small number of bugs (e.g., a few bug reports up to a few hundreds). Also, social network (Park et al., 2011; Xuan et al., 2012) and tossing graph (Jeong et al., 2009; Bhattacharya and Neamtiu, 2010) approaches appeared that target more complicated interactions between development objects (e.g., developers, bugs, commits or a combination of them).

The most prevalent methods were Information Retrieval (IR), and then Machine Learning (ML). Many of recent studies focused on IR based activity profiling since it usually leads to higher accuracies

(Shokripour et al., 2013; Anjali et al., 2016). In the recent years, most of the studies use at least one IR method.

In recent years, some of the studies combined different methods (e.g., combining machine learning and tossing graphs (Bhattacharya and Neamtiu, 2010) or combining KNN and IR methods (Zanjani et al., 2015; Zhang et al., 2016a)). Most recently, the researchers show a tendency toward *social* point of view. For example, (Hu et al., 2014), (Zhang et al., 2016c) and (Zhang et al., 2016a) build a social network of developers to model their relationship with each other or with bugs or even source code components. Also, in our previous research, we proposed a model to recommend developers in Github based on their Stack Overflow contributions and the votes casted by the community (Sajedi et al., 2016).

### 3. Select Empirical Bug-assignment Studies

The selected list of 74 papers are all introducing a new methodology for BA. In the next step, we want to focus on a subset of these studies in more detail –based on dimensions of variability and reproducibility criteria. The goal is to enable further researchers compare their results against some reproducible research, as exemplary BA studies. We also elaborate more on the methods, evaluation metrics, knowledge assumptions and design choices of those exemplary studies. To select the new subset of BA studies, we apply the following criteria on those 74 papers:

#### *Inclusion criteria.*

1. Papers must follow the prevalent formulation of “ranking the developers for the bugs based on appropriateness of a single developer for each bug”;
2. Papers must be supported by BA experimental evaluation;
3. Papers must report the experiment results based on major evaluation metrics;
4. Papers should report final results on BA effectiveness (e.g., instead of a comparison of tuning values or data sets);
5. Papers must experiment on full data of developers and bugs (i.e., have no major data filtering); and
6. Experiments must contain relatively high number of developers ( $\sim 20$ ) and bugs ( $\sim 500$ ) in at least one project, to make the experiment more realistic.

#### *Exclusion criteria.*

1. Papers which their technique relies on external sources (e.g., Stack Overflow) rather than issue-tracking information are excluded; and
2. Papers that are proposing techniques in other domains with applications in BA are excluded.

Examples of the removed studies are particular BA studies with focus on specific areas like multi-objective BA studies (Karim et al., 2016; Rahmana et al., 2012; Khalil et al., 2017), team BA (Jonsson et al., 2012; Jonsson, 2013), market-based bug allocation (Hosseini et al., 2012), component-level BA (Wang et al., 2014), time/cost enhancement in BA (Park et al., 2011; Nguyen et al., 2014; Park et al., 2016) and bug report enrichment (Zhang et al., 2017a) with application in BA. Also we eliminated our previous paper (Sajedi et al., 2016) and other studies that work only on a narrow subset of users (who are shared between version control system and a question answering system like Stack Overflow) (Zhang et al., 2017b; Sahu et al., 2016).

After considering the above criteria precisely, we obtained **13 papers**<sup>5</sup>. These are great BA examples regarding evaluation, reproducibility and further comparisons. We will focus on these studies in the rest

---

<sup>5</sup>Three studies ((Bhattacharya and Neamtiu, 2010), (Tamrawi et al., 2011a) and (Cavalcanti et al., 2014a)) are conference version of other studies by the same authors. So, we merged them into their journal version in the tables of this paper.

of our paper as “selected BA studies”. We summarize the techniques, knowledge assumptions and used metrics in the selected BA studies in more detail.

### 3.1. Review of the Selected Research

Čubranić and Murphy (Čubranić and Murphy, 2004) developed a method that uses a **Naïve Bayes** classifier to assign each bug report (a “text document” consisting of the bug summary and description) to a developer (seen as a topic category or the “class”) who actually fixed the bug. When a new bug report arrives, it uses the textual fields of the bug to predict the related class (e.g., developer). To the best of our knowledge, this was the first time a method was proposed to automate the bug-assignment process.

Canfora and Cerulo Canfora and Cerulo (2006) used an **Information Retrieval (IR)** approach for BA. Their method assumes that the developers who have solved similar bug reports in the past are the best candidates to solve the new one. So, it considers each developer as a document by aggregating the textual descriptions of the previous change requests that the developer has addressed. Given a new bug report, it uses a probabilistic IR model and considers its textual description as a query to retrieve a candidate from the document (developer) repository.

Jeong et al. Jeong et al. (2009) developed a method that captures tossing probabilities between developers from tossing history of the bugs. Then, it makes a **tossing graph** of developers based on Markov Model. In this graph, the nodes are developers and the weight of the directed edges show the probability of tossing from one developer to the other. Finally, for predicting the assignees of a bug, it first produces a list of developers using a machine learning method. Then, after each developer in this list, it adds the neighbor developer with the most probable tossing weight from the graph.

Matter et al. used **Vector Space Model (VSM)** to consider the source code contributions and the previous bug fixing as evidence of expertise and build a vocabulary of “technical terms”. Their model builds this vocabulary from the technical terms the developer used in the source code (captured by diff of the submitted revision) or commit messages. Also, it adds to this vocabulary the technical terms mentioned in the previous bug reports assigned to the developer. As a result, the developer’s expertise is modeled and captured as a term vector. Given a new bug report, it calculates the **cosine distance** between the new bug report’s term vector and the developers’ and sorts the developers based on this score and reports the top ones.

Tamrawi et al. (Tamrawi et al., 2011a,b) introduced a **fuzzy** approach toward bug assignment which computes a score for each “developer - technical term” based on the technical terms available in previous bug reports and their fixing history by the developers. Considering a new bug report, it calculates a score for each developer as a candidate assignee by combining his/her scores for all the technical terms associated with the bug report in question. Then it sorts the developers based on this score. The newer version of this fuzzy method (Tamrawi et al., 2011b) also applies a term selection method to reduce noise data and speed up the algorithm. It extracts the top k terms that are most related with each developer. Then, when calculating the fuzzy score for each developer, it just considers those selected terms and ignores other terms.

Bhattacharya et al. (Bhattacharya and Neamtiu, 2010; Bhattacharya et al., 2012) developed a **tossing-graph** based method, similar to TG1 Jeong et al. (2009), that also adds labels the edges to improve the graph. The label of each edge indicates product, component and latest activity date. Then, it recommends three developers based on a machine learning method. Also, the tosee ranking uses the graph labels (tossing probability, product, component and last activity date of the developer) for each of the top two developers in this list to recommend a substitute (tosee) and enhance this list to a top-5 recommendation.

Shokripour et al. (Shokripour et al., 2012) combined **bug report localization, Information Extraction (IE)** and **Natural Language Processing (NLP)** to predict the assignees for the bug reports. Their method first applies IE and NLP techniques on file-related components (e.g., commit messages and their comments, plus their related source code elements including phrases in methods and classes) and also the bug reports to elicit their important phrases. When a new bug is reported, it first estimates the location of the new bug (i.e., the files that should be changed to fix the bug) by comparing the important phrases in the new bug report and other file-related components as mentioned above. After

selecting the “possibly related” files it recommends the developers with the most activity regarding those files according the historical data.

Zhang et al. (Zhang et al., 2016c) proposed a method that builds a **heterogeneous social network** of developers, bugs and their comments, components and products. Then, it selects the top  $k$  similar bug reports to the given new bug report by using **KNN** classification, **Cosine similarity** and **tf-idf**. After that, it extracts the list of commenters of those  $k$  bug reports as the narrowed list of “candidate developers” and obtains the score of each developer by calculating overall **heterogeneous proximity** of each developer with all other “candidate developers” on component and product of the new bug report, using the previously built network. Then it sorts the developers based on this score and reports them.

Cavalcanti et al. (Cavalcanti et al., 2014a, 2016) introduced a semi-automatic approach that combines **rule-based expert system (RBES)** with **information retrieval (IR)** methods. This method first summarizes the previous bug reports and extracts some simple rules based on their meta-data (e.g., component of the bugs, or being critical) or keywords in the bug report, and their real assignee. These rules can decide on some circumstances which developer should be assigned to a new bug. In the second phase, if the simple rules cannot recommend any developer for the new bug report, it uses the machine learning SVM classifier to consider previous assignments to developers and assign a label (developer) to it (e.g., the developers who fixed similar bug reports are most likely to fix the new one).

Finally, Sun et al. (Sun et al., 2017) proposed a method that extracts and analyzes the commits that are possibly related to the new change request. This is done by obtaining **cosine similarity** between new bug report and historical commits. Those commits are considered relevant commits and the changed source code is considered relevant source code. Authors of those commits are considered as candidate developers to fix the new issue. Finally, it uses **Collaborative Topic Modeling (CTM)** to give a score to each of those developers (based on shared keywords in their relevant source code and the new issue) and sort and recommend them to fix the new issue.

### 3.2. Knowledge Assumptions of Bug-assignment Methods

There are a wide range of information that have been used in various BA research. This information is used in some way to relate a developer to a bug. Table 1 shows a summary of the information used by the selected BA research.

According to this table, older approaches like (Čubranić and Murphy, 2004), (Canfora and Cerulo, 2006) and (Tamrawi et al., 2011b) mostly rely on textual information (e.g., title and description) of the

Table 1: A review of the used information for bug assignment in selected BA studies

Method	Bugs’ info		Developers’ expertise info				
	Title + description	Meta	Bug fixing; title / description	Being a committer	Tossing history	Meta	Changed code
(Čubranić and Murphy, 2004)	✓		✓				
(Canfora and Cerulo, 2006)	✓		✓				
(Jeong et al., 2009)	✓		✓		✓		
(Matter et al., 2009)	✓		✓	✓			✓
(Tamrawi et al., 2011a) (Tamrawi et al., 2011b)	✓		✓				
(Bhattacharya and Neamtiu, 2010) (Bhattacharya et al., 2012)	✓	✓	✓		✓	✓	
(Shokripour et al., 2012)	✓		✓	✓			
(Zhang et al., 2016c)	✓	✓	✓	✓		✓	
(Cavalcanti et al., 2014a) (Cavalcanti et al., 2016)	✓	✓	✓			✓	
(Sun et al., 2017)	✓	✓		✓		✓	✓



bug reports as clues for indication of expertise of developers and matching with new bug reports. Many of the newer research like (Bhattacharya et al., 2012), (Zhang et al., 2016c) and (Cavalcanti et al., 2016) used variety of meta-data fields (e.g., component, product, severity and operating system) to address some of the limitations.

As mentioned earlier, in recent years, many researchers tried to combine different methods to achieve more accurate results. Hence their combined methods need different types of information –as their various methods demand. For example, (Sun et al., 2017) considers commit messages and some meta data. In addition, it extracts and deals with source code and file changes. This sometimes makes a huge demand on the needed data and processing.

Extra information of the meta-data can help obtaining higher accuracies in some cases, but still text-based methods are the most effective techniques used (Shokripour et al., 2015; Sun et al., 2014) and those meta-data based approaches are sometimes difficult to setup; for example, some need to know who maintained different pieces of code in the IDE (Hossen et al., 2014), or who interacted online with whom in different setups (Zhang et al., 2016c). As a result, the textual elements remain the most prevalent and effective information used for BA.

### 3.3. Metrics for Evaluation of Bug-assignment Research

We reviewed the 13 selected papers to identify the metrics they used for evaluation, and, cross-validation of their approaches against real bug data. The metrics they used are *top-k* accuracy, *precision @k*, *recall @k* (all with  $k=1, 5$  and  $10$ ), *MRR* and *MAP*. These metrics (except MAP) were the most frequently used metrics for reporting the results in the whole 74 papers as well. MAP, however, was used only in (Zhang et al., 2016c). The reported results of the selected papers are shown in Table 2. We also reported two important design choices of these studies in the Table; number of developers and number of bugs they experimented on. Those choices can affect some of the evaluation metrics and are needed to mention in BA experiments.

This Table is useful for further comparisons of efficiency of new BA methods against the previous research for two reasons. First, it gives the researchers flexibility (in terms of choosing metrics) in comparing their results against some other research. Then, since these studies have met all the inclusion criteria we set for the reproducible research (e.g., the projects and the number of bugs they tested on are big enough and they did not do major filtering on developers or bug reports), the comparison of the results of a new research against these research would give sufficient feedback.

In the next section, we discuss the suitability of these metrics for reporting efficiency of BA research experiments.

## 4. A Discussion of Bug-assignment Evaluation Metrics

The evaluation metric can affect the outcome of the used method. According to our survey on the 74 papers, the metrics used in the literature are as follows:

**Top-k accuracy** is a metric that considers the number of real assignees available in top  $k$  recommended ranks. According to the summarized studies,  $k$  is usually considered 1, 5 or 10.

**Precision @k** is the percentage of suggested developers in the first  $k$  ranks for a bug who are actually real assignees of that bug ( $k$  is usually 1, 5 or 10).

**Recall @k** is the percentage of all the real assignees of a bug who are actually suggested in the top  $k$  recommended developers (again,  $k$  is usually 1, 5 or 10).

**F-measure** is another set-based measure that is equal to harmonic mean of precision and recall. In fact, it combines precision and recall into one metric (Manning et al., 2008; Xu et al., 2010).

**Mean Reciprocal Rank (MRR)** of the real assignee is equal to the mean of “reciprocal rank of the highest-ranked assignee of a bug” over all the bugs; it indicates “how far down the list of recommended developers for a bug one should proceed to find a real assignee”?

**Mean Average Precision (MAP)** is the mean of “the average of precision values at all ranks where a real assignee for a bug exists”, over all the bug reports. Generally, to calculate MAP over all bug reports, first, we need to calculate Average Precision (AP) for each bug: we calculate AP at several points at which there is a correct recommendation. Then we calculate the mean of AP over all the bugs.

Table 2: The evaluation metrics and other design choices of the selected research; We mention results up to hundredths which is two decimal digits (except the cases that in the cited research reported less than two decimals). “~” means the exact value was unclear (e.g., estimated from a figure). “-” or *blank cell* means the value is not mentioned nor depicted in the respective research. “?” shows a missing but important value (the number was expected but it is not clearly mentioned).

Method / Project	#devs	#bugs	Top1	Top5	Top10	P@1 / r@1	p@5 / r@5	p@10 / r@10	MRR	MAP
(Ćubranić and Murphy, 2004)										
Eclipse	162	15,859	30.00							
(Canfora and Cerulo, 2006)										
Mozilla	637	12,447				- / 12.0	- / 21.0	- / 24.0		
KDE	373	14,396				- / 5.0	- / 10.0	- / 12.0		
(Jeong et al., 2009)										
Eclipse	?	46,426		77.14						
Mozilla	?	84,559		70.82						
(Matter et al., 2009)										
Eclipse	210	130,769				33.6 / ~27	~16 / ~59	~10 / 71.0		
(Tamrawi et al., 2011a)(Tamrawi et al., 2011b)										
Firefox	3,014	188,139	32.1	73.9						
Eclipse	2,144	177,637	42.6	80.1						
Apache	1,695	43,162	39.8	75.0						
Net Beans	380	23,522	31.8	60.4						
FreeDesktop	374	17,084	51.2	81.1						
GCC	293	19,430	48.6	79.2						
Jazz	156	34,220	31.3	75.3						
(Bhattacharya and Neamtiu, 2010)(Bhattacharya et al., 2012)										
Mozilla	?	549,962	27.67	77.87						
Eclipse	?	306,297	32.36	77.43						
(Shokripour et al., 2012)										
Eclipse	?	?				- / ~32	- / ~71			
Mozilla	?	?				- / ~27	- / ~48			
Gnome	?	?				- / ~10	- / ~45			
(Zhang et al., 2016c)										
Mozilla	~874	74,100							0.28	44.49
Eclipse	~544	42,560							0.28	56.42
Ant	~203	763							0.35	36.48
TomCat6	~79	489							0.35	36.54
(Cavalcanti et al., 2014a)(Cavalcanti et al., 2016)										
New SIAFI - A	70	781	31.40							
New SIAFI - B	70	1031	22.00							
(Sun et al., 2017)										
JEdit	123	?	28.0	60.1	79.8					
Hadoop	82	?	8.5	30.1	50.3					
JDT-Debug	47	?	14.4	46.6	66.4					
Elastic	661	?	13.6	43.6	75.2					
Libgdx	345	?	22.0	51.3	69.6					

The question then becomes “what is the best metric to evaluate BA effectiveness”? We propose four criteria based on which we should select the most meaningful metric:

1. **The interpretation of the evaluation metric should be independent of other evaluation metrics:**

This supports easier comparison of a study against other approaches. Unlike precision and recall, that should be reported and interpreted together, MAP, MRR, *f-measure* and *top-k accuracy* are single-figure metrics.

2. **The rank of all the real assignees should be taken into account:**

The number of real assignees for a bug can be more than one. This varies for different bugs. *Top-k accuracy*, *precision @k*, *recall @k* and *f-measure* are all bound to a chosen threshold, *k*. In other words, they only count the real assignees recommended in the top *k* ranks and ignore the others. MRR only considers the rank of the first assignee, but MAP considers all of them.

3. **Errors in the higher ranks are worse than errors in the lower ranks:**

This is because the triager usually checks the higher ranks in the list and may not proceed to the last one. Again, in set-based measures like *Top-k accuracy*, *precision @k*, *recall @k* and *f-measure*, it does not matter where in the first *k* ranks the real assignee is. MAP and MRR, however, are affected by a hit in the first ranks much more than the next ranks. In a sense, they penalize the mistakes in the first ranks more than the next ranks. Note that while MRR stops at the first real assignee, this penalization continues for MAP, for every incorrect guess until the last real assignee in the list.

4. **The evaluation measure should be robust to the number of real assignees:**

Assume that there are 10 real assignees (correct answers) per bug. The chance of having at least one of them in the *top-10 accuracy*, would be very high, as compared to the case when there is only one assignee per bug. Between all the mentioned metrics, MAP is affected the least from the average number of real assignees in the project since it does not stop in the first *k* ranks and penalizes the incorrect guesses until the last assignee.

With the mentioned qualities, **MAP** is the best metric to report BA results with. It provides a single-figure quality measure, representing both precision and recall, with good discrimination and stability properties (Shi et al., 2012). It is a good performance metric when a short list of items is provided to the user (i.e., the triager) (Shi et al., 2012; Shani and Gunawardana, 2011). MAP is a single *effectiveness* metric that measures how *all* the relevant documents are ranked highly (close to top of the list). It satisfies all the four above-mentioned criteria. The other metrics cannot reflect the goodness of a recommended ranked list of developers regarding at least one of the above aspects. Note that when experimenting on several projects, in addition to reporting MAP per project, reporting a final MAP over all the bug reports of all the projects makes the comparison against other methods more straightforward.

MAP is widely used in evaluating methods in IR problems like document retrieval (Manning et al., 2008) and other software engineering problems like bug-report de-duplication (Alipour, 2013; Aggarwal et al., 2017). Interestingly, it is rarely used for BA evaluation; the only study out of all the 74 papers of our survey that reported MAP was (Zhang et al., 2016c). As a result, although reporting MAP would be convincing enough, for comparison with previous approaches (e.g., meta analysis), it is still recommended to report the mostly used metrics as well as MAP (*top-k accuracy*, *Precision @k*, *recall @k* and *MRR*). For this reason, we extracted any of these metrics reported in the 13 selected papers, which can be used for comparison in further research (see Table 2).

## 5. Dimensions of Variability in Bug-assignment Empirical Studies

In addition to the metrics, there are two dimensions that can affect the measurement and evaluation of the results. We found that the choices of “real assignee” and “developer community” are very important in this regard and they vary a lot in previous work.

### 5.1. “Real assignee”

Current version control systems maintain valuable information regarding developers and bugs. For example, Github, as the most popular version control system, maintains a page for each bug which shows the work around that bug. It shows interactions (e.g., comments, being assigned or closed) about the bug and meta-data elements (like reporting date, reporter and labels) in addition to the bug title and description. In case the bug is referenced from a commit or other actions, then an entry will be shown in the list of interactions, with links to those actions. These types of information are used for extracting developers’ expertise and the real assignees (gold set) for cross-validation purposes.

In almost all the previous research which proposed a new BA approach, before recommending the assignees, the authors used some rule of thumb to identify at least one developer for each bug, as the real/actual assignee. Then they compared their recommendations against these real assignees to measure the effectiveness of their approach and report it using some metrics. The definition of “real assignee” (golden truth) is critical in validating and understanding the goodness of a method, and, comparatively analyzing the merits and shortcomings of alternative methods.

Consider the following example as the effect of this golden truth on the evaluation results. Anvik et al. (Anvik et al., 2006) adopted some heuristics to assume developers as assignees of a bug. Based on those heuristics, they obtained (on average) 30, 10 and 12 assignees per bug report in their three projects, Firefox, Eclipse and gcc respectively. This is the highest among all the studies we observed in our survey. In Firefox, they obtained 64% precision @1, which is again the highest among all the previous studies of all the time (even till now, after more than 10 years) regarding precision @1 metric. One reason to this high precision was the high number of real assignees per bug. Considering their results on their two other projects supports this claim; in Eclipse and gcc, their precision @1 decreased to 40% and 6% respectively. Based on our observations in almost all the previous studies, each bug on average has only one, or up to a few real assignees. One reason that this number varies from one research to another, is that they adopted different “definitions of *real assignee*”. So, the definition of *real assignee* adopted by a research experiment can highly affect its results. Without a clear definition of *real assignee*, fair judgment or reproduction of a research would be problematic.

In our survey on BA research, we found that there are various definitions of real assignee in different studies, and that there is no unanimity in using/addressing this definition. In other words, the definition(s) of “real assignee” adopted by different studies is not consistent over those studies. This can make the process of evaluating the effectiveness of a method more subjective and obstruct reproducibility of the research. We will show later, that the adopted definition of real assignee can affect or even bias the reported results.

We surveyed BA studies, and, based on interactions between the developers and development objects (e.g., bugs or commits) extracted the following definitions of *real assignee* (examples of the research studies that used each of these definitions are shown in Table 3):

- *Type 1 (T1); AUTHOR: The author of a commit referencing a bug number as resolved.* In this case, the author, who is the original developer of the code includes in the commit message a mention that the newly contributed code fixes a specific bug in the project. In the Github page for the bug report, this reference is shown as an event of type *commit* in the bug’s life cycle.
- *Type 2 (T2); COAUTHOR: A developer (other than the original author of the committed code), who actually commits the code and references a bug as resolved.* In this case, the *coauthor* is different than the *author*. Typically, the *coauthor* has some higher-level permissions. In some cases, it indicates an additional code reviewer role examining and confirming that a piece of code fixes a bug. For example, the project maintainer who merges the patch or last applies it (approves it), the one who accepts a pull request, or the one who does the rebase is called *coauthor*. In all these examples, this person is different from the one who actually writes the code (i.e., the *author*).

Note that this type of assignee has not been explicitly studied before. Previous research either ignored T2 assignees or considered them under T1. The fact is, however, that this is a different indication of bug-fixing contribution. Therefore, we believe that it is worth examining it separately.

Again, like T1, in the Github page for the bug report, this type of reference / fix the bug is shown as an event.

- *Type 3 (T3); ADMIN\_CLOSER: The developer who closes the bug.* If a developer decides to close a bug, one can assume that they know enough about that bug and may be competent to fix it. In some projects, any developer can close a bug –or re-open it later; however, in most big projects, this privilege is reserved for higher-level or administrative roles that only the *core* developers have. In some cases, these developers review the code and the proposed bug as soon as it is reported, and then bring the bug to the attention of appropriate programmers in the team. Note that a bug may be re-opened, worked on by a few developers, and closed again several times. In these cases, any developer who closes the bug is considered as a *T3 assignee*.
- *Type 4 (T4); DRAFTED\_ASSIGNEE: The developer tagged as “assignee” when the bug is closed.* At each point in time, several people can be assigned/unassigned/reassigned to a bug, either by their own initiative or by other project members. The developer who is tagged *assignee* (and remains the “tagged assignee”) until the time when the bug is closed is assumed to be the T4 assignee of the bug. Github shows this developer in the “Assignees” section of each bug. Note that just being tagged as an “assignee” is more like a temporary assignee and should not be considered as evidence of relevant expertise. For example, if a developer is tagged as assignee of a bug, but finds that cannot fix it, then they may opt out or may be un-assigned. As another example, a developer may be the tagged assignee while the bug remains open forever; we do not consider these cases *useful* and simply ignore them.
- *Type 5 (T5); ALL\_TYPES: The union of all the above four types including all sorts of work toward fixing the bug.* This definition is useful in that it leads to a broad and realistic formulation of the BA problem. It includes code authorship and co-authorship, administrative bug manipulation and being drafted as assignee.

Table 3: The types of assignment used in selected previous research, varied from T1 to T4.

Method	Assignee types	Developer community
(Čubranić and Murphy, 2004)	T3, T4	The real assignees of the selected bugs for the experiment (162 developers) are considered members of “developer community”.
(Canfora and Cerulo, 2006)	T1, T2	The real assignees of the selected bugs for the experiment (373 and 637 developers in two projects)
(Jeong et al., 2009)	T4	The real assignees of the selected bugs for the experiment (number of developers is not mentioned)
(Matter et al., 2009)	T1, T2, T4	The real assignees of all the bug reports (210 developers)
(Tamrawi et al., 2011a) (Tamrawi et al., 2011b)	T4	The real assignees of the selected bugs for the experiment (between 156 and 3,014 developers in 7 projects)
(Bhattacharya and Neamtiu, 2010) (Bhattacharya et al., 2012)	T3	The real assignees of all the bug reports (number of developers is not mentioned)
(Shokripour et al., 2012)	T1, T2	The real assignees of the selected bugs for the experiment (number of developers is not mentioned)
(Zhang et al., 2016c)	T3	The real assignees of all the bug reports except the developers who were assigned to only one bug (In total, between 70 and 874 developers in four projects)
(Cavalcanti et al., 2014a) (Cavalcanti et al., 2016)	T1, T2	The real assignees of the selected bugs for the experiment (70 developers)
(Sun et al., 2017)	T3, T4 <sup>6</sup>	The real assignees of all the bug reports (between 47 and 667 developers in 5 projects)

<sup>6</sup>the type of assignee in this paper was not directly mentioned in the text, but it is conceived indirectly from the text

Most previous research used the definitions of T1, T2<sup>7</sup>, T3 and T4 or a limited combination of them for the real assignee. No previous research, however, used the comprehensive combination of them, i.e., T5 (see Table 3). Note that these categories are inclusive of similar definitions across the existing literature which use other bug-tracking and version-control systems rather than Github. But no previous research has used the union of those types (e.g., T5 as we defined above).

Note that ideally, there can be another type of real assignee. The project manager can determine a list of candidates who would be proper developers to fix each bug. These developers might never have worked toward fixing that bug (due to high workload, unavailability or other reasons), but still would be included in the gold set. Although this definition would be the ideal golden truth, to the best of our knowledge, no previous study used it for its evaluation. It is very expensive to produce such a list for thousands of bugs in a big project. So, we just ignore it and only consider the five main types of assignee (e.g., T1 to T5) as we discussed above. These five types of assignee can be extracted from the issue-tracking system directly and easily.

## 5.2. “Developer community”

We define the *developer community* as the set of developers who are considered potential assignees in a project at any time. BA researchers try to sort and recommend the top developers in this set, based on their competence to fix a given bug. Just like how adopting a narrowed “golden truth” can impact the reported results, the size of this community affects the accuracy of the proposed BA methods. The more limited (and smaller) the developer community becomes, the easier the prediction of actual real assignee will be. For example, predicting the real assignee from a set of 10 developers is easier than a set of 500 candidates.

It is hard to obtain a unanimous definition for developer community. Github, through its comprehensive APIs, gives a list of collaborators to the project (Github, 2017b). Some of them are outside collaborators, who make contributions through pull requests (that should be reviewed before being approved as project contribution). These people do not have access to the organization but can have controlled contribution towards the project. Direct collaborators are appointed by the project managers and can have limited or full write access to the repository or organization. Also, there are organization members who have access to all the projects of the organization through team membership or other default organization permissions (note that in Github each organization is a virtual organization and can include several projects in it). Finally, there are the project managers and organization owners.

The project and organization managers can give different access levels to people regarding type of their collaboration. Even if a developer who is appointed by the managers as a project member (collaborator), does not contribute in the project (i.e., does not do any commits), he still is a project member. The fact is that the project managers found this developer a good candidate for contributing in the project. So, he should be considered as a possible assignee.

Some previous research considered all the project members as their developer community (Khatun and Sakib, 2016; Sajedi et al., 2015, 2016). This is the most comprehensive definition for developer community. Some others took a subset of the developers in the project (e.g., the committers) as the developer community (Hossen et al., 2014; Kagdi et al., 2008). Many others considered the set of previous assignees as developer community (Čubranić and Murphy, 2004; Anvik et al., 2006; Tamrawi et al., 2011b; Bhattacharya et al., 2012). Also, there are many others that did not have a clear definition for it (Rahman et al., 2009).

Regardless of the general definition for developer community, many of previous research have been filtering the community, to remove less-active developers and subtle bugs assigned to them. This is another restriction on the developer community in the data set and we discuss it later, along with the effect of developer community on the evaluation of the results.

---

<sup>7</sup>In some of the previous research it is not clear if they are using just T1 or a combination of T1 and T2. Due to lack of a clear separation in these cases, we assumed they considered both T1 and T2 in Table 3

## 6. Experiment setup and Data Set

To investigate the effect of adopted definition of “Real assignee” and “Developer community” on the evaluation of BA research, we perform a set of experiments. We use the findings of those experiments to answer the research questions of our study.

In our experiments, we use the IR notation where the description of the new bug is the query and the developers’ profiles (concatenated text of all the previously fixed bug reports by each developer) are the documents. We use the *tf-idf* method for giving a score to each document (developer) for the given, new query (bug report). This method or its variations were used previously to emphasize on specific keywords and de-emphasize the common terms, and shown to work well on textual data (Zhang et al., 2016c; Shokripour et al., 2015; Khatun and Sakib, 2016; Zhang et al., 2016a).

We applied the baseline *tf-idf* method to sort the developers for the given bug report. At each point in time, we considered each developer a document including the textual elements of all the bug reports assigned to that developer up to that time. We considered the bug report’s title and description, plus the main languages of the project (after removing stop-words) as the contents of these documents. We assumed main language of a project as any programming language that contains at least 15% of the lines of code of that project.

*Tf-idf* (Manning et al., 2008; Cavalcanti et al., 2014b; Shokripour et al., 2015) is an **IR measure** of the similarity between a query and a document. Having several documents, the document with highest similarity score with the given query is considered the document most similar (and most relevant as a response) to the given query. Having a **query (bug) “q”**, we use the following equation to calculate the score for **document (developer) “d”** in the **corpus “D”**:

$$\text{score}(\mathbf{q}, \mathbf{d}) = \sum_{i=1}^n \text{idf}(t_i, D) \cdot \text{tf}(t_i, d) \quad (1)$$

$n$  : number of terms in  $q$

In the above formula, *tf* (term frequency) measures number of times a term,  $t_i$ , is appeared in document  $d$ , normalized by document length. On the other hand, *idf* (inverse document frequency) measures the importance of a term with regard to all documents in the corpus  $D$  (Manning et al., 2008).

We use Equation 1 to evaluate the similarity of a bug and a developer. It assigns a score for each developer, regarding a given bug. Then we sort the developers in the potential-assignees community based on this relevance score from high to low. We implemented the above metric and the experiment using Java. Then, we ran our code and cross-validate our recommendations against the real assignees in our data set to recommend a ranked list of developers. Based on the position of all the real assignees in our recommended ranked list, we calculate the per-project and overall MAP. We made the source code of our experiments available online<sup>1</sup>.

### 6.1. The Data Set

In our previous research, we studied “several Github projects” (Sajedi et al., 2016), chosen due to their popularity in Github. In this paper, we used the new data for the same projects. There were 20 projects in our previous study, out of which we could extract the information of bug reports of 13 projects<sup>8</sup>. The other 7 projects stopped publishing their issues for public (e.g., switched to private issue tracking systems).

We extracted data of those 13 projects from their beginning (2009-04-28 or later, based on project start dates) to 2016-10-31. This data set is inclusive of the old data set and includes several times more bug reports. Another limitation that we resolved is that in our previous data set (Sajedi et al., 2016), we

---

<sup>8</sup>The link to the 13 projects we studied are: <http://github.com/lift/framework>, <http://github.com/html5rocks/www.html5rocks.com>, <http://github.com/yui/yui3>, <http://github.com/khan/khan-exercises>, <http://github.com/tryghost/ghost>, <http://github.com/fog/fog>, <http://github.com/julialang/julia>, <http://github.com/adobe/brackets>, <http://github.com/travis-ci/travis-ci>, <http://github.com/elastic/elasticsearch>, <http://github.com/saltstack/salt>, <http://github.com/angular/angular.js> and <http://github.com/rails/rails>

limited the developers to only the shared ones between Stack Overflow and Github (which was around 10% of the total developers). Here, we do not have such a dependency. So, we do not filter the data set. We extracted and stored information of all the project members in Github as developer community. To obtain the data, we wrote a set of JavaScript programs to extract the data of those projects online using Github APIs. The source code of this program is accessible online<sup>9</sup>.

Table 4 shows statistics of our data set, including different assignment types and the number of bug reports based on each definition. The important aspect of this data set is that we also reported the number of members in the developer community who have ever fixed any bugs during the captured lifetime of project. The minimum, average and median of this number are 38, 583 and 541 respectively, which shows the assignee prediction on this data set is not trivial. We made the data set available online<sup>1</sup> for further BA research.

Table 4: The data set including 13 projects and # of bug-assignments in each project, based on different assignment types, T1 to T5

Project	#of members in Developer Community	#of members in DC who have ever fixed any bugs	# of bugs	# of bug-assignments				
				T1: AUTHOR	T2: CO-AUTHOR	T3: ADMIN_CLOSER	T4: DRAFTED ASSIGNEE	T5: ALL TYPES
Framework	75	38	325	129	97	225	115	566
Html5rocks	(DC)159	47	627	90	1	638	269	998
Yui3	175	77	526	122	4	541	235	902
Khan-exercises	206	82	624	19	5	654	179	857
Ghost	473	357	3,578	1,371	113	3,713	945	6,142
Fog	770	208	1,124	91	27	1,146	63	1,327
Julia	831	541	9,086	1,759	54	9,590	1,345	12,748
Brackets	864	646	6,255	171	6	6,554	3,731	10,462
Travis-ci	1,159	1,096	5,473	13	2	5,716	603	6,334
Elasticsearch	1,262	758	10,423	2,192	498	10,362	3,132	16,184
Salt	2,283	1,227	10,237	2,344	233	10,682	2,274	15,533
Angular.js	2,386	1,069	7,402	503	196	7,671	1,288	9,658
Rails	4,079	1,431	8,794	857	138	9,366	944	11,305
Total	Average: 1,132 Median: 831	Average: 583 Median: 541	64,474	9,661	1,374	66,858	15,123	93,016

In order to capture different types of assignees, we used the definitions of T1 to T5 (Section 5.1) precisely. For the two assignee types related to the commits (i.e., T1 and T2), we needed to check the commit messages. According to Github documentaion (Github, 2017a), the fixers reference the bugs from commit messages with one of the following specific keywords (or the capital cases of any of the keywords), followed by an optional space, followed by a number sign (#) and one or more bug number(s)<sup>10</sup>:

- “fix”, “fixes”, “fixing”, “fixed”
- “close”, “closes”, “closing”, “closed”
- “resolve”, “resolves”, “resolving”, “resolved”

We cross-referenced this with issue events<sup>11</sup> (which were also extracted by our code using Github APIs)

<sup>9</sup><https://github.com/TaskAssignment/software-expertise>

<sup>10</sup>Bug numbers are iterative numbers, usually starting from 1. Pull requests also share this numbering system with bug reports in a manner that a number is considered only for a pull request or a bug report (called issue in Github).

<sup>11</sup>Issue events include every interaction about bugs including opening, closing, referencing, subscribing, assigning and reopening.



to avoid capturing of communications between developers or typos as bug resolving. For T2, we captured the referencing developer as a second assignee of this type only if the *committer* was different than the *author*. Otherwise we just considered one assignee as T1.

For T3 and T4 assignment types, we did not need commit messages, but we examined the issue events precisely (especially the events *closed*, *assigned* and *unassigned*) to extract those two types of assignment correctly. We also paid enough attention to the possible complications in which there are more than one assignee –e.g., a bug report is closed and opened again several times, each time assigned to a developer who does some work.

Our data set is one of the most complete data sets currently available for BA regarding size, number of bug reports and community members, duration of projects and selection of projects (i.e., based on popularity in Github). In terms of number of bug reports and especially size of *developer community*, it is one of the most extensive data sets, comparing with the respective values in other research in the field (even comparing the 13 selected BA studies).

We use the above experiment setup and data set for inspecting the research questions in next section. Our source code, data set, input and output files (as well as documentation for running the code in simple steps) are available online<sup>1</sup>.

## 7. Findings

In this section, we investigate the two dimensions of variability (mentioned in Section 5), and the effect they can have on the evaluation. Then we discuss the best choices regarding them.

### 7.1. Comparing Different Types of “Real Assignee”

In the previous sections, we discussed T1 through T4, the main types of “real assignees” used in the BA literature. We also proposed T5 as the union of the four. Here, we investigate which definition is better to be adopted as the golden truth in BA experiments. In other words, we address the following research question:

#### **RQ1: What is the best definition of “real assignee”?**

In order to study different types of real assignee and analyze their qualities, we implemented the baseline *tf-idf* method. We run this method using 5 different versions of the same data set related to T1 to T5 (see Table 4). Then we compare the results of this method on them and perform some statistical analyses.

Table 5 shows the results of the baseline method over 13 projects considering five different assignee types. The average overall MAP is shown in the last row. It starts from 34% for T1 and goes to 46% for T2 (around 35 percent difference, which is a big difference). The overall MAP regarding T5 is in a moderate level. This makes more sense since T5 is inclusive of all the extreme cases of other four definitions together. So, when using T5, those extreme cases would not easily bias the results, but will be considered along with other cases in a more reasonable way.

We also checked the variance in the distribution of the results based on each assignee type over the projects (for space limitation, we do not show the distribution diagrams here). T2 exhibits the highest variance over different projects. Also, T1, T3 and T4 have outliers, which shows the unexpected difference for different projects. On the other hand, T5 has low variance and no outliers. Overall, considering these differences with the fact that T5 produces a moderate average MAP, we can say that T5 gives the results in a more acceptable and robust range (without much tolerance over different projects). Note that in Table 4, some projects have only a small number of bug-assignments (e.g., less than 50) regarding T1 or T2. To verify that their high variance is not specific to those projects, we excluded those projects and re-analyzed the distributions (not shown here). Interestingly, again, T2 has the highest variance and T5 shows the most robust behavior over different projects. A quick look over the results of each project regarding T1 to T4 in Table 5 shows their high disparity. As the reported results based on those four gold-sets would vary a lot for a project, we see that those narrow definitions can lead to biased or extreme results in different projects. Hence, picking any of them for evaluation of a research can make subjective results.

Table 5: MAP for 13 projects using T1 to T5 as “assignee type”

Project \ Assignee Type	T1	T2	T3	T4	T5
Framework	56.41	83.02	51.24	44.19	49.13
Html5rocks	69.73	1.82	70.29	37.99	59.85
Yui3	48.69	13.75	40.24	47.40	53.13
Khan-exercises	28.06	1.70	39.32	48.84	41.80
Ghost	32.99	76.64	77.02	42.44	58.33
Fog	44.74	55.66	63.54	42.71	60.58
Julia	37.41	37.85	44.12	62.34	45.32
Brackets	38.36	42.08	32.71	37.17	34.70
Travis-ci	23.21	50.71	50.93	61.08	52.63
Elasticsearch	37.80	31.59	44.63	29.84	37.95
Salt	31.29	68.64	36.00	39.24	35.22
Angular.js	35.67	42.93	36.49	40.72	37.21
Rails	17.40	19.83	34.56	38.76	32.21
Total (13 proj)	34.27	46.25	42.51	40.28	40.52

In addition, we believe that each definition, from T1 to T4, is targeting only a specific aspect of bug fix. So, considering the results based on any of them as gold-set measures how good the proposed approach is regarding that specific aspect. The nature of T1, for example, is targeting whoever commits the code as the regular programmer. T3, however, indicates a developer with some higher-level status in the project (e.g., a core developer), who reviews the code and closes the bug. In some cases, this developer brings the bug to the attention of appropriate programmers in the team. The “regular developer” nature of T1 is not comparable with the “authoritative developer” nature of T3. In fact, each of the assignment types are targeting different golden truth, without having major coverage with other ones.

Since narrowed types of assignee just capture a limited type of work towards fixing a bug, evaluation or making inferences, based on these narrowed definitions may lead to subjective judgments and would bias the evaluation process. Software development is a collaborative work; many bugs are fixed by a group of developers, who can have different roles in the project. So, all the developers who perform any type of development work towards fixing a bug should count as proper assignees of that bug.

Considering from another point of view, the adopted definition for real assignee should not be impractically *broad* or *narrow*; Too broad definitions of real assignee can cause superficially high precisions because many developers would be assumed bug fixers of the bug (as mentioned in the above example). On the other hand, too narrow definition of *real assignee* can limit the total number and diversity of assignees. This fact can then be utilized by any insignificant approach to obtain high accuracies only by prioritizing the previous assignees at any point in time. Even we see later that it can restrict the *developer community* (if the developer community is not defined correctly) and bias in precision, recall, or accuracies.

When dealing with industrial projects, considering T5 –as the union of T1 to T4– compensates the bias in the single definitions and makes a richer data set including all types of work (towards bug-fix). T5 covers different indications of bug-fix on which we can fairly validate a BA approach. Since the assignees of all those types have done some useful work toward fixing a given bug, suggestion of any of those developers to the project manager would be some help towards fixing the bug.

Based on all the above analyses, we can conclude that:

The most comprehensive definition of “real assignee” is T5 (ALL\_TYPES). While it is essential to consider different types of work towards fixing a bug as indication of assignee, T5 enables us to capture this variety of assignees. This set of assignees is a proper gold standard for evaluating goodness of an assignee recommendation approach.

In the next sections, we perform the rest of our analyses based on this golden truth (T5).

## 7.2. The effect of “Developer Community”

The size of developer community is another important factor in evaluation of BA research. The bigger the developer community becomes, the more difficult the prediction of actual real assignee will be. In the previous sections, we discussed that “all the project members”, “committers” and “set of all the developers who have assigned any bugs during lifetime of project” are the mostly used options in previous research regarding developer community.

Regardless of the definition of developer community, many of the previous research tried to filter the members of this community in their data set (i.e., remove developers who have fixed less than a threshold number of bugs), which results in removing a number of bugs as well.

We would like to investigate which definition is more appropriate for evaluation of the results. Also, we want to understand if there is any side effect (e.g., bias in the results) in filtering this community or not. We address the following research question:

**RQ2: What is the best definition for “developer community” from which the bug-assignment methods recommend appropriate developers? And what is the effect of filtering this community?**

The assumptions of BA experiments should be realistic, to be useful in industrial applications. The definition of “developer community” determines a set of developers from which a BA approach recommends somebody to fix a bug. So, restricting this set to limited subsets of project members (e.g., the previous assignees, or the committers) eliminates the usage of the proposed BA method. Most open-source software companies cannot differentiate between their developers or restrict their bug-fixers to a subset of their developers. Note that in reality, even we do not know which developers would commit or fix a bug in advance, to limit our list to those developers.

Hence, we choose “*all the project members*” as the practical definition of *developer community*. It is the most realistic option and it is comprehensive enough to contain any developer who works in the project and would step in to fix a new bug report. In open-source projects, the list of project members is easily accessible<sup>12</sup>.

Also note that when considering previous assignees (instead of all developers in the project) as the *developer community*, this community would be sensitive to the adopted definition of real assignee (e.g., T1 to T5; narrower definition of real assignee makes the community smaller). This might itself lead to more subjective judgments. Especially some evaluation metrics (e.g., top-10 accuracy and precision @10) are highly affected by a narrow definition of developer community. Suppose that a developer community is narrowly defined to contain 15 developers and each bug report has a single real assignee. Then, even a random selection of developers would superficially give high accuracy (i.e., 66.67% top-10).

*Filtering* the data set is another adverse restriction. In our survey, we saw many examples of BA research that suffer from filtering data sets; this, biases the results and makes the reproduction of the research difficult (note that different data sets react differently against filtering). While we are not interested in pointing to those research here, we would like to investigate the effect of this filtering on the evaluation results in a high level. The studies that perform a data filtering, usually remove less-active developers. Some mention that they removed bug reports assigned to developers who fixed less than a threshold number of bugs. These two cases have the same result; they eliminate both developers and bug reports and get rid of the challenging bug reports. In fact, those bug reports were **specific bug reports** that need more in-depth investigation to be connected (and assigned) to some **specific developers**, both of which are removed from the data set for simplicity. In fact, in the real, industrial projects, these developers are not removed from the project, and, are actually important parts of the project.

Filtering less-active developers can cause flimsily high precision, recall, or accuracies by artificially shrinking the *community of developers* considered. This was verified before by Lee et al. (Lee et al., 2017). They compared their method on two versions of the data set, for three open-source projects; a

---

<sup>12</sup>As an example of list of project members in open source projects, Github has APIs for returning members of a project; <https://developer.github.com/v3/repos/collaborators/>

Least number of bugs fixed by each developer	Considered developers	Considered bug reports		Overall MAP
		Number	Percentage	
1 (no filter)	7577 (all)	93016 (all)	100	40.52
2	1956	87,395	95	42.68
6	568	83,794	90	44.30
40	236	79,125	85	46.46
88	186	74,506	80	48.52
145	156	69,832	75	50.55
202	113	65,208	70	53.32
321	67	60,274	65	55.62
415	55	55,978	60	59.00
532	44	50,981	55	62.02
613	36	46,431	50	65.50
656	29	42,021	45	68.22
872	22	36,979	40	74.13
1090	17	32,205	35	79.57
1269	13	27,673	30	86.26
1435	10	23,735	25	92.31
1919	7	19,185	20	95.66


Correlation: -0.98  negative (almost) linear relationship

Figure 1: The effect of filtering less-active developers, and the bug reports they fixed, on accuracy of results; first row is the complete data set (no filtering), and the last row is 20% of the data set.

version including all the developers, and another one including only the developers who fixed at least 10 bug reports. Interestingly, they obtained much better results in the second case. Similarly, Tamrawi, et al. (Tamrawi et al., 2011b) reported increases in their accuracy, and, Anvik and Murphy (Anvik and Murphy, 2011) reported increases in their recall by filtering the less active developers. In another study, Canfora and Cerulo (Canfora and Cerulo, 2006) showed that filtering the developers to 100 (out of  $\sim 400$ , to  $\sim 600$  developers in Mozilla and KDE projects), multiplies the recall by around 3 and 4 respectively.

To have a deeper understanding of the effect of filtering developers on the accuracy of bug assignment, we ran an experiment using the baseline *tf-idf* approach as we discussed before (again, we made the source code of this experiment available online<sup>1</sup>). We considered T5 as the definition of assignee and used MAP for evaluation. We ran the program several times and each time, applied a different filtering on the developer community and bug reports fixed by them. Then we captured the overall MAP. The results are shown in Figure 1. Moving from top to bottom, the number of bug reports decreases ( $\sim 5\%$  in each row). The first row is the complete data set. In the next rows, the less-active developers and the bugs they fixed are filtered. The cut-offs are selected so that in each row, the number of bugs is decreased around 5% of the total bugs, with respect to the higher row. In each row, the remaining bug reports are those that were handled by the most active developers. In the last row, only 20% of the bugs are considered. It is remarkable that the MAP increases up to 95%. In other words, a simple filtering on the developer community and their fixed bugs can increase the overall MAP by a factor of 150% (make it 2.5 times the original value) and make the overall MAP close to 100%. We also obtained the correlation between “percentage of remaining bugs that we run the algorithm on”, and “overall MAP”. They have a negative linear correlation (-0.98) which shows how the filtering can manipulate the results. Note that this is only an example with *tf-idf* as a baseline method. One could envision manipulating the utilized method to gain more benefit from the narrowed list (i.e., considering the time of activity of developers as well) and enhance results after filtering. Also note that this effect can be different (higher or lower) in another data set, based on the dynamics of that data set.

So, developer filtering can cause to obtain dramatically higher (biased) results than would be obtained

in real cases. In industrial projects, the project managers need realistic solutions that consider all the developers; even the less active developers may fix some bugs and the project managers need to take them into account. According to several previous research (Wang et al., 2014; Sean et al.; Zhang and Hurley, 2008), filtering less-active developers reduces the diversity of developer recommendation, which results in diminishing practicality of the proposed BA approach.

We conclude that the size of developer community should be reported clearly. In addition, the number of members of developer community who have ever fixed any bugs during lifetime of the project is also important. It is another useful indicator of the broadness of the project and should be reported. It gives us an indication of difficulty of predictions. Failing to do so makes it harder to compare against other research. In general, regarding the proposed research question, we conclude that:

The most practical definition to adopt as developer community is “all the project members”. It is the most realistic and comprehensive option to contain any developer who works in the project and may step in to fix a given bug. Filtering this set might produce biased results and mislead the evaluation.

## 8. Discussion

In our comprehensive survey on BA, we found that the field suffers from lack of a systematic framework for evaluation. Many of previous research are using small projects (or filtering the data of big projects and obtain limited circumstances) in which the assignee prediction is straightforward. Many of them implement another previously published method, to validate their approach through comparison. But again, the reporting may be skewed. Note that we are not criticizing the valuable amount of work toward research in the field. We identified lack of a systematic framework to facilitate (and standardize) the evaluation and reporting of scientific claims in BA domain, which needs more attention from software engineering community. In order to validate a new BA method and show that it works better than the previous approaches, we need to compare it against one or more previously published methods. Generally, there are three ways to do this: (1) to run the code of the previous approach and test it on our data, (2) to test the code of our approach on the data of the previous research, and (3) to compare the two approaches considering the reported evaluation metrics (e.g., meta analysis).

Initially, the first option above seems the best solution. Many research publications utilized their comparisons this way. However, in our survey, we found that most previous research did not publish their code. On the other hand, it is almost impossible to take into account all the details of a previous approach and re-implement it (note that usually the implementation details are not mentioned in the papers). Re-implementation of a generic version of their approach is possible but less efficient. Moreover, many of them have no comprehensive source code and are, in fact, just a series of commands or transactions (e.g., interactions with Weka) which are almost impossible to replicate. Even if the previous researchers published their code or exact instructions of the needed transactions, our data might be inefficient for their approach. Finally, a research might have a data cleaning, that is a critical part and have important effects on its results.

The second option is a better solution, since we do not judge about the previous approach unless from their reported results. However, as we encountered in our survey, most previous research in the field did not publish their data, or if they did, the link to their data is broken. Even having their data, it might not be compatible with our approach (e.g., assume that our method needs component information which is not available in the data set of a previous research).

Despite the fact that meta-analysis (the third above-mentioned option) is not used much in the field, it can provide appropriate comparison against the previous research. It is not dependent on the data or code of previous research. Then, it is feasible to compare against many previous research (e.g., the ones mentioned in Table 2). Doing so, excelling a bunch of approaches is more persuasive than outperforming only one or two implementations. However, the project conditions and characteristics might affect the comparisons in this option and make the comparison unfair.

In either case, validation of a new approach needs *fair* comparison against state-of-the-art methods. In this way, observing several points as mentioned in the next sub-section are essential.

### 8.1. The proposed evaluation framework

We provide our evaluation framework based on the discussed research questions of the paper, and regardless of how we compare a new approach against state-of-the-art methods (above three options). It contains guidelines for maintaining a reproducible BA research, with standards for judgment of scientific claims (Peng, 2011) and replicating the study (Fomel and Claerbout, 2009). They cover important aspects about evaluation, reporting and comparison against state-of-the-art methods:

1. Evaluation should include reporting based on MAP as the most stable and inclusive evaluation metric. MAP is a single measure, representative of both precision and recall. It bundles the rank of all the assignees for each bug report, with emphasis on higher ranks. When there are several projects, an overall MAP for all the bug reports over all the projects is extremely useful. This helps to compare fairly when project-based comparison of two methods is difficult. In addition to MAP, we recommend calculating and reporting the other widely used metrics (see Table 2), to enable meta-analysis and comparison against the previous approaches that reported other metrics.
2. The most comprehensive definition of real assignee, T5, should be considered for cross-validation. It includes every development work towards fixing the bugs. Using this as the ground truth, the results would be less sensitive to specific types of work (e.g., commit history) or other parameters in the project (e.g., number of bug reports in the project).
3. “All the project members” should be considered as the developer community. This generalization makes the proposed BA approach comprehensive and useful for industrial projects, in which all the developers may step in to fix a bug. This list, like real assignees, can be extracted from open-source projects easily. The important point is not to use any filtering (on bugs or developers). Filtering would eliminate the developer community, reduce the challenging bug reports and make an isolated evaluation which might not be representative of real situations.
4. The cross-validation should be done on relatively large number of bug reports, number of bug-assignments (since a bug can be assigned several times), size of developer community and number of developer community members who have ever fixed any bugs. If this last number is too small (e.g., roughly around 20 or less) in a project, then that project is not a good case for cross-validation, since it is not challenging enough to reflect the goodness of different methods. Also, to avoid judgments based on limited data, it is recommended to test on fairly large number of bug-assignments (e.g., roughly around 500 or more). After all, it is needed to report all these details per project.

To publish reproducible BA research, we recommend the above baselines to be established. This makes the reporting, review, judgment, comparison and replication of the study easier.

### 8.2. Threats to validity

Most reasonings of this paper were based on intuitions obtained from our survey (e.g., the metrics used, and the dynamics and settings used in various BA studies, as well as their data sets).

A threat to validity is that we supported our arguments based on some experiments on our data set. These arguments might do differently in other data sets and settings. While we admit this as a threat, we argue that first, the data set and settings we considered is one of the most comprehensive cases in the whole literature. It includes 13 big projects, thousands of active developers and near 100k of bug-assignments which is non-trivial. Moreover, the experiments we established are not the only basis for our statements. This paper is mainly based on our experience in several years of BA research, the comprehensive survey we proposed, the arguments we provided, and strong evidence from the literature. Having said that, the experiments also provide extra support to the claims, or bring the current problems to the attention of the reader (e.g., by providing some examples or counter examples).

To capture the real assignees (i.e., the golden truth to validate our approach), we used projects that use Github’s issue tracker. We looked for bugs in Github projects and their certain links to the commits and issue (bug) events, to find the real assignees. Although it is a common practice to mention and preserve those links in Github’s open-source projects (Github, 2017a), there might be some missing links (Bachmann et al., 2010) that we did not consider. This can potentially affect our validations. However, note that validation of these cases and inclusion of the full links between commits and bug reports is a tedious task, that must be done manually (Bachmann et al., 2010). To the best of our knowledge, no previous research has done this manual process. All the previous BA research in the field cross-validated their approach against heuristic-based golden-truth extracted automatically from available data of software projects.

## 9. Conclusions

In this paper, we accomplished three main contributions;

1. We comprehensively surveyed the previous research in BA, and reviewed the different BA objectives, methods, used data and dimensions of variability. This survey can be highly useful for further researchers planning to explore and research in the field.
2. We proposed a framework for evaluation of BA research; We investigated the mostly used evaluation metrics in BA studies. We argued that MAP is the best evaluation metric. It can be independently (from other metrics) interpreted and considers rank of all the real assignees. It emphasizes the higher ranks and is not highly affected by number of real assignees. Despite all these benefits, and its wide usage in evaluation of ranked retrieval results (Manning et al., 2008), it is rarely used in BA. We hope that this study aids for its adoption and usage in further BA research to provide more reliable evaluation.

Then, we demonstrated the impact of two important dimensions of variability and provided arguments for establishing a realistic evaluation; first, *definition of real assignee*, which is used as the ground truth in BA research, should comprehensively include every bug-fix effort by developers. Second, the *developer community*, who are the potential assignees used to validate a BA approach, should be inclusive of all project members.

All in all, validating a new BA approach needs some spirit of equity and fairness. In our proposed framework, we mentioned the important aspects of evaluation and reporting BA research. Addressing those aspects enables replication of the study and promotes its usage in other research or industrial applications.

3. Finally, the data set we extracted from popular Github projects contains the full set of assignees based on the comprehensive definitions of real assignee and developer community. We made it available online<sup>1</sup> for further research. This data set is one of the most comprehensive and recent data sets available for further BA research.

## Acknowledgements

The work is supported by Graduate Student Scholarship<sup>13</sup> funded by Alberta Innovates - Technology Futures (AITF)<sup>14</sup> and Queen Elizabeth II Graduate Scholarship<sup>15</sup> funded by Faculty of Graduate Studies and Research (FGSR)<sup>16</sup> at University of Alberta.

<sup>13</sup><https://fund.albertainnovates.ca/Fund/BasicResearch/GraduateStudentScholarships.aspx>

<sup>14</sup><https://innotech.alberta.ca>

<sup>15</sup><https://www.ualberta.ca/graduate-studies/awards-and-funding/scholarships/queen-elizabeth-ii>

<sup>16</sup><https://www.ualberta.ca/graduate-studies>

## References

- Aggarwal, K., Timbers, F., Rutgers, T., Hindle, A., Stroulia, E., Greiner, R.. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process* 2017;29(3).
- Ahsan, S.N., Ferzund, J., Wotawa, F.. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In: *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*. IEEE; 2009. p. 216–221.
- Akbarinasaji, S., Caglayan, B., Bener, A.. Predicting bug-fixing time: A replication study using an open source software project. *Journal of Systems and Software* 2017;.
- Alipour, A.. A CONTEXTUAL APPROACH TOWARDS MORE ACCURATE DUPLICATE BUG REPORT DETECTION. Master's thesis; University of Alberta; Canada; 2013.
- Aljarah, I., Banitaan, S., Abufardeh, S., Jin, W., Salem, S.. Selecting discriminating terms for bug assignment: a formal analysis. In: *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM; 2011. p. 12.
- Anjali, , Mohan, D., Sardana, N., et al. Visheshagya: Time based expertise model for bug report assignment. In: *Contemporary Computing (IC3), 2016 Ninth International Conference on*. IEEE; 2016. p. 1–6.
- Anvik, J.. Automating bug report assignment. In: *Proceedings of the 28th international conference on Software engineering*. ACM; 2006. p. 937–940.
- Anvik, J., Hiew, L., Murphy, G.C.. Who should fix this bug? In: *Proceedings of the 28th international conference on Software engineering*. ACM; 2006. p. 361–370.
- Anvik, J., Murphy, G.C.. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2011;20(3):10.
- Bachmann, A., Bird, C., Rahman, F., Devanbu, P., Bernstein, A.. The missing links: bugs and bug-fix commits. In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM; 2010. p. 97–106.
- Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., Cukic, B.. Automated triaging of very large bug repositories. *Information and Software Technology* 2016;URL: <http://www.sciencedirect.com/science/article/pii/S0950584916301653>. doi:<https://doi.org/10.1016/j.infsof.2016.09.006>.
- Banitaan, S., Alenezi, M.. Tram: An approach for assigning bug reports using their metadata. In: *Communications and Information Technology (ICCIT), 2013 Third International Conference on*. IEEE; 2013. p. 215–219.
- Baysal, O., Godfrey, M.W., Cohen, R.. A bug you like: A framework for automated assignment of bugs. In: *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE; 2009. p. 297–298.
- Baysal, O., Holmes, R., Godfrey, M.W.. Revisiting bug triage and resolution practices. In: *Proceedings of the First International Workshop on User Evaluation for Software Engineering Researchers*. IEEE Press; 2012. p. 29–30.
- Bhattacharya, P., Neamtiu, I.. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE; 2010. p. 1–10.



- Bhattacharya, P., Neamtiu, I., Shelton, C.R.. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software* 2012;85(10):2275–2292.
- Borg, M.. Embrace your issues: compassing the software engineering landscape using bug reports. In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM; 2014. p. 891–894.
- Bortis, G., Hoek, A.v.d.. Porchlight: A tag-based approach to bug triaging. In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press; 2013. p. 342–351.
- Budgen, D., Brereton, P.. Performing systematic literature reviews in software engineering. In: *Proceedings of the 28th international conference on Software engineering*. ACM; 2006. p. 1051–1052. URL: <https://dl.acm.org/citation.cfm?id=1134500>.
- Canfora, G., Cerulo, L.. Supporting change request assignment in open source development. In: *Proceedings of the 2006 ACM symposium on Applied computing*. ACM; 2006. p. 1767–1772.
- Cavalcanti, Y.C., do Carmo Machado, I., Neto, P.A.d.M.S., de Almeida, E.S.. Towards semi-automated assignment of software change requests. *Journal of Systems and Software* 2016;115:82–101.
- Cavalcanti, Y.C., Machado, I.d.C., Neto, P.A., de Almeida, E.S., Meira, S.R.d.L.. Combining rule-based and information retrieval techniques to assign software change requests. In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM; 2014a. p. 325–330.
- Cavalcanti, Y.C., Mota Silveira Neto, P.A., Machado, I.d.C., Vale, T.F., Almeida, E.S., Meira, S.R.d.L.. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process* 2014b;26(7):620–653.
- Chaparro, O.. Improving bug reporting, duplicate detection, and localization. In: *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press; 2017. p. 421–424.
- Chen, L., Wang, X., Liu, C.. Improving bug assignment with bug tossing graphs and bug similarities. In: *Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on*. IEEE; 2010. p. 1–5.
- Čubranić, D., Murphy, G.C.. Automatic bug triage using text categorization. In: *SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer; 2004. .
- Dedík, V., Rossi, B.. Automated bug triaging in an industrial context. In: *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*. IEEE; 2016. p. 363–367.
- Florea, A.C., Anvik, J., Andonie, R.. Parallel implementation of a bug report assignment recommender using deep learning. In: *International Conference on Artificial Neural Networks*. Springer; 2017a. p. 64–71.
- Florea, A.C., Anvik, J., Andonie, R.. Spark-based cluster implementation of a bug report assignment recommender system. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer; 2017b. p. 31–42.
- Fomel, S., Claerbout, J.F.. Guest editors’ introduction: Reproducible research. *Computing in Science Engineering* 2009;11(1):5–7. doi:10.1109/MCSE.2009.14.
- Github, . Closing issues using keywords. 2017a. URL: <https://help.github.com/articles/closing-issues-using-keywords/>.
- Github, . Collaborators. 2017b. URL: <https://developer.github.com/v3/repos/collaborators/>.

- Goyal, A.. Effective bug triage for non reproducible bugs. In: Proceedings of the 39th International Conference on Software Engineering Companion. IEEE Press; 2017. p. 487–488.
- Helming, J., Arndt, H., Hodaie, Z., Koegel, M., Narayan, N.. Automatic assignment of work items. In: International Conference on Evaluation of Novel Approaches to Software Engineering. Springer; 2010. p. 236–250.
- Hosseini, H., Nguyen, R., Godfrey, M.W.. A market-based bug allocation mechanism using predictive bug lifetimes. In: Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on. IEEE; 2012. p. 149–158.
- Hossen, M.K., Kagdi, H., Poshyvanyk, D.. Amalgamating source code authors, maintainers, and change proneness to triage change requests. In: Proceedings of the 22nd International Conference on Program Comprehension. ACM; 2014. p. 130–141.
- Hu, H., Zhang, H., Xuan, J., Sun, W.. Effective bug triage based on historical bug-fix information. In: Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on. IEEE; 2014. p. 122–132.
- Imtiaz, S., Ikram, N.. Dynamics of task allocation in global software development. *Journal of Software: Evolution and Process* 2017;29(1):e1832.
- Jain, S., Wilson, S.R.. Automated bug assortment system in datasets. In: Inventive Computation Technologies (ICICT), International Conference on. IEEE; volume 2; 2016. p. 1–7.
- Jain, V., Rath, A., Ramaswamy, S.. Field weighting for automatic bug triaging systems. In: Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on. IEEE; 2012. p. 2845–2848.
- Jeong, G., Kim, S., Zimmermann, T.. Improving bug triage with bug tossing graphs. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. ACM; ESEC/FSE '09; 2009. p. 111–120.
- Jie, Z., XiaoYin, W., Dan, H., Bing, X., Lu, Z., Hong, M.. A survey on bug-report analysis. *SCIENCE CHINA Information Sciences* 2015;58:1–24. doi:10.1007/s11432-014-5241-2.
- Jonsson, L.. Increasing anomaly handling efficiency in large organizations using applied machine learning. In: Software Engineering (ICSE), 2013 35th International Conference on. IEEE; 2013. p. 1361–1364.
- Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., Runeson, P.. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering* 2016;21(4):1533–1578.
- Jonsson, L., Broman, D., Sandahl, K., Eldh, S.. Towards automated anomaly report assignment in large complex systems using stacked generalization. In: Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE; 2012. p. 437–446.
- Kagdi, H., Gethers, M., Poshyvanyk, D., Hammad, M.. Assigning change requests to software developers. *Journal of Software: Evolution and Process* 2012;24(1):3–33.
- Kagdi, H., Hammad, M., Maletic, J.I.. Who can help me with this source code change? In: Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. IEEE; 2008. p. 157–166.
- Kagdi, H., Poshyvanyk, D.. Who can help me with this change request? In: Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE; 2009. p. 273–277.
- Karim, M.R., Ruhe, G., Rahman, M., Garousi, V., Zimmermann, T., et al. An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *Journal of Software: Evolution and Process* 2016;28(12):1025–1060.

- Keivic, K., Müller, S.C., Fritz, T., Gall, H.C.. Collaborative bug triaging using textual similarities and change set analysis. In: Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on. IEEE; 2013. p. 17–24.
- Khalil, E., Assaf, M., Sayyad, A.S.. Human resource optimization for bug fixing: balancing short-term and long-term objectives. In: International Symposium on Search Based Software Engineering. Springer; 2017. p. 124–129.
- Khatun, A., Sakib, K.. A bug assignment technique based on bug fixing expertise and source commit recency of developers. In: Computer and Information Technology (ICCIT), 2016 19th International Conference on. IEEE; 2016. p. 592–597.
- Kim, S., Whitehead Jr, E.J.. How long did it take to fix bugs? In: Proceedings of the 2006 international workshop on Mining software repositories. ACM; 2006. p. 173–174.
- Kitchenham, B., Charters, S.. Guidelines for performing systematic literature reviews in software engineering. Software Engineering Group, School of Computer Science and Mathematics, Keele University and Department of Computer Science, University of Durham, Tech Rep, EBSE 2007;URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471>.
- Lee, S.R., Heo, M.J., Lee, C.G., Kim, M., Jeong, G.. Applying deep learning based automatic bug triager to industrial projects. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM; 2017. p. 926–931.
- Lin, Z., Shu, F., Yang, Y., Hu, C., Wang, Q.. An empirical study on bug assignment automation using chinese bug data. In: Empirical software engineering and measurement, 2009. ESEM 2009. 3rd international symposium on. IEEE; 2009. p. 451–455.
- Linares-Vásquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M., Poshyvanyk, D.. Triageing incoming change requests: Bug or commit history, or code authorship? In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE; 2012. p. 451–460.
- Liu, J., Tian, Y., Yu, X., Yang, Z., Jia, X., Ma, C., Xu, Z.. A multi-source approach for bug triage. International Journal of Software Engineering and Knowledge Engineering 2016;26(09n10):1593–1604.
- Manning, C.D., Raghavan, P., Schütze, H.. Introduction to Information Retrieval. Cambridge University Press, 2008.
- Matter, D., Kuhn, A., Nierstrasz, O.. Assigning bug reports using a vocabulary-based expertise model of developers. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. IEEE; 2009. p. 131–140.
- Naguib, H., Narayan, N., Brugge, B., Helal, D.. Bug report assignee recommendation using activity profiles. In: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE; 2013. .
- Nasim, S., Razzaq, S., Ferzund, J.. Automated change request triage using alpha frequency matrix. In: Frontiers of Information Technology (FIT), 2011. IEEE; 2011. p. 298–302.
- Nguyen, T.T., Nguyen, A.T., Nguyen, T.N.. Topic-based, time-aware bug assignment. SIGSOFT Softw Eng Notes 2014;39(1):1–4. URL: <http://doi.acm.org/10.1145/2557833.2560585>. doi:10.1145/2557833.2560585.
- Park, J., Lee, M., Kim, J., Hwang, S., Kim, S.. Costriage: A cost-aware triage algorithm for bug reporting systems. In: Proceedings of the National Conference on Artificial Intelligence. 2011. p. 139.
- Park, J.w., Lee, M.W., Kim, J., Hwang, S.w., Kim, S.. Cost-aware triage ranking algorithms for bug reporting systems. Knowledge and Information Systems 2016;48(3):679–705.

- Peng, R.D.. Reproducible research in computational science. *Science* 2011;334(6060):1226–1227.
- Rahman, M.M., Ruhe, G., Zimmermann, T.. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society; 2009. p. 439–442.
- Rahmana, M.M., Karima, M.R., Ruhe, G., Garousic, V., Zimmermann, T.. An empirical investigation of a genetic algorithm for developer’s assignment to bugs. In: *Proceedings of the First North American Search based Symposium*. 2012. .
- Saha, R.K., Khurshid, S., Perry, D.E.. Understanding the triaging and fixing processes of long lived bugs. *Information and Software Technology* 2015;65:114–128.
- Sahu, T.P., Nagwani, N.K., Verma, S.. An empirical analysis on reducing open source software development tasks using stack overflow. *Indian Journal of Science and Technology* 2016;9(21).
- Sajedi, A., Hindle, A., Stroulia, E.. Crowdsourced bug triaging. In: *ICSME ’15*. IEEE; 2015. .
- Sajedi, A., Hindle, A., Stroulia, E.. Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment. In: *Proceedings of 19th International Conference on Fundamental Approaches to Software Engineering (FASE)*. Springer; FASE ’16; 2016. .
- Schwab, M., Karrenbach, M., Claerbout, J.. Making scientific computations reproducible. *Computing in Science & Engineering* 2000;2(6):61–67.
- Seacord, R.C., Plakosh, D., Lewis, G.A.. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional, 2003.
- Sean, J., McNee, M., Konstan, J.. Accurate is not always good: How accuracy metrics have hurt recommender systems. *extended abstracts on Human factors in computing systems (CHI06)* p ;:1097–1101.
- Servant, F., Jones, J.A.. Whosefault: automatic developer-to-fault assignment through fault localization. In: *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press; 2012. p. 36–46.
- Shani, G., Gunawardana, A.. Evaluating recommendation systems. In: *Recommender systems handbook*. Springer; 2011. p. 257–297.
- Sharma, M., Kumari, M., Singh, V.. Bug assignee prediction using association rule mining. In: *International Conference on Computational Science and Its Applications*. Springer; 2015. p. 444–457.
- Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A., Oliver, N.. Tfmap: optimizing map for top-n context-aware recommendation. In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM; 2012. p. 155–164.
- Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press; 2013. p. 2–11.
- Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.. Improving automatic bug assignment using time-metadata in term-weighting. *IET Software* 2014;8(6):269–278.
- Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.. A time-based approach to automatic bug report assignment. *Journal of Systems and Software* 2015;102:109–122.
- Shokripour, R., Kasirun, Z., Zamani, S., Anvik, J.. Automatic bug assignment using information extraction methods. In: *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*. 2012. p. 144–149. doi:10.1109/ACSAT.2012.56.

- Sisman, B., Akbar, S.A., Kak, A.C.. Exploiting spatial code proximity and order for improved source code retrieval for bug localization. *Journal of Software: Evolution and Process* 2017;29(1):e1805.
- Somasundaram, K., Murphy, G.C.. Automatic categorization of bug reports using latent dirichlet allocation. In: *Proceedings of the 5th India software engineering conference*. ACM; 2012. p. 125–130.
- Sun, X., Liu, X., Hu, J., Zhu, J.. Empirical studies on the nlp techniques for source code data preprocessing. In: *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies*. ACM; 2014. p. 32–39.
- Sun, X., Yang, H., Xia, X., Li, B.. Enhancing developer recommendation with supplementary information via mining historical commits. *Journal of Systems and Software* 2017;134:355–368.
- Tamrawi, A., Nguyen, T.T., Al-Kofahi, J., Nguyen, T.N.. Fuzzy set-based automatic bug triaging: Nier track. In: *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE; 2011a. p. 884–887.
- Tamrawi, A., Nguyen, T.T., Al-Kofahi, J.M., Nguyen, T.N.. Fuzzy set and cache-based approach for bug triaging. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM; 2011b. p. 365–375.
- Tian, Y., Wijedasa, D., Lo, D., Le Goues, C.. Learning to rank for bug report assignee recommendation. In: *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE; 2016. p. 1–10.
- Wang, S., Lo, D.. Amalgam+: Composing rich information sources for accurate bug localization. *Journal of Software: Evolution and Process* 2016;28(10):921–942.
- Wang, S., Zhang, W., Wang, Q.. Fixercache: Unsupervised caching active developers for diverse bug triage. In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM; 2014. p. 25.
- Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.. An approach to detecting duplicate bug reports using natural language and execution information. In: *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE; 2008. p. 461–470.
- Weidt, F., Silva, R.. Systematic literature review in computer science-a practical guide. *Relatórios Técnicos do DCC/UFJF* 2016;1.
- Wu, W., Zhang, W., Yang, Y., Wang, Q.. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In: *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*. IEEE; 2011. p. 389–396.
- Xia, X., Lo, D., Ding, Y., Al-Kofahi, J.M., Nguyen, T.N., Wang, X.. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering* 2017;43(3):272–297.
- Xia, X., Lo, D., Wang, X., Zhou, B.. Dual analysis for recommending developers to resolve bugs. *Journal of Software: Evolution and Process* 2015;27(3):195–220.
- Xie, X., Zhang, W., Yang, Y., Wang, Q.. Dretom: Developer recommendation based on topic models for bug resolution. In: *Proceedings of the 8th international conference on predictive models in software engineering*. ACM; 2012. p. 19–28.
- Xu, G., Zhang, Y., Li, L.. *Web mining and social networking: techniques and applications*. volume 6. Springer Science & Business Media, 2010.
- Xuan, J., Jiang, H., Ren, Z., Zou, W.. Developer prioritization in bug repositories. In: *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE; 2012. p. 25–35.

- Yan, M., Zhang, X., Yang, D., Xu, L., Kymer, J.D.. A component recommender for bug reports using discriminative probability latent semantic analysis. *Information and Software Technology* 2016;73:37–51.
- Yang, G., Zhang, T., Lee, B.. Utilizing a multi-developer network-based developer recommendation algorithm to fix bugs effectively. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM; 2014. p. 1134–1139.
- Yu, Y., Wang, H., Yin, G., Wang, T.. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology* 2016;74:204–218.
- Zanjani, M.B.. Effective assignment and assistance to software developers and reviewers. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM; 2016. p. 1091–1093.
- Zanjani, M.B., Kagdi, H., Bird, C.. Using developer-interaction trails to triage change requests. In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press; 2015. p. 88–98.
- Zhang, H., Gong, L., Versteeg, S.. Predicting bug-fixing time: an empirical study of commercial software projects. In: *Proceedings of the 2013 international conference on software engineering*. IEEE Press; 2013. p. 1042–1051.
- Zhang, M., Hurley, N.. Avoiding monotony: improving the diversity of recommendation lists. In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM; 2008. p. 123–130.
- Zhang, T., Chen, J., Jiang, H., Luo, X., Xia, X.. Bug report enrichment with application of automated fixer recommendation. In: *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press; 2017a. p. 230–240.
- Zhang, T., Chen, J., Yang, G., Lee, B., Luo, X.. Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software* 2016a;117:166–184.
- Zhang, T., Lee, B.. An automated bug triage approach: A concept profile and social network based developer recommendation. In: *International Conference on Intelligent Computing*. Springer; 2012. p. 505–512.
- Zhang, T., Lee, B.. A hybrid bug triage algorithm for developer recommendation. In: *Proceedings of the 28th annual ACM symposium on applied computing*. ACM; 2013. p. 1088–1094.
- Zhang, W., Wang, S., Wang, Q.. Baha: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis. *Chinese Journal of Electronics* 2016b;25(6):1011–1018.
- Zhang, W., Wang, S., Wang, Q.. Ksap: An approach to bug report assignment using knn search and heterogeneous proximity. *Information and Software Technology* 2016c;70:68–84.
- Zhang, X., Wang, T., Yin, G., Yang, C., Yu, Y., Wang, H.. Devrec: A developer recommendation system for open source repositories. In: *International Conference on Software Reuse*. Springer; 2017b. p. 3–11.
- Zhou, Y., Tong, Y., Gu, R., Gall, H.. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 2016;28(3):150–176.

## Appendix A Methods and techniques used by different studies covered by our survey

Table 6 shows the papers in our survey and the methods they used. We categorized the methods in six groups; First, different Machine Learning (ML) classifiers as well as Stacked Generalization (which combines several classifiers) are categorized in ML. Then, we put the general Information retrieval (IR) methods that consider the notation of query-document for BA problem as well as IR-based methods like Topic Modeling and Cosine similarity into IR category. The third category is Natural Language Processing (NLP) including the general NLP techniques as well as Information Extraction (IE) methods. Then, methods based on Markov chains (in which a state diagram determines the probability of events), Tossing Graphs (in which a network of developers represent the probability of substitute assignees) and Social Network Analysis (SNA) are represented in Network and Graph based category. The Statistical models including smoothed Unigram Model (in which each bug is represented as an n-dimensional probability vector for the n terms available in the corpus, and the probabilities are smoothed based on their occurrence in the whole corpus) and Kullback–Leibler (KL) Divergence (as a measure of difference between two probability distributions) are shown in the Statistical Models category. Finally, other methods like Fuzzy, Bug Localization (in which the related files to the new bug is estimated and the developers related to those files are considered as possible assignees), Rule-based (in which rules are extracted based on meta-data of the old bugs and used for deciding about the new bugs), Collaborative Filtering (CF) (substituting developers and files –or bug reports– instead of users and items in the regular usages of CF) and Genetic Algorithm (in which different combinations are built and tested adaptively, representing different assignment scenarios and objectives) are categorized in the last category.

Table 6: Details of the methods and techniques used in the literature

Study	Machine Learning (ML)						Information Retrieval (IR)						Natural Language Processing (NLP)		Network and Graph based			Statistical models				Other methods								
	Naïve Bayes (NB)	Bayesian Network (BN)	Support Vector Machine (SVM)	C4.5	K-Nearest Neighbor (KNN)	Convolutud Neural Network (CNN)	Stacked Generalization (SG) ensemble learner	IR (general)	Latent Semantic Indexing (LSI)	Latent Dirichlet Allocation (LDA)	Topic Modeling (general)	Vector Space Model (VSM)	Term Frequency – Inverse Document Frequency (TF-IDF)	Cosine similarity	NLP (general)	IE (Information Extraction)	Social Network Analysis (SNA)	Tossing Graph (TG)	Markov chains	smoothed Unigram Model (UM)	Kullback–Leibler (KL)	Divergence	Developers' vocabulary profile (as expertise matrix)	Term selection / Term weighting	Fuzzy	Bug localization	Rule based	Collaborative Filtering (CF)	Genetic Algorithm (GA)	Others
(Ćubranić and Murphy, 2004)	✓																													
(Anvik, 2006)			✓																											
(Canfora and Cerulo, 2006)								✓															✓							
(Anvik et al., 2006)			✓																											
(Kagdi et al., 2008)																										✓				✓
(Baysal et al., 2009)												✓																		
(Lin et al., 2009)			✓										✓																	
(Jeong et al., 2009)	✓	✓																✓	✓											
(Matter et al., 2009)																							✓							
(Ahsan et al., 2009)			✓						✓				✓										✓							
(Rahman et al., 2009)																														✓
(Kagdi and Poshyvanyk, 2009)								✓	✓																	✓				
(Bhattacharya and Neamtiu, 2010)	✓	✓																✓	✓											
(Chen et al., 2010)												✓						✓	✓											
(Nasim et al., 2011)	✓	✓																												✓
(Park et al., 2011)			✓							✓																		✓		
(Wu et al., 2011)					✓								✓				✓													
(Tamrawi et al., 2011a)																							✓		✓					
(Tamrawi et al., 2011b)																							✓		✓					
(Anvik and Murphy, 2011)	✓			✓																										
(Aljarah et al., 2011)		✓																						✓						✓
(Kagdi et al., 2012)								✓	✓																	✓				
(Bhattacharya et al., 2012)	✓	✓	✓	✓														✓	✓											

continued ...



continued (Table 6) ...

Study	Machine Learning (ML)						Information Retrieval (IR)						Natural Language Processing (NLP)		Network and Graph based		Statistical models			Other methods											
	Naïve Bayes (NB)	Bayesian Network (BN)	Support Vector Machine (SVM)	C4.5	K-Nearest Neighbor (KNN)	Convolved Neural Network (CNN)	Stacked Generalization (SG) ensemble learner	IR (general)	Latent Semantic Indexing (LSI)	Latent Dirichlet Allocation (LDA)	Topic Modeling (general)	Vector Space Model (VSM)	Term Frequency – Inverse Document Frequency (TF-IDF)	Cosine similarity	NLP (general)	IE (Information Extraction)	Social Network Analysis (SNA)	Tossing Graph (TG)	Markov chains	smoothed Unigram Model (UM)	Kullback–Leibler (KL)	Divergence	Developers' vocabulary profile (as expertise matrix)	Term selection / Term weighting	Fuzzy	Bug localization	Rule based	Collaborative Filtering (CF)	Genetic Algorithm (GA)	Others	
(Shokripour et al., 2012)															✓	✓										✓					
(Xuan et al., 2012)	✓		✓														✓														
(Xie et al., 2012)										✓	✓																				
(Jain et al., 2012)																							✓								
(Zhang and Lee, 2012)											✓						✓														✓
(Jonsson et al., 2012)	✓		✓		✓		✓																								✓
(Linares-Vásquez et al., 2012)								✓	✓																		✓				
(Servant and Jones, 2012)																										✓					✓
(Rahmana et al., 2012)																													✓		✓
(Hosseini et al., 2012)	✓																														✓
(Zhang and Lee, 2013)																	✓			✓	✓										✓
(Shokripour et al., 2013)															✓									✓		✓	✓				✓
(Naguib et al., 2013)										✓	✓																				
(Kevic et al., 2013)														✓																	✓
(Jonsson, 2013)	✓	✓	✓		✓		✓																								✓
(Banitaan and Alenezi, 2013)	✓																							✓							✓
(Nguyen et al., 2014)										✓	✓																				✓
(Yang et al., 2014)																	✓			✓	✓										
(Hossen et al., 2014)									✓																	✓					
(Hu et al., 2014)												✓					✓														
(Cavalcanti et al., 2014a)									✓			✓	✓														✓				
(Borg, 2014)							✓																								✓
(Wang et al., 2014)																															✓
(Shokripour et al., 2014)															✓									✓							

continued ...

continued (Table 6) ...

Study	Machine Learning (ML)					Information Retrieval (IR)					Natural Language Processing (NLP)		Network and Graph based		Statistical models			Other methods												
	Naïve Bayes (NB)	Bayesian Network (BN)	Support Vector Machine (SVM)	C4.5	K-Nearest Neighbor (KNN)	Convolved Neural Network (CNN)	Stacked Generalization (SG) ensemble learner	IR (general)	Latent Semantic Indexing (LSI)	Latent Dirichlet Allocation (LDA)	Topic Modeling (general)	Vector Space Model (VSM)	Term Frequency – Inverse Document Frequency (TF-IDF)	Cosine similarity	NLP (general)	IE (Information Extraction)	Social Network Analysis (SNA)	Tossing Graph (TG)	Markov chains	smoothed Unigram Model (UM)	Kullback–Leibler (KL)	Divergence	Developers' vocabulary profile (as expertise matrix)	Term selection / Term weighting	Fuzzy	Bug localization	Rule based	Collaborative Filtering (CF)	Genetic Algorithm (GA)	Others
(Shokripour et al., 2015)													✓		✓															
(Sharma et al., 2015)															✓													✓		✓
(Sajedi et al., 2015)																								✓						✓
(Zanjani et al., 2015)					✓								✓	✓											✓					
(Jain and Wilson, 2016)	✓																													✓
(Dedík and Rossi, 2016)	✓		✓										✓																	
(Zhang et al., 2016b)										✓	✓									✓										
(Zhang et al., 2016c)					✓								✓							✓										
(Tian et al., 2016)													✓	✓													✓			✓
(Jonsson et al., 2016)	✓		✓			✓							✓														✓			✓
(Zhang et al., 2016a)					✓					✓	✓		✓		✓				✓											✓
(Cavalcanti et al., 2016)			✓						✓				✓															✓		
(Anjali et al., 2016)																							✓	✓						
(Zanjani, 2016)					✓																						✓			
(Khatun and Sakib, 2016)													✓																	✓
(Sajedi et al., 2016)																								✓						✓
(Park et al., 2016)			✓							✓	✓											✓						✓		✓
(Liu et al., 2016)											✓																			✓
(Karim et al., 2016)																													✓	✓
(Lee et al., 2017)					✓																									✓
(Zhang et al., 2017a)										✓	✓				✓															
(Xia et al., 2017)										✓	✓																			✓
(Goyal, 2017)	✓																							✓						✓
(Florea et al., 2017a)			✓		✓																									✓
(Florea et al., 2017b)			✓							✓			✓		✓															
(Khalil et al., 2017)																													✓	✓
(Sun et al., 2017)										✓	✓				✓				✓								✓	✓		