Received: 7 August 2018 | Revised: 7 December 2019 | Accepted: 7 January 2020
DOI: 10.1002/smr.2250

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE - STANDARDS

Guidelines for evaluating bug-assignment research

Ali Sajedi-Badashian | Eleni Stroulia

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

Correspondence
Ali Sajedi-Badashian, 2-09 Athabasca Hall, Edmonton, AB, Canada. T6G 2E8.
Email: alisajedi@ualberta.ca

Funding information
Alberta Innovates - Technology Futures, Grant/Award Number: Graduate Student Scholarship; Faculty of Graduate Studies and Research, University of Alberta, Grant/Award Number: Queen Elizabeth II Graduate Scholarship

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 we collected from 13 long-term open-source projects, using a simple Tf-IDf similarity metric. On the basis of our arguments and/or experiments, we provide useful guidelines for performing further bug-assignment research. We conclude that mean average precision (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, research evaluation, research reproducibility, software engineering

1 | INTRODUCTION

Maintenance of software projects is a complicated task, including handling of bug reports, feature requests, and crash reports. In big projects, hundreds of bugs are reported every day. After receiving the bug reports, one of the main triaging steps is to assign them to appropriate developers. This needs a lot of time and effort from the project management team, which is too expensive. Hence, the researchers devised semi-automatic methods—called bug assignment—to assign bugs to developers based on their expertise and previous bug-fixing history. 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.[1-8] Despite the considerable attention of researchers, BA is still an expensive and time-consuming task in software projects.[9] Between 50% and 90% of software development cost is regarding maintenance,[10,11] and an important task in software maintenance is dealing with bugs. Large projects receive hundreds of bug reports daily,[12] 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.[10] In ArgoUML and PostgreSQL, the median time-to-fix for bugs in some projects is around 200 days.[13] In Eclipse, 24% of the bugs are reassigned to another developer, before getting fixed.[14] Even the fixed bugs, in both Eclipse and Mozilla, have been reassigned at least once in more than 90% of the cases.[10,15]

This indicates the need for more accurate BA methods for big projects[16,17] 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, semiautomated. Despite the long record of related research in the field, it is hard to apply the proposed BA research in practice.[18] Even if it is a question whether the previously introduced BA methods can be applied in large proprietary projects with acceptable performance or not.[15] Still, the industry needs more practical methods, with higher standards. [10]

A study can be useful in industry if it undergoes through a realistic validation framework, and, is reproducible. The term "reproducible research," introduced by Fomel and Claerbout, tries to infuse standards for publications in computing science.[19] 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.[19,20] In a *Science* paper, Peng mentioned the potential of serving as "*a minimum standard for judging scientific claims*" an important characteristic of reproducibility.[21] He indicated the need for *data*, *metadata*, 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. Finally, according to Juristo and Moreno,[22] to make the software engineering experiments repeatable, one should use objective metrics and clearly describe the qualitative factors.

Regarding BA research, one study may depend on very detailed data or sophisticated metadata that are rarely accessible. Or it may pose biased conditions or filtering (eg, 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" (who the best developer for a given bug really is) against which to validate BA methods (ie, the golden standard). The broader the definition of the 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 considered, the more difficult the BA task becomes.

In addition, there is no consensus in evaluating 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 critically reproducible problems. Moreover, 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 effectiveness 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"?
   Who would be the best available assignee? What is the best "golden standard" (ground-truth assignee) 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 and limiting the experiment bias the results?

Answering the above questions helps to establish a baseline for fair comparison of the results in further BA research. To answer the above questions, we have curated an extensive data set, including bug reports from 13 open-source projects, their metadata 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

* If we adopt a too broad definition of real assignee that includes most developers in the project, then the prediction will be irrationally easy. For example, in a project with 50 developers, if one assumes 30 developers as real assignees on average for each bug, then even a random prediction will result in 60% accuracy at the first hit, which is unreasonably high.

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 describes the experiment setup. In Section 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 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,[23] Budgen and Brereton,[24] and Weidt and Silva.[25]

The first goal is to select all 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 papers (based on some experimental design requirements that we will define) and discuss their methods more in-depth.

We performed a publication search using four research databases: ACM,[†] IEEE,[‡] Sciencedirect,[§] and Springer.[¶] We searched for four search strings in each of the four databases: *bug assignment*, *"bug assignment,"bug triaging*, and *"bug triaging."* For each search, we captured the top 100[#] returned results (which were usually returned in the first four or five pages). We reviewed all the 100 results of each search and considered them in our survey if they were introducing a BA method. The selection criteria were as follows:

2.1.1 | 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.

2.1.2 | 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 the 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,[26,27] applying BA methods in code reviewer recommendation,[28] bug fix time prediction,[29,30] tool-development for BA with no reported accuracy,[31] and bug-triaging[‖] related studies other than BA, including bug report deduplication,[32,33] bug report classification, [34] bug localization,[35-37] component-assignment (recommending a component for a given bug report),[38,39] and investigating problems of assignment and reassignment of bugs.[**][40,41]

[†] https://dl.acm.org/dl.cfm
[‡] http://ieeexplore.ieee.org/Xplore/home.jsp
[§] https://www.sciencedirect.com/
[¶] https://link.springer.com/
[#] We defined the threshold as high as 100 to make sure that we do not miss any important paper. We observed that most of those related papers were addressed in the first or second page of the retrieved results. In the last page leading to the 100th result, there were no related study in either of the databases.
[‖] Bug triage is the overall process of screening, studying and deciding about the received bug reports. It includes reviewing all the received bug reports, determining their severity and priority, detecting and removing the duplicate bug reports, appointing a developer as assignee and so on. We filtered those studies to control the broadness of the topic. Note that BA is one of the problems (with its own formulation and solutions) in the bigger domain, bug triaging.
[**] These are qualitative studies addressing some specific aspects of the problem rather than providing a new BA methodology.

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, 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 A1).

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"[2,42-45]; 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.

Jonsson et al[46,47] introduced the problem of "team bug-assignment" in which bugs are redirected to one of several available teams (instead of a single developer). 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 nonoverlapping, 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.[6,48,49] For example, Nguyen et al[6] 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, ie, expertise maximization and time-to-fix minimization.[7]

Finally, Karim et al[50] proposed a more pragmatic multi-objective problem formulation, assigning multiple bugs to several developers at the same time. Their method assumes the competency of developers in different areas (packages) as a function of the 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 subprojects 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,[51] Hosseini et al,[52] and Khalil et al[53] 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 A1 in Appendix **?**). Older methods 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 methods depends on the number of previously fixed bugs, and, as a result, they may perform poorly in small projects.[54]

Then other methods like fuzzy[55] and statistical[3] approaches appeared. These methods utilized an expertise perspective for the developers regarding keywords. On the basis of 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 metadata information to relate developers to the new bugs. Examples are location-based techniques.[5,56,57] They first predict or find the location of bugs (ie, 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 (eg, 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 (eg, a few bug reports up to a few hundreds). Also, social network[48,58] and tossing graph[1,2] approaches appeared, which target more complicated interactions between development objects (eg, developers, bugs, commits, or a combination of them).

The most prevalent methods were information retrieval (IR) and then machine learning (ML). Many of the recent studies focused on IR-based activity profiling since it usually leads to higher accuracies.[5,59] In the recent years, most of the studies use at least one IR method.

In recent years, some of the studies combined different methods (eg, combining machine learning and tossing graphs[2] or combining KNN and IR methods[17,60]). Most recently, the researchers show a tendency toward *social* point of view. For example, some previous research[16,17,43] built 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 cast by the community.[61] In that study, having a bug report at hand, we extracted a score for each developer based on their stack overflow activity regarding the keywords in that bug report.

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 the 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 to those 74 papers:

3.1 | 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 and report the results based on evaluation metrics (eg, instead of just theoretic ideas);
3. papers should report final results on BA effectiveness (eg, instead of a comparison of tuning values or data sets);
4. papers must experiment on full data of developers and bugs (ie, have no major filtering on "experiment data"); and
5. experiments must contain relatively high number of developers (approximately 20) and bugs (approximately 500) in at least one project to make the experiment more realistic.

3.2 | Exclusion criteria

1. Papers that their technique relies on external sources (eg, 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,[50,51,53] team BA,[46,47] market-based bug allocation,[52] component-level BA,[62] time/cost enhancement in BA,[6,48,49] and bug report enrichment[63] with application in BA. Many of those studies fail to meet the first inclusion criterion, by considering a varied scenario (eg, bug report enrichment) instead of the prevalent formulation. Also, we eliminated our previous paper[61] 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).[64,65]

After considering the above criteria, we obtained 13 papers.[††] These are great BA examples regarding evaluation, reproducibility, and further comparisons. We will focus on these studies in the rest of our paper as "selected BA studies." Note that meta-analysis would help in the comparison of different studies, which would be very useful for further research. However, for several reasons, in the current state of BA research, it is not applicable to perform meta-analysis research on the BA studies.[66-71] First, different studies are measuring different things (ie, different methods) and not the same thing or factor. Second, the experiment conditions like sample sizes differ. Finally, most previous research did not report all the details like standard deviation of their reported results. These factors led us to the fact that at the current state, meta-analysis is not applicable and we just perform our survey by summarizing the techniques, knowledge assumptions, and used metrics in the selected BA studies in more detail.

3.3 | Review of the selected research

Čubranić and Murphy[72] 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 (eg, 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[73] used an IR method 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[1] 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 the 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[42] used a vector space model (VSM) to consider the source code contributions and 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

[††]Three studies[2,55,74] are conference version of other studies by the same authors.[10,75,76] So, we merged them into their journal version in the tables of this paper.
[‡‡]If a bug report is assigned to developer A, reassigning it to another developer (because of any reason like lack of expertise of A regarding that bug report or his high workload) is called bug tossing.

TABLE 1 A review of the used information for bug assignment in selected bug assignment (BA) studies

Method	Bugs' Info		Developers' Expertise Info				
	Title/Desc	Meta	Bug Fixing; Title/Desc	Being Committer	Tossing History	Meta	Changed Code
Čubranić and Murphy[72]	✓		✓				
Canfora and Cerulo[73]	✓		✓				
Jeong et al[1]	✓		✓		✓		
Matter et al[42]	✓		✓	✓			✓
Tamrawi et al[55,75]	✓		✓				
Bhattacharya et al[2,10]	✓	✓	✓		✓	✓	
Shokripour et al[54]	✓		✓	✓			
Zhang et al[16]	✓	✓	✓	✓		✓	
Cavalcanti et al[74]							
Cavalcanti et al[76]	✓	✓	✓			✓	
Sun et al[77]	✓	✓		✓		✓	✓

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[55,75] introduced a *fuzzy* method toward bug assignment, which computes a score for each pair of "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[75] 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 to each developer. Then, when calculating the fuzzy score for each developer, it just considers those selected terms and ignores other terms.

Bhattacharya et al[2,10] developed a *tossing-graph*–based method, similar to the work done by Jeong et al,[1] which also adds labels to 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 tossee 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 (tossee) and enhance this list to a top 5 recommendation.

Shokripour et al[54] combined *bug report localization*, *extraction (IE)*, and *natural language processing (NLP)* to predict the assignees for the bug reports. Their method first applies IE and NLP techniques to file-related components (eg, 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 (ie, 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 based on historical data.

Zhang et al[16] proposed a method that builds a *heterogeneous social network* of developers and 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 (k-nearest neighbors) classification, *Cosine similarity*, and *tf-idf* (as two IR-based similarity metrics). 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[74,76] introduced a semi-automatic method that combines a *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 metadata (eg, 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 the 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 (eg, the developers who fixed similar bug reports are most likely to fix the new one).

Finally, Sun et al[77] 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 the 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)* (a technique that combines probabilistic topic modeling with traditional collaborative filtering) to give a score to each of those developers (on the basis of shared keywords in their relevant source code and the new issue) and sort and recommend them to fix the new issue.

3.4 | Knowledge assumptions of bug-assignment methods

A wide range of information has 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 methods[72,73,75] mostly rely on textual information (eg, title and description) of the bug reports as clues for indication of expertise of developers and matching with new bug reports. Many of the newer methods[10,16,76] used variety of metadata fields (eg, 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[77] considered commit messages and some metadata. In addition, their method extracts and deals with source code and file changes. This sometimes makes a huge demand on the needed data and processing (eg, processing Giga bytes of source code, html files and documentations for different branches of each project, on several projects).

Extra information in the metadata can help obtain higher accuracies in some cases, but still, text-based methods are the most effective techniques used[44,78] and those metadata-based approaches are sometimes difficult to setup; for example, some need to know who maintained different pieces of code in the IDE[56] or who interacted online with whom in different setups.[16] As a result, the textual elements remain the most prevalent information used for BA.

3.5 | 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 methods against real bug data. The metrics they used are *top-k* accuracy, *precision @k*, *recall @k* (all with k = 1, 5, and 10), *mean reciprocal rank (MRR)*, and *mean average precision (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 only used once in previous BA research by Zhang et al.[16] 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 m et all the inclusion criteria we set for reproducible research (eg, the projects and the number of bugs they tested on are big enough and they ran their experiment on all or nearly all bugs in their data set and not a small subset of them), the comparison of the results of new research against these 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 used in a research paper can affect the outcome (eg, accuracy) of its 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 present in the 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 the harmonic mean of precision and recall. In fact, it combines precision and recall into one metric.[79,80]

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"?

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 the 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.

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.

   It is desired to boil down several evaluation metrics into one.[79,81] This supports easier comparison of a study against other approaches. Unlike precision and recall, which should be reported and interpreted together, MAP, MRR, *F* measure, and *top-k* accuracy are single-figure metrics.[82]

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 bounded 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. [16,82,83]
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.[82,84,85] 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.[83,86]
4. The evaluation measure should be robust to the number of real assignees.

TABLE 2 The evaluation metrics and other design choices of the selected research

Method / Project	#devs	#bugs	Top1	Top5	Top10	p@1 / r@1	p@5 / r@5	p@10 / r@10	MRR	MAP
Čubranić and Murphy[72]										
Eclipse	162	15 859	30.00							
Canfora and Cerulo[73]										
Mozilla	637	12 447				- / 12.0	- / 21.0	- / 24.0		
KDE	373	14 396				- / 5.0	- / 10.0	- / 12.0		
Jeong et al[1]										
Eclipse	?	46 426		77.14						
Mozilla	?	84 559		70.82						
Matter et al[42]										
Eclipse	210	130 769				33.6 / 30	20 / 60	10 / 71.0		
Tamrawi et al[55,75]										
Firefox	3014	188 139	32.1	73.9						
Eclipse	2144	177 637	42.6	80.1						
Apache	1695	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 et al[2,10]										
Mozilla	?	549 962	27.67	77.87						
Eclipse	?	306 297	32.36	77.43						
Shokripour et al[54]										
Eclipse	?	?				- / 30	- / 70			
Mozilla	?	?				- / 30	- / 50			
Gnome	?	?				- / 10	- / 45			
Zhang et al[16]										
Mozilla	870	74 100							0.28	44.49
Eclipse	540	42 560							0.28	56.42
Ant	200	763							0.35	36.48
TomCat6	80	489							0.35	36.54
Cavalcanti et al[74,76]										
New SIAFI - A	70	781	31.40							
New SIAFI - B	70	1031	22.00							
Sun et al[77]										
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					

Note. 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 (eg, 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). Abbreviations: MAP, mean average precision; MRR, mean reciprocal rank.

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 with 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.[82]

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.[87] It is a good performance metric when a short list of items is provided to the user (ie, the triager).[87,88] 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[79] and other software engineering problems like bug report deduplication.[89,90] Interestingly, it is rarely used for BA evaluation (as we mentioned before, the only study out of all the 74 papers of our survey that reported MAP was done by Zhang et al[16]). As a result, although reporting MAP would be convincing enough, for comparison with previous methods (eg, meta-analysis[91]), it is still recommended to report the other well-known metrics as well (*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).

4.1 | More detailed elaboration on MAP

Let us have more elaboration with illustration on MAP for BA. Figure 1 shows an example of MAP for three assignments. The four criteria for evaluation metrics that has been mentioned earlier in this section are evident from this figure. For example, it highlights higher ranks by calculating precision at several points (eg, @1, 4, 6, 11, and 12 for the first bug in Figure 1) at which point, the previous correct recommendations are taken into account.

As an example, explaining the superiority of MAP over other metrics, consider the three bug reports in Figure 1. The *top-5* and *top-10* accuracies both show 100% accuracy (because there is at least one real assignee in the first five and 10 ranks in all the three bug reports). But these two metrics fail to reflect the goodness of the used method precisely; the captured real assignees are at different ranks for the three bug reports but they are treated the same. Also, in some cases, there are other real assignees out of the first five and 10 recommendations, but these metrics give us 100% accuracy without considering those real assignees.

On the basis of these assumptions, and deriving from previous IR notations,[79,87,88,92] having *n* bug reports, we reformulate the following definition of MAP considering BA notations:

$$
\begin{aligned}
&MAP = \frac{\sum_{i=1}^{n} AP_i}{n} \\
&AP_i = \frac{\sum_{k \in R_i} p@k}{\#\ \ of\ \ real\ \ assignees\ \ for\ \ bug\ \ i} \\
&R_i : \{ranks\ \ of\ \ real\ \ assignees\ \ in\ \ the\ \ recommended\ \ list\ \ of\ \ developers\ \ for\ \ bug\ \ i\} \\
&p@k = \frac{\#\ \ of\ \ real\ \ assignees\ \ in\ \ the\ \ top\ \ k\ \ recommended\ \ developers}{k}
\end{aligned}
\tag{1}
$$

AP_i is the average precision for bug *i*, and R_i is the set of all the ranks of the real assignees in a recommended ranking for bug *i*. Note that when there is exactly one assignee per bug over all the bug reports, MAP becomes equal to MRR.

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 the "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 of that bug. It shows interactions (eg, comments, being assigned, or closed) about the bug and metadata 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, 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.

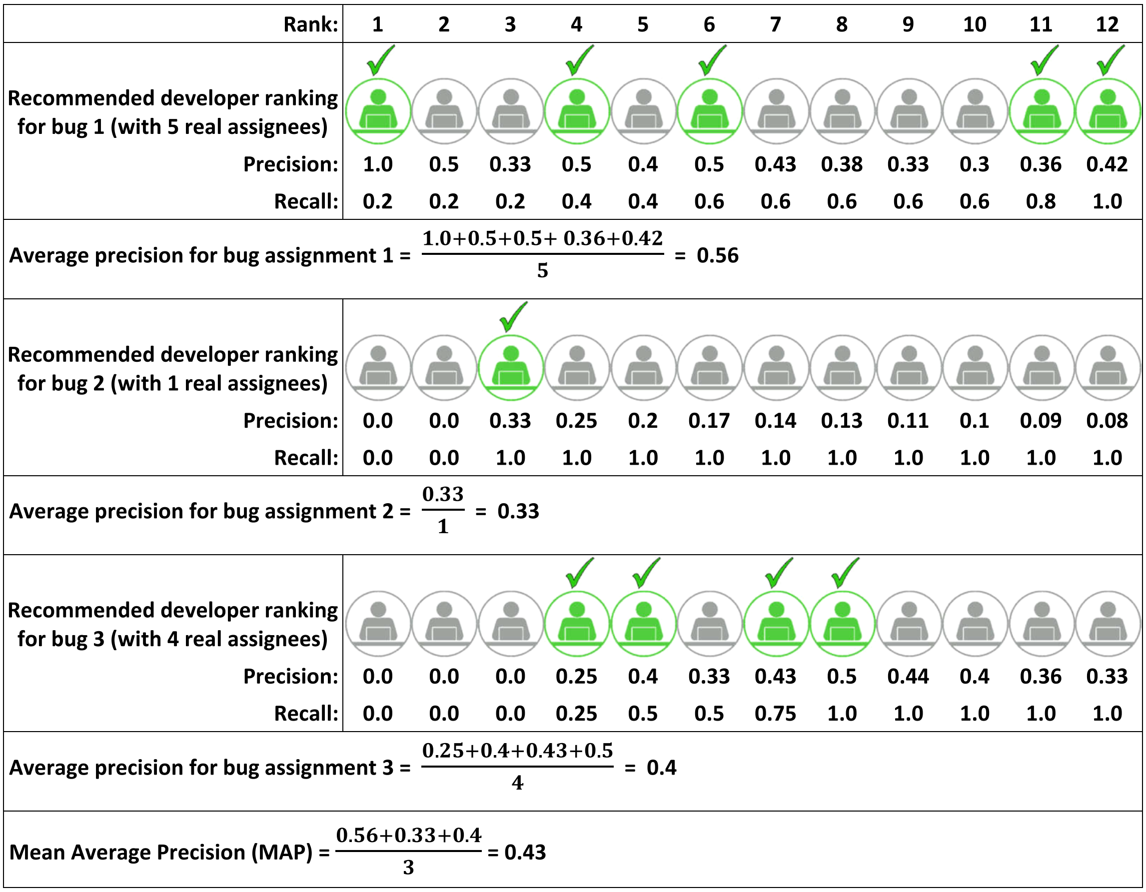

FIGURE 1 An example of use of mean average precision (MAP) for bug assignment over three bugs

In almost all the previous research, which proposed a new BA method, 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 method and report it using some metrics. The definition of 'real assignee' (ground-truth assignee) is critical in validating and understanding the effectiveness of a method and comparatively analyzing the merits and shortcomings of alternative methods.

Consider the following example as the effect of this golden standard on the evaluation results. Anvik et al[93] adopted some heuristics to assume developers as assignees of a bug. On the basis of 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 (even till now, after more than 10 years) regarding the precision @1 metric. One reason for 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. On the basis o f our observations in almost all previous studies, each bug on average has only one or 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 results. Without a clear definition of *real assignee*, fair judgment or reproduction of 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 (eg, 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

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

Method	Assignee types	Developer community
Čubranić and Murphy[72]	T3, T4	The real assignees of the selected bugs for the experiment (162 developers) are considered members of "developer community."
Canfora and Cerulo[73]	T1, T2	The real assignees of the selected bugs for the experiment (373 and 637 developers in two projects)
Jeong et al[1]	T4	The real assignees of the selected bugs for the experiment (number of developers is not mentioned)
Matter et al[42]	T1, T2, T4	The real assignees of all the bug reports (210 developers)
Tamrawi et al[55,75]	T4	The real assignees of the selected bugs for the experiment (between 156 and 3014 developers in seven projects)
Bhattacharya et al[2,10]	T3	The real assignees of all the bug reports (number of developers is not mentioned)
Shokripour et al[54]	T1, T2	The real assignees of the selected bugs for the experiment (number of developers is not mentioned)
Zhang et al[16]	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[74,76]	T1, T2	The real assignees of the selected bugs for the experiment (70 developers)
Sun et al[77]	T3, T4	The real assignees of all the bug reports (between 47 and 667 developers in five projects)

[a]the type of assignee in this paper was not directly mentioned in the text, but it is conceived indirectly from the text

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 cannot fix it, then they may opt out or may be un-assigned. A developer may also 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.

Most previous research used the definitions of T1, T2,[§§] T3, and T4 or a limited combination of them for the real assignee. No previous research, however, used the comprehensive combination of them, ie, 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 (eg, 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 (because of high workload, unavailability, or other reasons), but still would be included in the gold set. Although this definition would be the ideal golden standard, 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 (eg, T1 to T5) as we discussed above. These five types of assignee can be extracted from the issue-tracking system directly and easily.

[§§]In some of the previous research, it is not clear if they are using just T1 or a combination of T1 and T2. Because of lack of a clear separation in these cases, we assumed they considered both T1 and T2 in Table 3

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 standard" 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 for the project.[94] 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 (ie, 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.[45,61,95] This is the most comprehensive definition for developer community. Some others took a subset of the developers in the project (eg, the committers) as the developer community.[56,96] Many others considered the set of previous assignees as developer community.[10,72,75,93] Also, there are many others that did not have a clear definition for it.[97]

Regardless of the general definition for developer community, many researchers ran their experiments on a subset of the bugs—ie, filtered the community to remove less-active developers, which also removes the subtle bugs assigned to them. This is another restriction on the developer community in the data set. We discuss it later along with the effect of developer community on the evaluation of the results.

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 are the documents. In order to prepare the documents, at each point in time, we concatenated text of all the previously fixed bug reports by each developer as a single document representing that developer. 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 the main language of a project as any programming language that contains at least 15% of the lines of code of that project.

We use *tf-idf*,[79,98] an IR term weighting scheme for matching the documents against the given query. It is used to emphasize specific keywords and de-emphasize the common terms, and shown to work well on textual data.[16,17,41,44,45,99] *Tf-idf* gives an "overlap score"[79] to a document regarding the query. Having several documents (developer profiles), the document with the highest score is considered the document most similar (and most relevant as a response) to the given query (bug report). We chose *tf-idf* since it is simply based on textual information of bug reports (rather than additional metadata or source code file information which might engage other potential factors) and makes the evaluation more reliable. Note that this method or its variations were widely used previously—more than all other IR-based measures like cosine similarity—in the BA studies (see Table A1 in the Appendix for details).

Having a query (bug) "*q*," we use the following equation to calculate the score for document (developer) "*d*" in the corpus "*D*":

$$score(q,\ d) = \sum_{t \in q} tf(t,\ d) \cdot idf(t, D). \tag{2}$$

This formula has two main components: *tf* (term frequency) measures number of times a term *t* appears in document *d*, normalized by document length:

$$tf(t,\ d) = \frac{\#\ \ of\ \ \ times\ t\ is\ mentioned\ \ \ in\ d}{total\ \ \#\ \ of\ \ \ terms\ \ \ in\ \ \ d}. \tag{3}$$

On the other hand, the inverse document frequency (*idf*) measures the importance of a term with regard to all documents in the corpus *D*[79]:

$$idf(t, D) = \log[10] \frac{total\ \ \ \#\ of\ documents\ in\ the\ corpus}{\#\ \ of\ \ \ documents\ \ \ containing\ \ \ t}. \tag{4}$$

[¶¶] Github organizations are shared accounts that are used for businesses. Organizations allow collaboration of many developers at many projects at once. Access to the projects in an organization is managed by its administrators in Github.

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

Project	#of Members in Developer Community (DC)	#of Members 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	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	3578	1371	113	3713	945	6142
Fog	770	208	1124	91	27	1146	63	1327
Julia	831	541	9086	1759	54	9590	1345	12 748
Brackets	864	646	6255	171	6	6554	3731	10 462
Travis-ci	1159	1096	5473	13	2	5716	603	6334
Elasticsearch	1262	758	10 423	2192	498	10 362	3132	16 184
Salt	2283	1227	10 237	2344	233	10 682	2274	15 533
Angular.js	2386	1069	7402	503	196	7671	1288	9658
Rails	4079	1431	8794	857	138	9366	944	11 305
Total (all projects)	Average: 1132 Median: 831	Average: 583 Median: 541	64 474	9661	1374	66 858	15 123	93 016

We use Equation (2) to evaluate the similarity of a bug and a developer and produce a score for each developer. 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-validated our recommendations against the real assignees in our data set to recommend a ranked list of developers. We use leave-one-out (LOO) cross validation[100] as an enhanced version of *K*-fold cross-validation (and a logical extreme case of it). In this model, at each point, all the previous instances (ie, bug reports) are considered as training set and a single new instance (ie, bug report) is considered a test set. Then, this process continues for the next instances (ie, bug reports). So, the process is online, in which expertise information from previous *n* bug reports are considered to conclude about the n + 1th bug report. Note that most previous BA research adopted this method or similar validation methods of the same family (ie, holdout or *K*-fold cross validation).[1,2,16,42,54,55,61,76,95] On the basis of 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.[##]

6.1 | The data set

In our previous research, we studied "several Github projects,"[61] chosen because of 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[‖‖]. The other seven projects stopped publishing their issues (eg, switched to private issue tracking systems).

We extracted data of those 13 projects from their beginning (April 28, 2009 or later, based on project start dates) to October 31, 2016. 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,[61] we 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.[***]

Table 4 shows the 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 that the assignee prediction on this data set is not trivial. We made the data set available online[##] for further BA research.

In order to capture the 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 (ie, T1 and T2), we needed to check the commit messages. According to Github documentaion,[101] the fixers reference the

[##]The data set, source code, documentations, and detailed output results are available at: https://github.com/TaskAssignment/MSBA-outline

[‖‖]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

[***]https://github.com/TaskAssignment/software-expertise

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)[†††]:

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

We cross-referenced this with issue events[‡‡‡] (which were also extracted by our code using Github APIs) 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—eg, 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 (ie, 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.[##]

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 standard 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*. 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 MAP results of the *tf-idf* method on them and perform some statistical analyses. We compare the average, variance, and existence of outliers regarding the MAP results based on each efinitions of real assignee. As we mentioned in Section 6, we chose *tf-idf*—because of its wide usage in previous BA studies—to eliminate potential confounding variables. The ultimate goal is to see and interpret how the notion of "real assignee" can affect MAP and argue toward the best choice of "real assignee."

Note that to avoid complexity, in this step, we do not do any filtering or processing on the developer community. We just consider "all the project members" as the developer community. In the next subsection, we will provide reasoning for this choice and will show that this is actually the most reliable option.

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. To calculate this overall MAP, we consider all the bug reports in all projects as bug reports of a single big project and then calculate the overall MAP. The overall MAP over all projects is like a weighted average over all projects taking into account number of bug-assignments of each project (as weight) in the average. The overall MAP starts from 34% for T1 and goes to 46% for T2 (around 35% 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. Figure 2 reports the distribution of project-based MAP values, using different assignee types over 13 projects; T2 exhibits the highest variance over different projects. Also, T1, T3, and T4 have outliers (see the small circles in the diagram), 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

[†††] 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).
[‡‡‡] Issue events include every interaction about bugs including opening, closing, referencing, subscribing, assigning and reopening.

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

TABLE 5 MAP for 13 projects using T1 to T5 as "assignee type"

can say that T5 gives the results in a more acceptable and robust range (without much tolerance or skewness over different projects). Note that in Table 4, some projects have only a small number of BA (eg, less than 50) regarding T1 or T2 and have very small MAP (eg, 1.82 for Html5rocks). Since the the overall MAP is like a weighted average and number of BA in those projects (eg, Html5rocks) is not high, those small MAP values do not have high effect on the overall variance. To verify that the high variance of T2 is not specific to those projects, we excluded those projects and reanalyzed 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 golden standards 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.

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 example in Section 5.1 based on what Anvik et al[93] showed in their research (ie, 30 assignees per bug report as opposed to approximately 1.4 assignees in our broadest definition, T5). 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 method 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.

Finally, we believe narrowed types of assignee just capture a limited type of work towards fixing bugs. 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 ground-truth assignee measures how good the proposed method is regarding that specific aspect. The "regular developer" nature of T1 is different than the "authoritative developer" nature of T3. 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 (eg, 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. While all those narrow types are truly capturing the notion of "bug-fix," a more realistic golden standard considers all of them. Since software development is a collaborative work and many bugs are fixed by a group of developers (with different roles in the project), all the developers who perform any type of development work towards fixing a bug should be counted as proper assignees of that bug. Since the assignees of all those types have done some useful work toward fixing that bug, suggestion of any of those developers to the project manager would be some help towards fixing it. As a result, T5 covers different indications of bug-fix on which we can fairly validate a BA method.

On the basis of 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 golden standard for evaluating goodness of an assignee recommendation method.

In the next sections, we perform the rest of our analyses based on this golden standard (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,"

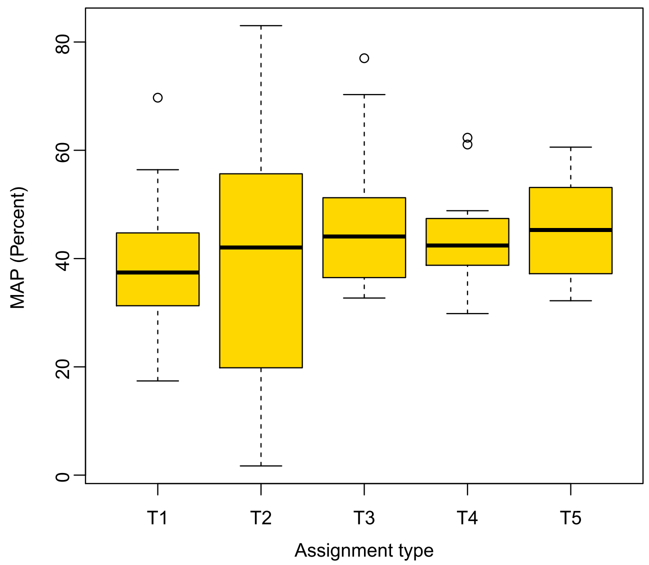

FIGURE 2 Summary of distribution of MAP over 13 projects using five different assignment types (golden standards). T5 has the lowest variance over different projects

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 limit their experiment data set and filter the members of this community in it (ie, 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 (eg, 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 method recommends somebody to fix a bug. So, restricting this set to limited subsets of project members (eg, 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.[§§§]

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 (eg, T1 to T5; narrower definition of real assignee makes the community smaller). This might itself lead to more subjective judgments. Especially some evaluation metrics (eg, 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 (ie, 66.67% top-10).

Limiting the experiment to a subset of all bugs is another adverse restriction. In our survey, we saw many examples of BA research that suffer from this limitation; 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 reduction (filtering) on the evaluation results in a high level. The studies that performed their experiment on a controlled subset of bugs, 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.

Limiting the experiment and 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.[102] They compared their method on two versions of the data set, for three open-source projects; a 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[75] reported increases in their accuracy, and,

[§§§]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

TABLE 6 The effect of limiting the experiment to a subset of all bugs (filtering less-active developers, and the bug reports they fixed) on accuracy of results; first row is the complete data set (no filtering). In each row, 5% of the bug reports are removed until the last row which includes only 20% of the data set. MAP increases to more than twice (from 40% to up to 95%) by filtering more data in each row

Anvik and Murphy[103] reported increases in their recall by filtering the less active developers. In another study, Canfora and Cerulo[73] showed that limiting the experiment and filtering the developers to 100 (out of approximately 400 to approximately 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 this *limiting* action on the accuracy of bug assignment, we ran an experiment using the baseline *tf-idf* method as we discussed before (again, we made the source code of this experiment available online [##]). 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 Table 6. Moving from top to bottom, the number of bug reports decreases (approximately 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 limitation on the experiment (ie, 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 (ie, considering the time of activity of developers as well) and enhance results after limiting the experiment. Also, note that this effect can be different (higher or lower) in another data set, based on the dynamics of that data set.

So, limiting the experiment to a subset of all bugs (ie, developer filtering) can cause to obtain dramatically higher results than would be obtained in real cases.[¶¶¶] In that case, the researcher is producing biased results with eliminating parts of data. For example, in Table 6, if the researcher filters 20% of the data and only considers 80% of it (which is shown in the fifth line of the table), he will obtain 48% MAP instead of 40%. 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.

Finally, we support our arguments by evidence from the literature. According to several previous research, [62,104,105] eliminating less-active developers reduces the diversity of developer recommendation, which results in diminishing practicality of the proposed BA method. If any bug or developer selection needs to be done, this should be performed as randomly as possible.[22,66]

[¶¶¶]Note that we are not opposed to developer filtering in general, which might help project managers in practice. They might use this "greedy" approach to sacrifice a few developers in exchange of higher accuracy. We dispute the usage of this technique for evaluation of BA methods in scientific papers.

We conclude that the size of developer community should be reported clearly. In addition, the number of members of the developer community who have ever fixed any bugs during the 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 of developer community is "all the project members". It is the most realistic option to contain any developer who works in the project and may step in to fix a given bug. Limiting 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. Much of the previous research uses small projects (or filters the data of big projects and provides 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. Indeed, we identified lack of a systematic framework to facilitate (and standardize) the evaluation and reporting of scientific claims in the BA domain, which needs more attention from the software engineering community. In order to validate a new BA method and show that it works better than the previous methods, we need to compare it against one or more previously published methods. Generally, there are three ways to do this: (a) to run the code of the previous method and test it on our data, (b) to test the code of our method on the data of the previous research, and (c) to compare the two methods considering the reported evaluation metrics (eg, 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, if we want to reimplement a previous method, it is almost impossible to take into account all the important details (note that usually the implementation details are not mentioned in the papers). Reimplementation of a generic version of their method 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 (eg, interactions with Weka), which are almost impossible to replicate. Even if the previous researchers published their code or explanation of the steps (ie, exact instructions) needed to run their method, our data might be inefficient for their method. Finally, a research might have specific steps for cleaning its data (before running the code on it), which is a critical step and usually has important effects on its results.

The second option is a better solution, since we do not judge about the previous approach except 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 method (eg, 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 (eg, the ones mentioned in Table 2). In that case, showing that a new method outperforms a bunch of previous methods (eg, by meta-analysis) is more persuasive than outperforming only one or two implementations. According to the case study research methods mentioned by Runeson et al,[106] if enough quantitative data are available from the primary studies, meta-analysis methods can be used to synthesize the evidence and make the review and comparison of the studies meaningful. In fact, confidence in meta-analysis increases when the degree of similarity between different studies increases.[106,107] However, note that there are a lot of qualitative and quantitative data and conditions in case studies. So, it might be hard to perform meta-analyses when synthesizing case study research.[106] As a result, if researchers try to measure and report the effectiveness of their BA methods using *the same metrics*, carrying out synthesis studies about those papers and performing comparisons among them becomes easier.

In either case, validation of a new method needs *fair* comparison against state-of-the-art methods. In this way, observing several points as mentioned in the next subsection 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 method 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[21] and replicating the study.[19] 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 methods that reported other metrics. Also, the standard deviation of the results is useful in conducting meta-analysis and recommended to be reported.

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 method 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 to "not limit the experiment to a subset of all bugs or developers". Filtering the data for the experiment 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. Note that in cross-validation, the data of future bugs can only be used for testing, not for training the models.

8.2 | Effects on reproducibility

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.

In terms of metrics, using MAP reduces the dependency on specific conditions of the data set. The reported number (MAP) is more generic and independent of the experiment. Also, it enables easier comparison against new research. With increasing the similarity between different research, the confidence in meta-analysis increases.[106,107] Meta-analysis allows to analyze and compare the results of a research against new research without reimplementing its algorithm or method, which enhances reproducibility. [19,20]

Capturing the comprehensive definition of "real assignee" as the ground-truth assignee provides a comprehensive golden standard to validate a method, without making bias or subjective evaluations. So, as a comprehensive golden standard, it enables higher levels of reproducibility in research.[20,21] Similarly, capturing the realistic option for developer community (i.e., all the project members) avoids imposing special conditions under which the experiment might produce biased results. Instead of those special conditions –which isolate the experiment and its usage– the rather comprehensive choice promotes reproducing the experiment with similar conditions and facilitates fair judgment and review. [21]

Finally, it is obvious that other provided recommendations regarding choosing the proper number of bug reports and developers are aimed at proper evaluation of the BA methods to understand their goodness in a realistic manner. Reporting those details in a research paper, helps other researchers or reviewers to reproduce or judge about that paper easily.[20]

8.3 | Threats to validity

There are a few points about how we obtained the results as well as generalization of the findings that we address below.

8.3.1 | Internal validity

1. To capture the real assignees (ie, the golden standard to validate our method), 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,[101] there might be some missing links[108] 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, which must be done manually.[108] To the best of our knowledge, no previous BA research has done this manual process. All the previous BA research in the field cross-validated their method against heuristic-based ground-truth assignees extracted automatically from available data of software projects.
2. One may argue that the notion of gold standard in this study is affected by project limitations. In other words, we considered the developers who eventually worked toward fixing a bug as its ground-truth assignees. Nevertheless, those developers may have been imposed to the bug because of project limitations (eg, developer workload and wages). The optima assignee would be the best developer in the team who would fix the bug (regardless of other constraints), not the real assignee.

To answer this limitation, we argue that all the captured information to support the arguments are based on real data which is in any case affected by the realistic conditions. It is very cumbersome to find the optima assignee since it needs interviewing project managers (plus the developers) to find proper developers regardless of local constraints. Furthermore, because of its high demands, to the best of our knowledge, no previous BA study has been adopting this notion of optima assignee. Hence, we discussed the case of optima assignee as a future work.

8.3.2 | External validity

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, projects and settings.

While we admit this as a threat, we argue that the experiments we established are not the only basis for our statements. Much of the reasoning of this paper is mainly based on our experience in several years of BA research, intuitions from the comprehensive survey we proposed (eg, the metrics used, and the dynamics and settings used in various BA studies, as well as their data sets), 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 (eg, by providing some examples or counter examples). Regarding the used data set, we argue that 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 BA which is nontrivial. This makes it easier to generalize the results.

9 | CONCLUSIONS AND FUTURE WORK

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 set of guidelines 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,[79] 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 method, should be inclusive of all project members.

   All in all, validating a new BA method 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 improves reproducibility of research; it 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 [##] for further research. This data set is one of the most comprehensive and recent data sets available for further BA research.

The choices of real assignee and developer community are not the only dimensions of variability in BA empirical studies. There are other factors that are worth investigating and might affect the research outcomes and reports. As an example, the real assignee of a bug (i.e., the developer who eventually worked on the bug to fix it) and the optima (i.e., the best developer who would help fixing that bug) might not be the same person. Capturing the optima assignee can be more beneficial than the real assignee. It can further enhance the evaluation. One of the further avenues of this research is to study a notion of distance between the real assignee and optima assignee. A future work in this area can quantify the error between these two. Although this is a tedious task, it can be done by approaching the real assignee (who eventually, due to project conditions worked on the bug) or the project managers and asking for more clarifications regarding that error. Understanding and finding the optima assignees of the bug reports is not easily achievable, but it can help further BA studies measure their research against even better ground-truth assignee. Another future work is to consider other limitations and conditions under which a developer is appointed as the fixer of a bug, which led us to the notion of real assignee. Examples are accessibility to developers (e.g., because of distance-related problems) as well as their workloads and wages. Considering those factors is extremely time-consuming and difficult and would result in multi-dimensional concepts but eventually can enhance the notion of golden standard and ground-thuth assignee in BA research. Finally, since all the examples and experiments of this study are based on big open-source projects, the same concepts in industrial, proprietary software can be further studied.

ACKNOWLEDGEMENTS

The work is supported by Graduate Student Scholarship[###] funded by Alberta Innovates - Technology Futures (AITF)[‖‖‖] and Queen Elizabeth II Graduate Scholarship[****] funded by Faculty of Graduate Studies and Research (FGSR)[††††] at University of Alberta.

https://fund.albertainnovates.ca/Fund/BasicResearch/GraduateStudentScholarships.aspx
‖‖‖ https://innotechalberta.ca
**** https://www.ualberta.ca/graduate-studies/awards-and-funding/scholarships/queen-elizabeth-ii
†††† https://www.ualberta.ca/graduate-studies

ORCID

Ali Sajedi-Badashian https://orcid.org/0000-0002-2556-9581
Eleni Stroulia https://orcid.org/0000-0002-8784-8236

REFERENCES

1. 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, ESEC/FSE '09ACM; 2009:111-120.
2. 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; 2010:1-10.
3. 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; 2011:12.
4. Linares-Vásquez M, Hossen K, Dang H, Kagdi H, Gethers M, Poshyvanyk D. Triaging incoming change requests: Bug or commit history, or code authorship? In: Software maintenance (icsm), 2012 28th ieee international conference on IEEE; 2012:451-460.
5. Shokripour R, Anvik J, Kasirun ZM, 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; 2013:2-11.
6. Nguyen TT, Nguyen AT, Nguyen TN. Topic-based, time-aware bug assignment. *SIGSOFT Softw Eng Notes*. February 2014;39(1):1-4. http://doi.acm.org/10.1145/2557833.2560585
7. Liu J, Tian Y, Yu X, Yang Z, Jia X, Ma C, Xu Z. A multi-source approach for bug triage. *Int J Soft Eng Knowledge Eng*. 2016;26(09n10):1593-1604.
8. Xia X, Lo D, Wang X, Zhou B. Dual analysis for recommending developers to resolve bugs. *J Soft Evol Process*. 2015;27(3):195-220.
9. Saha RK, Khurshid S, Perry DE. Understanding the triaging and fixing processes of long lived bugs. *Inform Soft Technol*. 2015;65:114-128.
10. Bhattacharya P, Neamtiu I, Shelton CR. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *J Syst Soft*. 2012;85(10):2275-2292.
11. Seacord RC, Plakosh D, Lewis GA. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Boston, MA, United States: Addison-Wesley Professional; 2003.
12. Tian Y, Wijedasa D, Lo D, Le Gouesy C. Learning to rank for bug report assignee recommendation. In: Program comprehension (icpc), 2016 ieee 24th international conference on IEEE; 2016:1-10.
13. Kim S, Whitehead Jr EJames. How long did it take to fix bugs? In: Proceedings of the 2006 international workshop on mining software repositories ACM; 2006:173-174.
14. Baysal O, Godfrey MW, 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; 2009:297-298.
15. Jonsson L, Borg M, Broman D, Sandahl K, Eldh S, Runeson P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Emp Soft Eng*. 2016;21(4):1533-1578.
16. Zhang W, Wang S, Wang Q. Ksap: An approach to bug report assignment using knn search and heterogeneous proximity. *Inform Soft Technol*. 2016;70:68-84.
17. Zhang T, Chen J, Yang G, Lee B, Luo X. Towards more accurate severity prediction and fixer recommendation of software bugs. *J Syst Soft*. 2016;117:166-184.
18. 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.
19. Fomel S, Claerbout J. Guest editors' introduction: reproducible research. *Comput Sci Eng*. 2009;11(1):5-7.
20. Schwab M, Karrenbach M, Claerbout J. Making scientific computations reproducible. *Comput Sci Eng*. 2000;2(6):61-67.
21. Peng RD. Reproducible research in computational science. *Sci*. 2011;334(6060):1226-1227.
22. Juristo N, Moreno AM. *Basics of Software Engineering Experimentation*. Madrid, Spain: Springer Science & Business Media; 2013.
23. Kitchenham B, Charters S. Guidelines for performing systematic literature reviews in software engineering. *Tech. Rep., EBSE*, Software Engineering Group, School of Computer Science and Mathematics, Keele University and Department of Computer Science, University of Durham; 2007. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471
24. Budgen D, Brereton P. Performing systematic literature reviews in software engineering. In: Proceedings of the 28th international conference on software engineering ACM; 2006:1051-1052. https://dl.acm.org/citation.cfm?id=1134500
25. Weidt F, Silva R. Systematic literature review in computer science-a practical guide. *Relatórios Técnicos do DCC/UFJF*. 2016;1:1–10.
26. 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; 2010:236-250.
27. Imtiaz S, Ikram N. Dynamics of task allocation in global software development. *J Soft Evol Process*. 2017;29(1):e1832.
28. 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? *Inf Soft Technol*. 2016;74:204-218.
29. 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; 2013:1042-1051.
30. Akbarinasaji S, Caglayan B, Bener A. Predicting bug-fixing time: a replication study using an open source software project. *J Syst Soft*. 2017;136:1–41.
31. Bortis G, Hoek Andrévander. Porchlight: a tag-based approach to bug triaging. In: Proceedings of the 2013 international conference on software engineering IEEE Press; 2013:342-351.

32. 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; 2008:461-470.
33. Banerjee S, Syed Z, Helmick J, Culp M, Ryan K, Cukic B. Automated triaging of very large bug repositories. *Information and Software Technology*. 2016;89:1-13. http://www.sciencedirect.com/science/article/pii/S0950584916301653
34. Zhou Y, Tong Y, Gu R, Gall H. Combining text mining and data mining for bug report classification. *J Soft Evol Process*. 2016;28(3):150-176.
35. Chaparro O. Improving bug reporting, duplicate detection, and localization. In: Proceedings of the 39th international conference on software engineering companion IEEE Press; 2017:421-424.
36. Wang S, Lo D. Amalgam+: composing rich information sources for accurate bug localization. *J Soft Evol Process*. 2016;28(10):921-942.
37. Sisman B, Akbar SA, Kak AC. Exploiting spatial code proximity and order for improved source code retrieval for bug localization. *J Soft Evol Process*. 2017;29(1):e1805.
38. Somasundaram K, Murphy GC. Automatic categorization of bug reports using latent dirichlet allocation. In: Proceedings of the 5th india software engineering conference ACM; 2012:125-130.
39. Yan M, Zhang X, Yang D, Xu L, Kymer JD. A component recommender for bug reports using discriminative probability latent semantic analysis. *Inf Soft Technol*. 2016;73:37-51.
40. Baysal O, Holmes R, Godfrey MW. Revisiting bug triage and resolution practices. In: Proceedings of the first international workshop on user evaluation for software engineering researchers IEEE Press; 2012:29-30.
41. Cavalcanti YC, Mota Silveira Neto PA, Machado IdC, Vale TF, Almeida ES, Meira SilvioRomerodeLemos. Challenges and opportunities for software change request repositories: a systematic mapping study. *J Soft Evol Process*. 2014;26(7):620-653.
42. 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; 2009:131-140.
43. 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; 2014:122-132.
44. Shokripour R, Anvik J, Kasirun ZM, Zamani S. A time-based approach to automatic bug report assignment. *J Syst Soft*. 2015;102:109-122.
45. 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; 2016:592-597.
46. 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; 2012:437-446.
47. Jonsson L. Increasing anomaly handling efficiency in large organizations using applied machine learning. In: Software engineering (icse), 2013 35th international conference on IEEE; 2013:1361-1364.
48. 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; 2011:139.
49. Park J-w, Lee M-W, Kim J, Hwang S-w, Kim S. Cost-aware triage ranking algorithms for bug reporting systems. *Knowledge Inform Syst*. 2016;48(3):679-705.
50. Karim MR, Ruhe G, Rahman M, Garousi V, Zimmermann T. An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *J Soft Evol Process*. 2016;28(12):1025-1060.
51. Rahmana MM, Karima MR, Ruhea G, Garousic V, Zimmermannd T. An empirical investigation of a genetic algorithm for developer's assignment to bugs. In: Proceedings of the first north american search based symposium; Elsevier, 2012:1–15.
52. Hosseini H, Nguyen R, Godfrey MW. A market-based bug allocation mechanism using predictive bug lifetimes. In: Software maintenance and reengineering (csmr), 2012 16th european conference on IEEE; 2012:149-158.
53. Khalil E, Assaf M, Sayyad AS. Human resource optimization for bug fixing: balancing short-term and long-term objectives. In: International symposium on search based software engineering Springer; 2017:124-129.
54. Shokripour R, Kasirun ZM, Zamani S, Anvik J. Automatic bug assignment using information extraction methods. *In: Advanced computer science applications and technologies (acsat) 2012 international conference on*; 2012;11:144-149.
55. Tamrawi A, Nguyen TT, Al-Kofahi J, Nguyen TN. Fuzzy set-based automatic bug triaging: Nier track. In: Software Engineering (ICSE), 2011 33rd International Conference on IEEE; 2011:884-887.
56. Hossen MK, 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; 2014:130-141.
57. Kagdi H, Poshyvanyk D. Who can help me with this change request? In: Program comprehension, 2009. icpc'09. ieee 17th international conference on IEEE; 2009:273-277.
58. Xuan J, Jiang H, Ren Z, Zou W. Developer prioritization in bug repositories. In: Software Engineering (ICSE), 2012 34th International Conference on IEEE; 2012:25-35.
59. Anjali, Mohan D, Sardana N. Visheshagya: time based expertise model for bug report assignmentIEEE; 2016:1-6.
60. Zanjani MB, 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; 2015:88-98.
61. 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), FASE '16. Springer; 2016.
62. 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; 2014:25.
63. 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; 2017:230-240.

64. 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; 2017:3-11.
65. Sahu TP, Nagwani NK, Verma S. An empirical analysis on reducing open source software development tasks using stack overflow. *Indian J Sci Technol*. 2016;9(21).
66. Santos A, Juristo N. Comparing techniques for aggregating interrelated replications in software engineering. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement ACM; 2018:8.
67. Aaron B, Kromrey JD, Ferron J. *Equating "r"-Based and "d"-Based Effect Size Indices: Problems With a Commonly Recommended Formula*. Orlando, FL, United States: ERIC Clearinghouse; 1998.
68. McGaw B, Glass GV. Choice of the metric for effect size in meta-analysis. *Amer Educ Res J*. 1980;17(3):325-337.
69. Yang B. Meta-analysis research and theory building. *Adv Developing Human Res*. 2002;4(3):296-316.
70. Cuzick J. Forest plots and the interpretation of subgroups. *Lancet*. 2005;365(9467):1308.
71. Borenstein M, Hedges LV, Higgins JP, Rothstein HR. *Introduction to Meta-analysis*. Chichester, United Kingdom: John Wiley & Sons; 2011.
72. Čubranić Davor, Murphy GC. Automatic bug triage using text categorization. In: In seke 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering Citeseer; 2004:1–6.
73. Canfora G, Cerulo L. Supporting change request assignment in open source development. In: Proceedings of the 2006 ACM Symposium on Applied Computing ACM; 2006:1767-1772.
74. Cavalcanti YC, Machado IC, Neto PA, de Almeida ES, Meira SilvioRomerodeLemos. 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; 2014:325-330.
75. Tamrawi A, Nguyen TT, Al-Kofahi JM, Nguyen TN. 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; 2011:365-375.
76. Cavalcanti YC, do Carmo Machado I, Neto PAMS, de Almeida ES. Towards semi-automated assignment of software change requests. *J Syst Soft*. 2016;115:82-101.
77. Sun X, Yang H, Xia X, Li B. Enhancing developer recommendation with supplementary information via mining historical commits. *J Syst Soft*. 2017;134:355-368.
78. 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; 2014:32-39.
79. Manning CD, Raghavan P, Schütze H. *Introduction to Information Retrieval*. Cambridge, United Kingdom: Cambridge University Press; 2008.
80. Xu G, Zhang Y, Li L. *Web Mining and Social Networking: Techniques and Applications*, Vol. 6. Berlin, Germany: Springer Science & Business Media; 2010.
81. Bama SSathya, Ahmed M, Saravanan A. A survey on performance evaluation measures for information retrieval system. *Int Res J Eng Technol*. 2015;2(2):1015-1020.
82. Hindle A, Alipour A, Stroulia E. A contextual approach towards more accurate duplicate bug report detection and ranking. *Emp Soft Eng*. 2016;21(2):368-410.
83. Thom J, Scholer F. A comparison of evaluation measures given how users perform on search tasks. In: Adcs2007 australasian document computing symposium RMIT University, School of Computer Science and Information Technology; 2007:100–103.
84. Katerenchuk D, Rosenberg A. Rankdcg: rank-ordering evaluation measure. arXiv preprint arXiv:180300719; 2018.
85. Lioma C, Simonsen JG, Larsen B. Evaluation measures for relevance and credibility in ranked lists. In: Proceedings of the ACM Sigir International Conference on Theory of Information Retrieval ACM; 2017:91-98.
86. Clough P, Sanderson M. Evaluating the performance of information retrieval systems using test collections. *Inform Res*. 2013;18(2):18-2.
87. 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; 2012:155-164.
88. Shani G, Gunawardana A. Evaluating Recommendation Systems. In: Recommender Systems Handbook, Springer; 2011: 257-297.
89. Alipour A. A contextual approach towards more accurate duplicate bug report detection. *Master's Thesis*. Canada;2013.
90. Aggarwal K, Timbers F, Rutgers T, Hindle A, Stroulia E, Greiner R. Detecting duplicate bug reports with software engineering domain knowledge. *J Soft: Evol Process*. 2017;29(3):e1821.
91. Cruzes DS, Dybå T. Research synthesis in software engineering: A tertiary study. *Inform Soft Technol*. 2011;53(5):440-455.
92. Cormack GV, Lynam TR. Statistical precision of information retrieval evaluation. In: Proceedings of the 29th annual international acm sigir conference on research and development in information retrieval ACM; 2006:533-540.
93. Anvik J, Hiew L, Murphy GC. Who should fix this bug? In: Proceedings of the 28th international Conference on Software Engineering ACM; 2006:361-370.
94. Github. Collaborators. https://developer.github.com/v3/repos/collaborators/; 2017.
95. Sajedi A, Hindle A, Stroulia E. Crowdsourced bug triaging. In: ICSME '15 IEEE; 2015.
96. Kagdi H, Hammad M, Maletic JI. Who can help me with this source code change? In: Software maintenance, 2008. ICSM 2008. IEEE International Conference on IEEE; 2008:157-166.
97. Rahman MM, 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; 2009:439-442.
98. Kumari M, Jain A, Bhatia A. Synonyms based term weighting scheme: an extension to tf. idf. *Procedia Comput Sci*. 2016;89:555-561.
99. Sajedi-Badashian A, Stroulia E. Vocabulary and time based bug assignment (vtba): a recommender system for open-source projects. *Manuscript accepted for publication*. 2020:1–24. Preprint available at https://github.com/TaskAssignment/TTBAOutline/blob/master/TTBA.pdf and https://github.com/TaskAssignment/TTBA-Outline/blob/master/TTBA.pdf

100. Vehtari A, Mononen T, Tolvanen V, Sivula T, Winther O. Bayesian leave-one-out cross-validation approximations for gaussian latent variable models. *J Mach Learn Res*. 2016;17(1):3581-3618.
101. Github. Closing issues using keywords. https://help.github.com/articles/closing-issues-using-keywords/; 2018.
102. 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; 2017:926-931.
103. Anvik J, Murphy GC. Reducing the effort of bug report triage: recommenders for development-oriented decisions. *ACM Trans Soft Eng Method (TOSEM)*. 2011;20(3):10.
104. 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)*. 2006;1:1097-1101.
105. Zhang M, Hurley N. Avoiding monotony: improving the diversity of recommendation lists. In: Proceedings of the 2008 ACM Conference on Recommender Systems ACM; 2008:123-130.
106. Runeson P, Host M, Rainer A, Regnell B. *Case study research in software engineering: guidelines and examples*. Hoboken, New Jersey, United States: John Wiley & Sons; ,2012;1.
107. Miller J. Applying meta-analytical procedures to software engineering experiments. *J Syst Soft*. 2000;54(1):29-39.
108. 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; 2010:97-106.
109. Anvik J. Automating bug report assignment. In: Proceedings of the 28th International Conference on Software Engineering ACM; 2006:937-940.
110. 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; 2009:451-455.
111. Ahsan SN, 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; 2009:216-221.
112. 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; 2010:1-5.
113. Nasim S, Razzaq S, Ferzund J. Automated change request triage using alpha frequency matrix. In: Frontiers of information technology (fit), 2011 IEEE; 2011:298-302.
114. 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; 2011:389-396.
115. Kagdi H, Gethers M, Poshyvanyk D, Hammad M. Assigning change requests to software developers. *J Soft Evol Process*. 2012;24(1):3-33.
116. 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; 2012:19-28.
117. 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; 2012:2845-2848.
118. 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; 2012:505-512.
119. Servant F, Jones JA. Whosefault: automatic developer-to-fault assignment through fault localization. In: Proceedings of the 34th international conference on software engineering IEEE Press; 2012:36-46.
120. Zhang T, Lee B. A hybrid bug triage algorithm for developer recommendation. In: Proceedings of the 28th annual acm symposium on applied computing ACM; 2013:1088-1094.
121. 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; 2013:22–30.
122. Kevic K, Müller SC, Fritz T, Gall H. 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; 2013:17-24.
123. 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; 2013:215-219.
124. 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; 2014:1134-1139.
125. 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; 2014:891-894.
126. Shokripour R, Anvik J, Kasirun ZM, Zamani S. Improving automatic bug assignment using time-metadata in term-weighting. *IET Soft*. 2014;8(6):269-278.
127. Sharma M, Kumari M, Singh V. Bug assignee prediction using association rule mining. In: International conference on computational science and its applications Springer; 2015:444-457.
128. Jain S, Wilson SR. Automated bug assortment system in datasets. In: Inventive computation technologies (icict), international conference on, Vol. 2 IEEE; 2016:1-7.
129. Dedík V, Rossi B. Automated bug triaging in an industrial context. In: Software engineering and advanced applications (seaa), 2016 42th euromicro conference on IEEE; 2016:363-367.
130. Zhang W, Wang S, Wang Q. Baha: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis. *Chinese J Elect*. 2016;25(6):1011-1018.
131. Zanjani MB. 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; 2016:1091-1093.

132. Xia X, Lo D, Ding Y, Al-Kofahi JM, Nguyen TN, Wang X. Improving automated bug triaging with specialized topic model. *IEEE Trans Soft Eng*. 2017;43(3):272-297.
133. Goyal A. Effective bug triage for non reproducible bugs. In: Proceedings of the 39th international conference on software engineering companion IEEE Press; 2017:487-488.
134. 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; 2017:64-71.
135. 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; 2017:31-42.

How to cite this article: Sajedi-Badashian A, Stroulia E. Guidlines for Evaluating Bug-assignment Research. *J Softw Evol Proc*. 2020;e2250. https://doi.org/10.1002/smr.2250

APPENDIX A: METHODS AND TECHNIQUES USED BY DIFFERENT STUDIES COVERED BY OUR SURVEY

Table A1 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 A1 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)	Convoluted 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
Čubranić and Murphy[72]	✓																												
Anvik[109]			✓																										
Canfora and Cerulo[73]								✓														✓							
Anvik *et al*[93]			✓																										
Kagdi *et al*[96]																									✓				✓
Baysal *et al*[14]												✓																	
Lin*et al*[110]			✓										✓																
Jeong *et al*[1]	✓	✓																✓	✓										
Matter *et al*[42]																						✓							
Ahsan *et al*[111]			✓						✓				✓									✓							
Rahman *et al*[97]																													✓
Kagdi and Poshyvanyk[57]								✓	✓																✓				
Bhattacharya and Neamtiu[2]	✓	✓																✓	✓										
Chen *et al*[112]												✓						✓	✓										
Nasim *et al*[113]	✓	✓																											✓
Park *et al*[48]			✓						✓																		✓		
Wu *et al*[114]					✓								✓				✓												
Tamrawi *et al*[55]																						✓		✓					
Tamrawi *et al*[75]																						✓		✓					
Anvik and Murphy[103]	✓			✓																									
Aljarah *et al*[3]		✓																					✓						✓
Kagdi *et al*[115]								✓	✓																✓				
Bhattacharya *et al*[10]	✓	✓	✓	✓														✓	✓										
Shokripour *et al*[54]															✓	✓								✓					

TABLE A2 continued (Table A1).

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)	Convoluted 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
Xuan *et al*[58]	✓		✓														✓												
Xie *et al*[116]										✓	✓																		
Jain *et al*[117]																						✓							
Zhang and Lee[118]											✓					✓													✓
Jonsson *et al*[46]	✓		✓		✓		✓																						✓
Linares-Vásquez *et al*[4]								✓	✓																✓				
Servant and Jones[119]																									✓				✓
Rahman *et al*[51]																												✓	✓
Hosseini *et al*[52]	✓																												✓
Zhang and Lee[120]																	✓			✓	✓								✓
Shokripouret *al*[5]															✓								✓		✓	✓			✓
Naguibet *al*[121]										✓	✓																		
Kevic *et al*[122]														✓															✓
Jonsson[47]	✓	✓	✓		✓		✓																						✓
Banitaan and Alenezi[123]	✓																						✓						✓
Nguyen *et al*[6]										✓	✓																		✓
Yang *et al*[124]																	✓			✓	✓								
Hossenet *al*[56]									✓																✓				
Hu *et al*[43]												✓					✓												
Cavalcanti *et al*[74]									✓			✓	✓													✓			
Borg[125]							✓																						✓
Wang *et al*[62]																													✓
Shokripour *et al*[126]															✓								✓						

TABLE A3 continued (Table A1).

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)	Convoluted 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*[44]														✓			✓															
Sharma *et al*[127]																	✓												✓			✓
Sajedi *et al*[95]																										✓						✓
Zanjani *et al*[60]					✓									✓	✓													✓				
Jain and Wilson[128]	✓																															✓
Dedík and Rossi[129]	✓		✓											✓																		
Zhang *et al*[130]											✓	✓							✓													
Zhang *et al*[16]					✓									✓				✓														
Tian *et al*[12]														✓	✓													✓				✓
Jonsson *et al*[15]	✓		✓				✓							✓																		✓
Zhang *et al*[17]					✓						✓	✓		✓		✓		✓														✓

TABLE A4 continued (Table A1).

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)	Convoluted 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
Cavalcanti *et al*[76]			✓							✓				✓															✓			
Anjali *et al*[59]																									✓	✓						
Zanjani *et al*[131]					✓																							✓				
Khatun and Sakib[45]														✓																		✓
Sajedi *et al*[61]																										✓						✓
Park *et al*[49]			✓								✓	✓											✓							✓		✓
Liu *et al*[7]												✓																				✓
Karim *et al*[50]																															✓	✓
Lee *et al*[102]						✓																										✓
Zhang *et al*[63]											✓	✓					✓															
Xia *et al*[132]											✓	✓																				✓
Goyal[133]	✓																									✓						✓
Florea *et al*[134]			✓			✓																										✓
Florea *et al*[135]			✓								✓			✓			✓															
Khalil *et al*[53]																															✓	✓
Sun *et al*[77]											✓	✓					✓		✓									✓		✓		