JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 0000; **00**:1–**??**
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

An Empirical Study on How Expert Knowledge Affects Bug Reports

Paige Rodeghero[1], Da Huo[1], Tao Ding[2], Collin McMillan[1], and Malcom Gethers[2]

[1]*Department of Computer Science and Engineering*
University of Notre Dame, Notre Dame, IN 46545
Email: {prodeghe, dhuo, cmc}@nd.edu

[2]*Information Systems Department*
University of Maryland, Baltimore County, Baltimore, MD 21250
Email: {tding1027, mgethers}@umbc.edu

SUMMARY

Bug reports are crucial software artifacts for both software maintenance researchers and practitioners. A typical use of bug reports by researchers is to evaluate automated software maintenance tools: a large repository of reports is used as input for a tool, and metrics are calculated from the tool's output. But this process is quite different from practitioners, who distinguish between reports written by experts, such as programmers, and reports written by non-experts, such as users. Practitioners recognize that the content of a bug report depends on its author's expert knowledge. In this paper, we present an empirical study of the textual difference between bug reports written by experts and non-experts. We find that a significant difference exists, and that this difference has a significant impact on the results from a state-of-the-art feature location tool. Through an additional study, we also found no evidence that these encountered differences were caused by the increased usage of terms from the source code in the expert bug reports. Our recommendation is that researchers evaluate maintenance tools using different sets of bug reports for experts and non-experts. Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: bugs, experts, empirical, recommendation, features, textual

1. INTRODUCTION

A *bug report* is a description of unwanted software behavior. Bug reports are one of the most important artifacts in software maintenance. Software engineering practitioners use them to diagnose, locate, and repair software defects [1], and a recent study at Microsoft found that between 5% and 15% of a typical programmer's time is spent reproducing unwanted behavior described in bug reports [2]. Meanwhile, bug reports are used in many corners of software maintenance research, including developer recommendation [3, 4], change impact analysis [5], feature location [6, 7], defect localization [8], and traceability [9]. Effective software maintenance procedures almost

always rely on effective communication of problems via bug reports [10, 11, 12]; in research, they are so ubiquitous that their presence is often taken for granted.

But the source of bug reports is often obscure. Bug reports may be written by users of software who experience failures in that software in order to communicate those failures to maintainers. In contrast, bug reports may be written by the software's own programmers, as a way of recording and monitoring the progress of repairing defects. What distinguishes these reports is that some reporters, such as the software's programmers, have a high degree of expert knowledge about the software, while other reporters have almost none. Different studies have shown that people with this expert knowledge understand software behavior differently than people without it [13, 14, 15, 16]. But crucially, experts do not necessarily write "better" bug reports than non-experts; reports from experts and non-experts provide complementary information [17, 18]. Studies in industry have shown that the source of bug reports is important because programmers seek out this complementary information [19].

What is not known is the degree of *textual difference* between reports written by experts versus non-experts, and the effect that this difference has on software maintenance research [20]. The term "textual difference" refers to how experts may use keywords, sentence structure, and semantics that are unlike those that non-experts would use to describe the same problem. This textual difference is important because a large number of software maintenance research tools rely on text processing techniques such as information retrieval [4, 7, 3]. These text processing techniques are sensitive to textual differences [21, 22, 23]; software maintenance research that uses these techniques will also be sensitive to textual differences.

Textual differences between bug reports from experts versus non-experts are especially important in software maintenance research. A typical strategy for software maintenance tools is to treat every bug report in a database equally. For example, a developer recommendation tool will follow the same methodology when analyzing bug reports from experts, as it will follow for reports from non-experts [4, 7, 3], even though these reports may describe the same problem in quite different language. Current software maintenance literature gives little guidance on how — or whether — these reports should be treated differently. Given that industrial programmers distinguish between reports from experts versus non-experts [17], it is plausible that this distinction is also relevant for software maintenance researchers.

In this paper, we present an empirical study contrasting bug reports written by experts to bug reports written by non-experts. For our study, we define an "expert" as anyone who has contributed to the source code of a project, all others we consider "non-experts." In the first part of our study, we manually review some illustrative examples of textual differences in duplicate bug reports in a qualitative analysis. Then, we automatically compute the textual difference between numerous bug reports submitted by experts versus *duplicates* of those bug reports submitted by non-experts. We compute the textual difference using short text similarity metrics: STASIS [24] and LSS [25]. We found that, for the same bugs, non-experts are more likely to write similar reports than experts are.

To follow up on this finding, we separately evaluated the effectiveness of two well-cited software maintenance tools: a developer recommendation tool [4] and a feature location tool [26]. Over a large corpus of software projects, we found, although indirectly, that textual differences 1) did not affect performance of the bug triage technique that is based on text classification for

DOI: 10.1002/smr

different developers, but 2) did affect performance of the feature location technique based on textual similarity analysis.

Additionally, we explored the possibility that the textual differences were caused by increased usage of keywords from the source code by the experts. To do so, we analyzed the textual similarities between the bug reports and the source code responsible for the bug mentioned. We separated the expert comparisons from the non-expert comparisons and contrasted them to determine if there was a significant difference between the amount of source code mentioned in the two types of bug reports. We found that although expert and non-expert bug reports are not significantly similar, the differences are not necessarily caused by an abundance of source code terms used in expert bug reports.

2. THE PROBLEM

We address the following gap in software maintenance research literature: there is currently little understanding about the degree of and effect of textual differences in bug reports written by persons with expert knowledge about the program, and without that particular knowledge. At present, these textual differences may be causing unknown biases or performance problems in software maintenance research because, currently, research treats those bug reports identically. Software engineering practitioners, on the other hand, treat bug reports differently based on the source of the reports, and find benefits from reading reports from a diverse set of sources [17, 19]. Software maintenance researchers are, in effect, making a different assumption about bug report data than software maintenance practitioners. But as Panichella *et al.* point out in a recent ICSE paper, "poor parameter calibration or wrong assumptions about the nature of the data could lead to poor results" [27]. Hence, in our view, we should investigate whether the performance of software maintenance research tools may be increased if they more closely match the behavior of industrial practitioners.

The potential impact of the study is quite extensive. Bug reports are used in almost every corner of software maintenance research. For example, developer recommendation tools use text from bug reports to locate the correct developer to repair the bug [13, 14, 4, 15]. Feature location tools match text from bug reports to text in source code [28, 6]. Traceability tools connect bug reports with a diverse set of software artifacts based on textual data [9, 29]. Impact analysis tools predict which artifacts will be affected by incoming change requests, which are typically bug reports [5, 30, 31, 32]. Defect prediction tools use text from bug reports as training data to help predict what areas of source code may contain future problems or how much time will be required to repair the bugs [33, 34, 35]. Numerous other software maintenance tools use bug reports, and this study has the potential to impact the performance of those tools.

3. BACKGROUND

This section will describe background on the short text similarity metrics, developer recommendation technique, and feature location that we use in this study. These metrics and

DOI: 10.1002/smr

tools have been proposed and evaluated elsewhere; we discuss them here because they are key components of our study.

3.1. Short Text Similarity Metrics

We employ two Short Text Similarity Tools in our approach: STASIS [24] and LSS [25]. STASIS, by Li *et al.*, computes a similarity score between two text documents by blending "word semantic" similarity, "sentence semantic" similarity, and "word order" similarity. Word semantic similarity is similarity in meaning of each word in one document to each word in another document. STASIS uses WordNet [36] to calculate the distance between each pair of words in a knowledge base. A knowledge base is a hierarchical structure in which words are organized according to their meanings. STASIS then uses the word semantic similarity to compute the sentence semantic similarity, which is the similarity between all the words in one sentence to all the words in a different sentence. The sentence semantic similarity is then computed for all sentences in one document to all sentences in another document. Finally, the word order similarity is used to determine whether the words in the sentences appear in roughly similar order, which is important to preserve meaning, in particular, for adjectives referring to the same nouns, the location of qualifiers such as "not", and subject-verb-object placement. STASIS then combines these similarities with a weight of 0.85 given to sentence semantic similarity (which includes word semantics) and 0.15 to word order.

Croft *et al.* describe LSS as an alternative. Like STASIS, LSS uses WordNet to determine the word semantic similarity. But unlike STASIS, LSS does not consider the order of the words in the documents. Also, LSS calculates word similarity using "synsets" of words, which are sets of cognitive synonym words. The advantage is that LSS is able to identify highly similar meanings without respect to details such as verb tense, and without relying on the documents' authors to choose identical words. This process is different than in STASIS, which identifies synonyms solely through the knowledge base distance. The intent is that LSS is better suited to very short documents, perhaps only one or two sentences long, while STASIS may be better suited to longer documents, up to several paragraphs. We use both in our study because bug reports are likely to range in size from one sentence up to perhaps a page or more of text. Note also that we do not use Latent Dirichlet Allocation (LDA) in our study — the reason here being that even though LDA is widely-used for software artifacts [27], both STASIS and LSS have been shown to outperform LDA for computing the similarity between short natural language documents [24, 25], such as bug reports.

3.2. Developer Recommendation

Open source software is developed by a community of developers that can be distributed across various geographical locations. Bug tracking systems are particularly important in open-source software development because they are not only used to track problems, but also to coordinate work among developers. Bug tracking systems allow people anywhere in the world to report a bug. As a result, there are a large number of bugs that are submitted each day. The time commitment required to filter the invalid bugs and decide what to do with new report becomes a burden. Despite the tedious nature of the task, most bugs are assigned manually to developers, such as in the case of Mozilla and Eclipse, and have therefore been forced to introduce team members who are dedicated to bug triaging.

DOI: 10.1002/smr

To solve the problem, various approaches have been proposed to semi-automate the bug triage process. A number of techniques are based on information retrieval. Canfora and Cerulo [37] presented an approach based on information retrieval, in which they use a probabilistic text similarly to support change request assignment. The paper presented a case study on Mozilla and KDE which reported recall levels of around 20% for Mozilla. Some studies combine information retrieval techniques and processing of source code authorship information to recommend developers [38].

Some studies use various machine learning techniques. Cubranic and Murphy [39] proposed a Bayesian learning approach for bug triage. The prediction model is learned from labeled bug reports and makes prediction based on observed rules. It achieved precision levels of around 30% on Eclipse. Somasundaram [40] combined a supervised learning model based on support vector machines and an unsupervised generative model based on LDA. Tamrawi proposed Bugzie for triaging based on fuzzy set-based modeling of the bug-fixing expertise of developers [41]. Jeong *et al.* [42] applied a graph model based on Markov chains to reveal developer networks and, combined with Bayesian learning, help better assign developers. Anvik *et al.* [4] utilized support vector machines to classify bugs and improve precision up to 64% by refining data sets, which filtered out developers who did not make enough contribution in the most recent 3 months.

3.3. Feature Location

The goal of feature location is to identify source code associated with a given feature of the software system [43]. Over the years, researchers have proposed several semi-automated techniques to assist with the process of feature location [43]. Three main types of analyses are typically employed: dynamic, static, and textual analysis [43]. In this paper, we focus primarily on textual analysis-based techniques.

Several of the feature location techniques proposed by researchers have applied information retrieval as a means of analyzing textual information in source code artifacts for feature location. Marcus *et al.* [44] first proposed an information retrieval approach for feature location based on Latent Semantic Indexing (LSI). Cleary and Exton [45] presented an approach based on a complementary information retrieval method which used information flow and co-occurrence information derived from non-source code artifacts to implement a query expansion-based concept location technique. Rao and Kak [46] applied SUM for feature location, which was found to be the best performance model.

Some researchers have focused on supplementing the textual analysis of IR techniques with other sources of information to enhance the performance of feature location techniques. Gay *et al.* [47] proposed to augment information retrieval-based feature location with an explicit relevant feedback mechanism. Poshyvanyk *et al.* [48] proposed a feature location method called PROMESIR based on Latent Semantic Indexing and Scenario-based Probabilistic Ranking (SPR), a dynamic analysis-based technique. Lukins *et. al* [49] proposed an approach which used Latent Dirichlet Allocation (LDA), a more recent technique for information retrieval, to search for bug-related methods and files. This technique was found to have significant advantages over LSI and pLSI (probabilistic Latent Semantic Indexing) by using a faster, more stable probabilistic generation of hidden, or latent, topics in the document. [49] Rao and Kak [46] found the performance of the Vector Space Model (VSM) technique in bug localization is worse than SUM but better than LDA and LSI.

DOI: 10.1002/smr

Zhou *et al.* proposed BugLocator, an information retrieval-based method for locating the relevant source code files for fixing bugs according to an initial bug report [26]. BugLocator uses revised the Vector Space Model to rank all files based on text similarity between the initial bug report and the source code ($rVSMScore$). BugLocator also takes into consideration information about similar bugs that have been previously fixed ($SimiScore$). The final score is a weighted sum of these two scores.

$$FinalScore = (1 - \alpha) * N(rVSMScore) + \alpha * N(SimiScore) \quad (1)$$

, where α is a weighting factor. Zhou *et al.* compared BugLocator with LDA, SUM, and LSI. The results clearly showed BugLoctor outperforms all other methods. In our empirical study section, we use BugLocator to evaluate how textual differences will affect feature location.

4. QUALITATIVE ANALYSIS

In this section, we describe a qualitative analysis of duplicate bug reports written by experts and non-experts. These reports were taken from the Mozilla public database. We manually compare the textual difference between two expert reports, two non-expert reports, and one of each. Before automating the process of comparing duplicate bug reports, we manually inspect pairs of bug reports between experts and non-experts. This inspection will help reveal how similar we may expect descriptions in duplicate bug reports to be during the empirical study. Since these examples were all taken from the same randomly chosen bug with duplicate reports, they serve as good illustrations of what we might see when exploring the entire bug reports repositories.

First, we will look at a duplicate pair of expert bug reports:

1. *227146:* When a destination is unreachable, the Page load error converts the URLbar into something rather unmemorable (Its a `chrome://` thing, if I recall correctly). Instead, the failed URL should remain in the url bar. I won't go into the uselessness of the actual XUL error page in this bug. Pike has whipped up an extension to resolve this issue, but I think it should be in FB by default.
2. *157004:* When error page support is enabled (see bug 28586), the error page url is displayed in the location bar of the browser. This needs to be suppressed somehow and the page should not appear in the session history either.

In these reports, both experts are describing the same basic problem. The problem described here is when a page is not able to be loaded, a long, incomprehensible error URL replaces the URL of the unavailable page. Both experts believe the unavailable URL should remain in the address bar, instead. As can be seen from the samples, although both reports are describing the same problem and solution, neither are worded exactly the same. Both reports use the terms bar, error, page, and URL. However, many of the other words are dissimilar. For example, both reports mention the address bar, but one calls it the URL bar, while the other refers to it as the location bar. Other textual differences are seen by the fact that the reports sometimes mention specifics that are not mentioned in the other report. For example, the first report mentions that the error URL begins with `chrome://`, that someone else may have already solved this problem, and a personal statement about the usefulness

Prepared using smrauth.cls
DOI: 10.1002/smr

of the error page itself. In contrast, the second report includes a reference to an old bug number and mentions that the session history should not record the opening of the error page. From this sample, one can see that even though two expert bug reports explain the same problem, they may only be slightly textually similar.

Second, we will look at a duplicate pair of non-expert bug reports:

1. *160626:* When the new friendly html error messages are displayed the current url is replaced, preventing corrections to the url or use of the back arrow to get to the previous page. open browser, type asdf in location box, hit enter. This is returned: The connection was refused when attempting to contact www.asdf.com. connectionFailure long description goes here. Try again Search for this address The url line is changed to: `chrome://global/content/netError.xhtml ?e=connectionFailure&u=http%3A//www.as df.com/&d=The%20connection%20was%20ref used%20when%20attempting%20to%20contact%20www.asdf.com.` The url line should not be changed. The end user may want to correct a type-o. Also the back arrow goes back two pages instead of to the previous one.
2. *274917:* If a url is opened in a new window or tab and that url ultimately doesn't load or times out, the url is not kept in the address bar and you cannot attempt to reload it. This problem can be annoying if you have a flaky internet connection, as I do. The problem is most prominent when I open several tabs off of one page, then close the original page. Then if any of the tabs do not load, I've lost the URL that I was trying to view and ahve to navigate back to the original page and figure out which link didn't load, since there is no indication.

These reports are also duplicates of the two expert reports we inspected above, so they are explaining the same basic problem: the 'page unavailable' error causes the address bar to be filled with an unneeded error URL. Textually, these two reports use many of the same terms, such as URL, connection, and attempt. However, as we've seen, there are many textual differences. Using the same example as with the expert samples, both of these reports mention the address bar, but call them different names. The first report calls it the location box, while the second calls it the address bar. As before, many textual differences come from the fact that the two reports have additional comments unique to its description. The first report is more thorough than the second one. The first report gives a full procedure on how to repeat the bug. This report also gives the full error URL that normally appears, gives a personal reason for why this is a problem, and mentions that hitting the back button should return to the unreachable URL. In contrast, the second report mentions personal internet problems that led to the creation of this bug report. Another issue that can cause textual differences that was not seen with the expert reports are misspelled words. The second non-expert report in this sample has a couple of misspelled words. Just as with the experts, one can see that although these reports are duplicates, they do not explain the problem in exactly the same way.

Last, we will look at a duplicate bug report between an expert and a non-expert:

1. *157004 (expert):* When error page support is enabled (see bug 28586), the error page url is displayed in the location bar of the browser. This needs to be suppressed somehow and the page should not appear in the session history either.
2. *160626 (non-expert):* When the new friendly html error messages are displayed the current url is replaced, preventing corrections to the url or use of the back arrow

Prepared using smrauth.cls
DOI: 10.1002/smr

> to get to the previous page. open browser, type asdf in location box, hit enter. This is returned: The connection was refused when attempting to contact www.asdf.com. connectionFailure long description goes here. Try again Search for this address The url line is changed to: `chrome://global/content/netError.xhtml ?e=connectionFailure&u=http%3A//www.as df.com/&d=The%20connection%20was%20ref used%20when%20attempting%20to%20contact%20www.asdf.com.` The url line should not be changed. The end user may want to correct a type-o. Also the back arrow goes back two pages instead of to the previous one.

These two bug reports are two of the same reports from the previous expert and non-expert comparisons shown above. Therefore, they describe the same bug we have been describing during the rest of this analysis. As we have seen before, these two reports are textually similar because they are trying to explain the same problem, and therefore, use similar words. Also, these reports are textually different for the same reason as before: both reports include descriptions unique to the reporter's situation. However, unlike the previous pairs, both of these reports include that the error URL should not be included in the history. Even with this similarity, though, since the reports describe internet history differently, they are still mostly textually different.

Although only taken from one bug instance, we see these examples as indicative of the other bug reports because of the random sampling. From these examples, we believe that many of the similarities naturally come from the reports being duplicates. We also believe that many of the textual differences come from the extra notes (beyond the description of the bug itself) that are not very common between bug pairings. These reasons for similarities and differences are important to notice before we begin the large automated portion of the study because it will help with determining why we see the levels of textual difference during our analysis. With these textual differences in mind, we now move onto the empirical study.

5. EMPIRICAL STUDY DESIGN

This sections explains the design of our empirical study including our research objective, research questions, methodology, and study conditions.

5.1. Research Questions

The research objective of our empirical study is two-fold: 1) to determine the degree of the textual difference between bug reports written by experts and non-experts, and 2) to determine the degree to which that similarity may affect the performance of software maintenance tools. Towards the first part of this objective, we pose the following Research Questions (RQ):

RQ_1 Are there textual similarities between bug reports written by experts and duplicates of those bug reports written by non-experts?

RQ_2 Are there textual similarities between duplicate bug reports written by experts?

RQ_3 Are there textual similarities between duplicate bug reports written by non-experts?

DOI: 10.1002/smr

The rationale behind RQ_1 is that both experts and non-experts write bug reports, and that some percentage of these reports will be duplicates. Because those duplicates refer to the same underlying problem, the textual similarity of those duplicates will indicate the difference between reports written by experts and non-experts for the same bug. Likewise, the rationale behind RQ_2 and RQ_3 is to obtain a baseline for comparing the textual similarities relative to each other (see Analysis Questions).

Towards the second part of our research objective, we ask these research questions:

RQ_4 What is the performance of a software maintenance tool when the inputs to that tool are bug reports written solely by experts?

RQ_5 What is the performance of a software maintenance tool when the inputs to that tool are bug reports written solely by non-experts?

RQ_6 What is the performance of a software maintenance tool when the inputs to that tool are bug reports written by both experts and non-experts?

The rationale behind RQ_4, RQ_5, and RQ_6 is that it is plausible that software maintenance tools will have different performance when provided bug reports from different sources. Here, performance is the correctness of the developer recommendation or feature location. This rationale is based on the idea that human programmers treat expert and non-expert bug reports differently, and that software maintenance tools might benefit from this distinction, as well (see Section 2).

5.2. Analysis Questions

To analyze and draw conclusions from the data collected by answering the research questions, we pose the following two Analysis Questions (AQ):

AQ_1 Is there a statistically-significant difference between the textual similarity values calculated for RQ_1, RQ_2, and RQ_3?

AQ_2 Is there a statistically-significant difference between the performance values calculated for RQ_4, RQ_5, and RQ_6?

The rationale behind AQ_1 is that if the textual similarity among bug reports written by experts versus non-experts is higher than the textual similarity among reports written solely by experts or solely by non-experts, then it is evidence that experts and non-experts tend to write bug reports using different vocabulary to describe similar situations.

Likewise, the rationale for AQ_2 is that if the performance of the software maintenance tool is higher using one data set versus another, then it is evidence that software maintenance tools benefit more from the information in that data set.

5.3. Research Subjects

We used two bug repositories for our data for the first two studies. We obtained the bug reports via public databases for Eclipse* and Mozilla†. These repositories are extensive and include several

*https://bugs.eclipse.org/bugs/
†https://bugzilla.mozilla.org/

Prepared using smrauth.cls
DOI: 10.1002/smr

years of data over several versions of different software products, including contributions by many programmers and users. Table I presents details about the repositories. In total we extracted over 410,000 bug reports, of which 268,000 were duplicates suitable for our empirical study.

In the third study, a subset of the data in Table II is downloaded from the project site of BugLocator[‡]. We used the sample data set of SWT (98 bugs) and Eclipse (3070 bugs), which was experimental data used in paper [4]. Note that some of the sample data is also a subset of Table I.

5.4. Methodology Overview

The methodology we follow to answer our research questions is to perform three empirical studies. The first involves AQ_1, and the other two involve AQ_2.

5.4.1. Textual Comparison with Metrics The first empirical study is a textual comparison of bug reports. In this study we divided a repository of bug reports into two groups: bugs written by experts, and bugs written by non-experts (an expert is defined as a contributor to source code, all others we consider non-experts"). Then we extracted any bug reports labeled as *duplicates*. This extraction produced three groups of pairs of duplicates: 1) pairs where both reports were written by experts, 2) pairs where both reports were written by non-experts, and 3) pairs where one report was written by an expert and one was written by a non-expert. For each of these three groups, we used two different Short Text Similarity algorithms (see Section 3.1) to compute a similarity value for each pair of duplicates. The result was a list of similarity values for each of the three groups of duplicate bugs. These lists were our basis for answering RQ_1, RQ_2, and RQ_3. Finally, we performed a statistical hypothesis test to determine the significance of any difference among the mean values of these groups, which allowed us to answer AQ_1.

5.4.2. Developer Recommendation The second empirical study is to evaluate how textual differences impact textual analysis-based automated approaches to bug triaging. Table I shows the number of bug reports, and the number of developers involved. We extracted those bugs which were fixed and tagged as ASSIGNED, RESOLVED, FIXED, VERIFIED, and CLOSED. We labeled each of the bugs with a developer id based on who was assigned to fix the bug. We divided the bug reports into two groups, as was done in the first study: expert and non-expert. The description of bugs includes two parts in each report: the title, which briefly summarized the issue, and the long description, which provided details about the bug. In this study, we determine the impact of textual differences, considering the case where only the summary is used, as well as the case where the summary and short description is used. Two subgroups are produced for each group: bug descriptions containing title only (E, NE) and bug descriptions containing title and long description together (EL, NEL).

There are two phases: training and prediction. First, the classifier model will be built using labeled bug reports, where the label is the developer who actually fixed the bug. When selecting training reports, we used the same strategy mentioned by Anvik *et al.* [4]. We refined the set of training reports based on profiles of each developer, filtering out those bug reports where the developer fixed

[‡]`http://code.google.com/p/bugcenter/wiki/BugLocator`

DOI: 10.1002/smr

less than 9 bugs in the most recent 1000 bugs. After filtering, we consider 24 developers for Mozilla and 21 developers for Eclipse, as shown in Table I. For each group, we set 10000 as the sample size and use 90% as training data with the remaining 10% being used as testing data.

Second, when a new report arrives, the classifier will suggest a ranked list of suitable developers to fix the bug. The higher the ranking score is, the more suitable the developer will be. To evaluate our approach, we use the same methodology as Anvik *et al.* [4]. We use this methodology because it is well-cited, appeared in a top venue, and is simple enough to be applied widely. We search the top 3 recommended developers based on probability; if one is the correct developer, we consider the bug as being assigned correctly. In this study, we evaluate the impact of textual differences on the accuracy of the developer recommendation technique in order to answer RQ_4, RQ_5, and RQ_6.

5.4.3. Feature Location In this empirical study, we evaluated how the textual differences impact a textual analysis-based feature location technique, namely BugLocator [26]. BugLocator will provide a ranked list of code files for each bug based on a similarity score, which captures the relationship between a new bug report and the source code. BugLocator actually combines textual analysis with an analysis of existing bug reports in order to identify relevant source code files. The weighting factor α is used to control how much weight is given to the historical information. In the prior study conducted by Zhou *et al.* [26], they were able to achieve the highest accuracy by configuring BugLocator with $\alpha = 0.2$ for SWT and $\alpha = 0.3$ for Eclipse. Thus, we use the same α values in our experiments. We also examine different levels of α to see how focusing primarily on textual information impacts the results. Here we use the same groups that were used in the developer recommendation study and shown in Table II.

5.5. Measurement Metrics

We use accuracy to evaluate the performance of the developer recommendation tool for our second empirical study. The accuracy of prediction is a fraction between the number of bugs assigned to correct developers and the total number of bug assignments.

$$Accuracy = \frac{\#\ correct\ predicted\ bug}{\#\ predicted\ bug} \tag{2}$$

To measure the effectiveness of the feature location method, we use the following metrics in the third empirical study:

- Top N rank, which is the number of bugs whose associated files are ranked in top N files. Given a bug report, if the top N query result contains at least one file at which the bug should be fixed, we consider the bug located.
- MHR (Mean Highest Rank), which is the mean of highest ranks for all known relevant files when testing n bug reports, which was used to complement BugLocator's "FinalScore".

$$MHR = \frac{\sum HighestRank}{n} \tag{3}$$

5.6. Statistical Tests

We use a Mann-Whitney U-test [50] to determine the statistical significance of the difference of means among the groups of short text similarities. The Mann-Whitney test is appropriate for this

DOI: 10.1002/smr

analysis because it is unpaired, it is tolerant of unequal sample sizes, and it is non-parametric. Our data are unpaired in the sense that we are comparing similarities of different pairs of bug reports. Our data are of unequal size because of the varied numbers of duplicate bugs in the repositories. Finally, we cannot guarantee that the data are normally-distributed, so we conservatively choose a non-parametric test.

5.7. Threats to Validity

As with any study, our work carries threats to validity. One threat is our bug report repository. The results from our study are dependent on these repositories, and the results may not be applicable to bug reports for all programs. We have attempted to mitigate this threat by using large repositories with hundreds of thousands of bug reports from different programs, however we still cannot guarantee that our results are consistent for every repository. Another threat is our definitions of experts and non-experts. It is possible that one of our ”experts” could have committed a single time throughout the entire project while one of our ”non-experts” could have been interacting with this project throughout its entire life-cycle and knows it very well. We believe, however, that our data set is large enough, and this occurrence is rare enough, that any inconsistencies cause by this are minimized. Additionally, we used a filtering process to eliminate users with less than a certain number of bug reports. Another threat to validity is our selection of text similarity algorithms. We use these algorithms as metrics for determining relative similarity of text. Different text similarity algorithms might return different results. Plus, we are exposed to the risk that these algorithms may have inaccuracies. We attempt to mitigate this risk by using two different algorithms which have been independently verified in related literature. A similar risk also exists from the developer recommendation tool we chose, however this risk is reduced by the design of our study: we are studying the effect of different data sets on the tool, and can draw a conclusion related to these data sets for the tool regardless of inaccuracies in the tool. A potential threat to validity exists in that different developer recommendation tools may be affected differently, though we attempt to mitigate this risk by using a prominent and frequently-cited tool.

5.8. Reproducibility

For the purposes of reproducibility and independent study, we have made all raw data, scripts, tools, processed data, and statistical results available via an online appendix: `http://www.cse.nd.edu/~cmc/projects/bugsim/`

6. TEXTUAL SIMILARITY RESULTS

This section describes the results of our empirical study for RQ_1, RQ_2, and RQ_3. These are the questions from our study focusing on the textual similarity of bug reports. We first provide an overview of our results, then provide raw details related to our research questions, and finally present our data interpretation that led to our results.

Prepared using smrauth.cls
DOI: 10.1002/smr

6.1. Results Overview

A brief overview of our results is that we found evidence that 1) experts use different language than non-experts to describe the same bugs, and 2) experts are more consistent in their use of language in bug reports than non-experts.

6.2. RQ_1 - Textual Similarities: Expert vs. Non-Expert

The degree of textual similarity between bug reports written by experts and duplicates of those reports written by non-experts is, on average, 0.609 according to STASIS and 0.983 according to LSS. Descriptive statistics are in Table VIII and Figure 1. Though it is not appropriate to draw conclusions from these numbers in isolation, we observe that while the range of values is quite large, approximately half of the values for STASIS lie between 0.55 and 0.65. This is a broader range than for LSS, where the values fall in a relatively narrow band. While we do not draw conclusions from these ranges, we note that they are likely due to differences in the operation of STASIS and LSS, in that STASIS is intended mostly for larger blocks of text, while LSS is intended for shorter blocks, in general (see Section 3.1). This is an important distinction because the repositories contain bug reports of varying lengths.

6.3. RQ_2 - Textual Similarities: Expert vs. Expert

The degree of textual similarity among duplicate bug reports written by experts is 0.609 according to STASIS and 0.979 according to LSS. We observe a similar pattern for these similarity values as for RQ_1: a somewhat narrow band for LSS as compared to STASIS. Full descriptive statistics are in Table VIII and Figure 1.

6.4. RQ_3 - Textual Similarities: Non-Expert vs. Non-Expert

The degree of textual similarity among duplicates written by non-experts is 0.619 for STASIS and 0.978 for LSS. The patterns we observe are consistent with our observations for RQ_1 and RQ_2. As before, statistics are in Table VIII and Figure 1.

7. EVALUATION OF IMPACTS ON SOFTWARE MAINTENANCE

This section describes the results of our two empirical studies: developer recommendation and feature location for answering RQ_4, RQ_5, and RQ_6.

7.1. RQ_4 - Tool Performance: Experts Only

In the developer recommendation study, when we used bug descriptions without long descriptions and only expert bugs are considered: for Eclipse (in Table III), accuracy for development recommendation is 0.81, for Mozilla (in Table III), accuracy is 0.64. When long descriptions are added in the bug descriptions, both accuracies are decreased to 0.75 and 0.58, respectively.

In the feature location study, for SWT when α equals 0.2, the accuracy is 35% at the top 1 when we used bug descriptions without long descriptions(E), which is higher than the accuracy of

Prepared using smrauth.cls
DOI: 10.1002/smr

32% when long descriptions are considered(EL). With increasing ranking range, the EL's accuracy is higher than E's. E's MHR is higher than EL's when history is only one factor($\alpha = 1$), which means longer descriptions are helpful when only considering similarity with bugs that had been fixed before.

7.2. RQ_5 - Tool Performance: Non-Experts Only

In the developer recommendation study, when we used bug descriptions without long descriptions and only non-expert bugs are considered: for Eclipse (in Table III), accuracy for development recommendation in Eclipse is 0.74, for Mozilla (in Table III, accuracy is 0.64. When long descriptions are added in the bug descriptions, both accuracies are decreased to 0.75 and 0.58, respectively.

In studies of the feature location, from Tables IV and V, we can see that the non-expert bug reports with long description (NEL) will achieve the highest ranks among all bug report types. Comparing all bugs with and without long descriptions (All and AllLong), for most α values, we can say that the long descriptions help to achieve higher ranks. We can also get the same result from comparing the non-expert bug reports with and without long descriptions (NE and NEL). However, for the bug reports written by experts, the longer description will lower the ranks.

7.3. RQ_6 - Tool Performance: Experts and Non-Experts

From Tables VI and VII, all the bug report types with long descriptions (AllLong, EL and NEL) have relatively equal ranking performance, which is much better than the types without long descriptions (AllLong, E and NE). Also, the NE type (non-expert bug descriptions) has the worst MHR performance among all types.

8. COMPARISON ANALYSIS

In this section, we describe our results for AQ_1 and AQ_2.

8.1. AQ_1 - Textual Similarity

We found statistically significant evidence for the difference of the means for both STASIS and LSS reported for RQ_1, RQ_2, and RQ_3. We compared the STASIS values from RQ_1, RQ_2, and RQ_3. Likewise, we compared the LSS values for those RQs. Note that we do *not* compare STASIS values to LSS values in our statistical analysis, as the two metrics operate differently.

The procedure we used to obtain the evidence is as follows. Consider the statistical data in Table VIII. We organized this table by similarity value for expert-written reports to duplicates written by other experts ("Expert-Expert") compared to similarity values for non-expert-written reports to duplicates written by other non-experts ("NonEx.-NonEx."). We also compared these similarity values to similarities for expert-written reports to non-expert-written duplicates ("Expert-NonEx."). These similarity values are further separated by the metric which produced those values (STASIS or LSS). We then posed six hypotheses:

H_1 There is no difference between STASIS similarities for Experts-Experts and NonEx.-NonEx.

Prepared using smrauth.cls
DOI: 10.1002/smr

H_2 There is no difference between STASIS similarities for Experts-Experts and Experts-NonEx.

H_3 There is no difference between STASIS similarities for NonEx.-NonEx. and Experts-NonEx.

H_4 There is no difference between LSS similarities for Experts-Experts and NonEx.-NonEx.

H_5 There is no difference between LSS similarities for Experts-Experts and Experts-NonEx.

H_6 There is no difference between LSS similarities for NonEx.-NonEx. and Experts-NonEx.

We performed a Mann-Whitney test for each of these hypotheses (see Section 5.6). We rejected a hypothesis only if the value for Z exceeded Z_{crit} and p falls below 0.05. Based on these criteria, we found evidence to reject all hypotheses.

8.2. AQ_2 - Effects on Tools

We compare the mean for the ranking score for correct developers from different groups (All, E, NE). Each group contains 1000 bugs in our study. To find statistically significant evidence for RQ_3, RQ_4, and RQ_5 in developer recommendations, we produced this hypothesis:

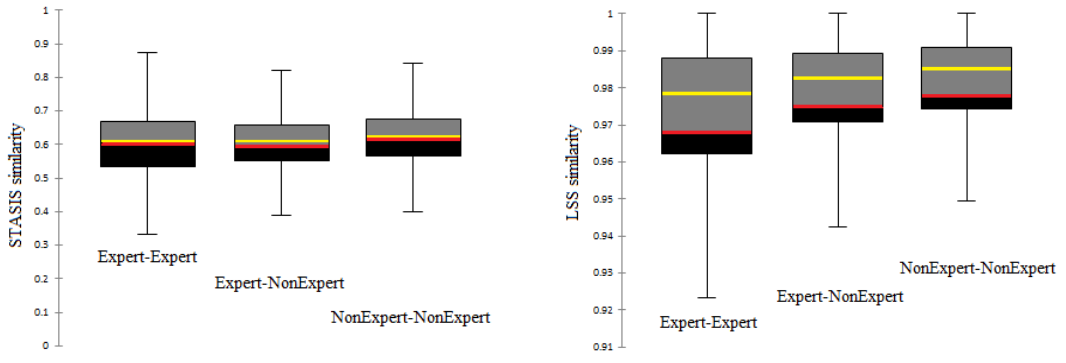
H_7 There is no difference between accuracy in developer recommendation from Experts reports and NonEx. reports.

We perform a U-test for H_7 in two projects: Eclipse and Mozilla. We rejected the hypothesis when p falls below 0.05. Table IX presents the hypothesis in Eclipse when long descriptions are considered in bug reports.

We also compare the mean for final similarity (see "FinalScore" Section 3) between bug reports and relevant source code files from each group. To find statistically significant evidence for RQ_3, RQ_4, and RQ_5 in feature locations, we produced this hypothesis:

H_8 There is no difference between final similarities in feature location study for Experts reports and NonEx. reports.

We performed a U-tests for H_8 to compare Expert and NonEx (see Section 5.6). At 0.05 confidence level, we reject hypothesis H_8.



(a) (b)

Figure 1. Boxplots comparing (a) STASIS similarity (b) LSS similarity for Expert-Expert, Expert-NonExpert, and NonExpert-NonExpert. The red line indicates the mean and the yellow line separating the gray and black areas indicates the median

Prepared using smrauth.cls
DOI: 10.1002/smr

9. INTERPRETATION

One key finding we discovered is that, **for the same bugs, non-experts are more likely to write similar reports than experts are** (we rejected both H_1 and H_4). In other words, the variation in information described by experts is higher than the variation among non-experts. A possible explanation from related work is that experts focus on knowledge about parts of the source code which cause the bug, while non-experts describe a range of faulty behaviors they experience [17, 18]. However, this empirical study alone does not include evidence for this explanation — it only confirms the existence of a difference in how experts and non-experts write bug reports, and that the variation among experts is lower than among non-experts. The qualitative study provides a few examples as evidence of a few possible reasons, but that is not enough to fully prove any of those reasons.

We did not find evidence that experts write more or less varied reports compared to other experts, versus compared to non-experts. While we rejected both H_2 and H_5, the direction of the difference in means was different for STASIS and LSS, showing that the similarity metrics did not fully agree. This finding, combined with the lack of strong variations in the qualitative samples, leaves us unable to draw a strong conclusion about the textual similarities.

A further finding, although not clearly evident in the qualitative study, is that the textual similarity of bug reports written by non-experts is higher as compared to other non-experts, versus when compared to experts. We rejected both H_3 and H_6, and both similarity metrics agreed on the direction of the difference. What this means is that the most consistent group of bug reports was the non-experts to other non-experts. Therefore, **non-experts write reports that are more similar to each other, than to reports written by experts.**

In analyzing the effects of these differences, we came to two conclusions. First, that the developer recommendation tool was not affected by the differences between expert and non-expert bug reports by a statistically-significant margin (only one of four statistical tests rejected). But on the other hand, the feature location tool was affected (all four tests rejected). Therefore, we find that different tools are affected to different degrees by the expert and non-expert bug reports. Our recommendation is that software maintenance tools be tested for this effect to maximize the performance of the tools.

10. SOURCE CODE COMPARISON STUDY DESIGN

This sections explains the design of our source code comparison study including our research objective, research questions, methodology, and study conditions.

10.1. Research Questions

The research objective of our source code comparison study is to determine the degree of the textual difference between bug reports written by experts and non-experts and the source code that the bug report derives from. Towards this objective, we pose the following Research Question (RQ):

RQ_7 Are the bug reports written by experts more textually similar to source code than the bug reports written by non-experts?

Prepared using smrauth.cls
DOI: 10.1002/smr

The rationale behind RQ_7 is that although we have shown how similar, and different, expert and non-expert bug reports are to each other, we have not found evidence to explain these differences entirely. One plausible explanation is that since experts know the source code better, they may use keywords directly from the source code to describe the problem. However, since non-experts may have seen the source code very little, or not at all, they may not use any of the same keywords as the experts.

10.2. Methodology Overview

The methodology we follow to answer our research question is a textual comparison between bug reports and source code. In this study we used the same sample data set of Eclipse bug reports from BugLocator[§] as was utilized during the feature location empirical study (see Section 5.4.3). We then divided this repository of bug reports into two groups: bugs written by experts, and bugs written by non-experts, as was done during the textual comparison empirical study. For both of these groups, we used two different Short Text Similarity algorithms (see Section 3.1) to compute a similarity value between the bug report and the source code referenced in the bug report. The result was a list of similarity values for both the expert-source and non-expert-source pairings. These lists were our basis for answering RQ_7 and determining any statistically significant difference between the similarity values of these two groups.

10.3. Research Subjects

The Eclipse bug report data in Table II was downloaded from the project site of BugLocator, which was experimental data used by Anvik *et al.* [4]. This sample data is also a subset of the data located in Table I. The Eclipse source code used for comparison was gathered from the Eclipse Project Archive [¶].

10.4. Statistical Tests

We use the Mann-Whitney non-parametric statistical test [50] to determine the significance of the difference of means among the groups of short text similarities. The Mann-Whitney test is appropriate for this analysis because it is unpaired and is tolerant of unequal sample sizes [50]. Our data are unpaired in the sense that we are comparing similarities of different types of bug reports together with different source code files. Our data are of unequal size because of the varied numbers of expert vs. non-expert bug reports in the repository. Finally, we cannot guarantee that the data are normally-distributed, so we conservatively choose a non-parametric test.

10.5. Threats to Validity

Many of the threats to validity in this source code comparison study are similar to the threats associated with the empirical study. One threat is the sample bug report repository we used. The results from our study are dependent on BugLocator's Eclipse repository, and the results may not be applicable to all bug reports for all programs. Another threat to validity is the size of this bug

[§]`http://code.google.com/p/bugcenter/wiki/BugLocator`
[¶]`http://archive.eclipse.org/eclipse/downloads/`

Prepared using smrauth.cls
DOI: 10.1002/smr

report sampling. The amount of bug reports is not insignificant, but is also not large enough to guarantee that are results are fully generalizable. Another threat to validity is our selection of short text similarity algorithms. Different short text similarity algorithms could have resulted in different outcomes. Also, these algorithms may not be fully accurate when determining similarity values between two pieces of text. In order to decrease this risk, we used multiple algorithms that have been externally verified by short text comparison experts. Lastly, there is a threat to validity in the creation of scripts for extracting source code files associated with each bug report. There may be unknown internal issues with select bug reports or select source code files that cause inconsistencies for some text comparisons. We believe that this would occur a minimal amount of time, however, since both the bug reports and source code files come from well-known, prominent sources.

11. SOURCE CODE COMPARISON STUDY RESULTS

In this section, we present our results to our research question. We also present a discussion of these results.

11.1. RQ_7 - Similarity to Source Code

We found no statistical evidence that experts use any more source code keywords in their bug reports than non-experts. In fact, the mean similarity scores for expert-source pairs and non-expert-source pairs were roughly the same. Descriptive statistics are in Table XI. To determine significance, we computed the similarity scores between bug report and source code pairs using both STASIS and LSS short text similarity metrics. On average, the experts had a STASIS similarity to the source code of 0.3968, while the non-experts had a similarity score of 0.4074. Likewise, the experts averaged a LSS similarity score of 0.7931, while the non-experts averaged a score of 0.7880. We used these scores to compare the similarity between expert pairs and non-expert pairs. We then posed H_9 and H_{10}:

H_9 The difference between STASIS similarities for Experts-Source and NonEx.-Code is not statistically significant.

H_{10} The difference between LSS similarities for Experts-Source and NonEx.-Code is not statistically significant.

Using the Wilcoxon signed-rank test, we could not reject the null hypothesis for either hypothesis posed. These results indicate that experts do not necessarily use more source code keywords than non-experts when writing bug reports.

11.2. Discussion

We derive one main interpretation from these source code comparison results. The reason for the differences between expert and non-expert bug reports is not because the experts are using more keywords from source code. This does not necessarily mean that both experts and non-experts are using source code throughout their bug reports. It could easily be the case that both experts and

DOI: 10.1002/smr

non-experts use little to no source code when describing bugs to fix. Either way, these results mean that there must be other reasons for why expert and non-expert bug reports are not very similar.

One other possibility for the differences is that although experts are not necessarily using source code keywords specifically, they could still be using more technical terms that non-experts do not know. An example of this type of difference was seen during the qualitative analysis (see Section 4). Another possibility is that the experts tend to describe more about how the bug is being caused and how to fix it, while the non-experts tend to describe more about what the bug is actually doing and why it is a problem. The qualitative samples have an example of this possibility, as well. We believe that other possible reasons for differences such as these should be explored in future work.

12. RELATED WORK

Dit *et al.* present a technique for comparing the semantic similarity of bug reports, though this study did not explore the differences of expert and non-expert knowledge [20]. Also, Kochhar *et al.* present an analyses on the effect of different kinds of bugs on bug localization [51]. Considerable effort has been devoted to analyzing software bug reports for software maintenance tasks, motivated by studies of the high volume of bug reports constantly being submitted. For example, a Mozilla developer claimed that, "everyday, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [52]. Researchers' answers to this problem have analyzed past changes to the system to identify which developers possess relevant expertise [53, 54, 55, 56, 57, 58, 59]. Other approaches which take into account textual information from bug reports, commit logs, and source code have also been proposed [60, 52, 61, 62, 63, 64, 65].

Another issue resulting from the openness of bug repositories and the high volume of reports submitted is the presence of duplicate reports. Several techniques have been proposed to identify duplicates, and they typically leverage information retrieval techniques to compare the descriptions of bug reports and identify those that are textually similar [66, 67, 68, 69, 70, 71]. Note that in certain cases, other information sources, such as execution traces, are also used to identify duplicate bug reports [68]. Researchers have also conducted several empirical studies to investigate the impact of authorship on code quality [72, 73], developer contributions, and working habits [74, 75, 76, 77, 78], as well as other properties related to quality: time to fix issues, severity, and classification [79, 1, 80, 81, 82, 83, 84, 34, 85, 86].

13. CONCLUSION

We have presented a study of the textual difference between bug reports written by experts and non-experts, with experts being defined as persons who have contributed to the source code. We found that experts and non-experts wrote bug reports differently, as measured by textual similarity metrics. Our results support the thesis that expert knowledge affects the way in which people write bug reports. We also found that this difference is relevant for software maintenance researchers because it affects the performance of software maintenance research tools. Additionally, we found

DOI: 10.1002/smr

no evidence that the cause of these differences can be attributed to the increased usage of terms from the source code in expert bug reports.

ACKNOWLEDGEMENTS

The authors would like to thank the Eclipse Foundation and Mozilla Foundation for providing the repositories of data used in this paper.

REFERENCES

1. Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T. What makes a good bug report? *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, 2008; 308–318, doi:10.1145/1453101.1453146.
2. Bugde S, Nagappan N, Rajamani S, Ramalingam G. Global software servicing: Observational experiences at microsoft. *2013 IEEE 8th International Conference on Global Software Engineering* 2008; **0**:182–191, doi: http://doi.ieeecomputersociety.org/10.1109/ICGSE.2008.18.
3. Kagdi H, Poshyvanyk D. Who can help me with this change request? *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, 2009; 273–277, doi:10.1109/ICPC.2009.5090056.
4. Anvik J, Hiew L, Murphy GC. Who should fix this bug? *Proceedings of the 28th international conference on Software engineering*, ICSE '06, 2006; 361–370, doi:10.1145/1134285.1134336.
5. Gethers M, Kagdi H, Dit B, Poshyvanyk D. An adaptive approach to impact analysis from change requests to source code. *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, 2011; 540–543, doi:10.1109/ASE.2011.6100120.
6. Liu D, Marcus A, Poshyvanyk D, Rajlich V. Feature location via information retrieval based filtering of a single scenario execution trace. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007; 234–243, doi:10.1145/1321631.1321667.
7. Poshyvanyk D, Gethers M, Marcus A. Concept location using formal concept analysis and information retrieval. *ACM Trans. Softw. Eng. Methodol.* Feb 2013; **21**(4):23:1–23:34, doi:10.1145/2377656.2377660.
8. Lukins SK, Kraft NA, Etzkorn LH. Bug localization using latent dirichlet allocation. *Inf. Softw. Technol.* Sep 2010; **52**(9):972–990, doi:10.1016/j.infsof.2010.04.002.
9. Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.* Oct 2002; **28**(10):970–983, doi:10.1109/TSE.2002.1041053.
10. Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.* Jul 2002; **11**(3):309–346, doi:10.1145/567793.567795.
11. Jarzabek S. *Effective Software Maintenance and Evolution: A Reuse-Based Approach*. 2007.
12. Nosek JT, Palvia P. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance: Research and Practice* 1990; **2**(3):157–174, doi:10.1002/smr.4360020303.
13. Letovsky S. Cognitive processes in program comprehension. *Journal of Systems and Software* 1987; **7**(4):325 – 339, doi:http://dx.doi.org/10.1016/0164-1212(87)90032-X.
14. Levesque LL, Wilson JM, Wholey DR. Cognitive divergence and shared mental models in software development project teams. *Journal of Organizational Behavior* 2001; **22**(2):135–144, doi:10.1002/job.87.
15. Begel A, Simon B. Struggles of new college graduates in their first software development job. *39th SIGCSE technical symposium on Computer science education*, 2008; 226–230, doi:10.1145/1352135.1352218.
16. Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 2013; 712–721.
17. Hofman R. Behavioral economics in software quality engineering. *Empirical Softw. Engg.* Apr 2011; **16**(2):278–293, doi:10.1007/s10664-010-9140-x.
18. Ko A. Mining whining in support forums with frictionary. *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, 2012; 191–200, doi:10.1145/2212776.2212797.
19. Bettenburg N, Premraj R, Zimmermann T, Kim S. Duplicate bug reports considered harmful... really? *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 2008; 337–345, doi:10.1109/ICSM.2008.4658082.
20. Dit B, Poshyvanyk D, Marcus A. Measuring the semantic similarity of comments in bug reports. *Proc. of 1st STSM* 2008; **8**.

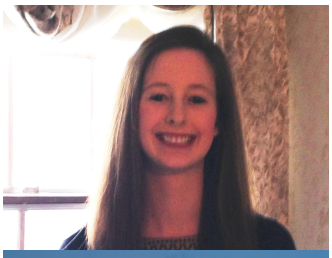
DOI: 10.1002/smr

21. Voorhees EM. Using wordnet to disambiguate word senses for text retrieval. *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93, 1993; 171–180, doi:10.1145/160688.160715.
22. Krovetz R, Croft WB. Lexical ambiguity and information retrieval. *ACM Trans. Inf. Syst.* Apr 1992; **10**(2):115–141, doi:10.1145/146802.146810.
23. Sanderson M, Van Rijsbergen CJ. The impact on retrieval effectiveness of skewed frequency distributions. *ACM Trans. Inf. Syst.* Oct 1999; **17**(4):440–465, doi:10.1145/326440.326447.
24. Li Y, Bandar ZA, McLean D. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Trans. on Knowl. and Data Eng.* Jul 2003; **15**(4):871–882, doi:10.1109/TKDE.2003.1209005.
25. Croft D, Coupland S, Shell J, Brown S. A fast and efficient semantic short text similarity metric. *Computational Intelligence (UKCI), 2013 13th UK Workshop on*, 2013; 221–227, doi:10.1109/UKCI.2013.6651309.
26. Zhou J, Zhang H, Lo D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. *Software Engineering (ICSE), 2012 34th International Conference on*, 2012; 14–24, doi:10.1109/ICSE.2012.6227210.
27. Panichella A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, 2013; 522–531.
28. Revelle M, Poshyvanyk D. An exploratory study on assessing feature location techniques. *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, 2009; 218–222, doi:10.1109/ICPC.2009.5090045.
29. Wu R, Zhang H, Kim S, Cheung SC. Relink: Recovering links between bugs and changes. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, 2011; 15–25, doi:10.1145/2025113.2025120.
30. Orso A, Apiwattanapong T, Law J, Rothermel G, Harrold MJ. An empirical comparison of dynamic impact analysis algorithms. *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, 2004; 491–500.
31. Ren X, Ryder BG, Stoerzer M, Tip F. Chianti: A change impact analysis tool for java programs. *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, 2005; 664–665, doi:10.1145/1062455.1062598.
32. Torchiano M, Ricca F. Impact analysis by means of unstructured knowledge in the context of bug repositories. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, 2010; 47:1–47:4, doi:10.1145/1852786.1852847.
33. Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, 2007; 9–9, doi:10.1109/PROMISE.2007.10.
34. Weiss C, Premraj R, Zimmermann T, Zeller A. How long will it take to fix this bug? *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, 2007; 1–, doi:10.1109/MSR.2007.13.
35. Kim S, Zhang H, Wu R, Gong L. Dealing with noise in defect prediction. *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, 2011; 481–490, doi:10.1145/1985793.1985859.
36. Miller GA. Wordnet: A lexical database for english. *Commun. ACM* Nov 1995; **38**(11):39–41, doi:10.1145/219717.219748.
37. Canfora G, Cerulo L. How software repositories can help in resolving a new change request. *In Workshop on Empirical Studies in Reverse Engineering*, 2005.
38. Linares-Vasquez M, Hossen K, Dang H, Kagdi H, Gethers M, Poshyvanyk D. Triaging incoming change requests: Bug or commit history, or code authorship? *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, 2012; 451–460, doi:10.1109/ICSM.2012.6405306.
39. ubrani D. Automatic bug triage using text categorization. *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering Knowledge Engineering*, 2004; 92–97.
40. Somasundaram K, Murphy GC. Automatic categorization of bug reports using latent dirichlet allocation. *Proceedings of the 5th India Software Engineering Conference*, ISEC '12, 2012; 125–130, doi:10.1145/2134254.2134276.
41. Tamrawi A, Nguyen TT, Al-Kofahi J, Nguyen TN. Fuzzy set-based automatic bug triaging (nier track). *Proceedings of the 33rd International Conference on Software Engineering*, 2011; 884–887, doi:10.1145/1985793.1985934.
42. Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. *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 '09, 2009; 111–120, doi:10.1145/1595696.1595715.
43. Dit B, Revelle M, Gethers M, Poshyvanyk D. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process* 2013; **25**(1):53–95, doi:10.1002/smr.567.

44. Marcus A, Sergeyev A, Rajlich V, Maletic J. An information retrieval approach to concept location in source code. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, 2004; 214–223, doi:10.1109/WCRE.2004.10.
45. Cleary B, Exton C, Buckley J, English M. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Softw. Engg.* Feb 2009; **14**(1):93–130, doi:10.1007/s10664-008-9095-3.
46. Rao S, Kak A. Retrieval from software libraries for bug localization: A comparative study of generic and composite text models. *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, 2011; 43–52, doi:10.1145/1985441.1985451.
47. Gay G, Haiduc S, Marcus A, Menzies T. On the use of relevance feedback in ir-based concept location. *Software Maintenance, 2009. IEEE International Conference on*, 2009; 351–360, doi:10.1109/ICSM.2009.5306315.
48. Poshyvanyk D, Gueheneuc YG, Marcus A, Antoniol G, Rajlich V. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Trans. Softw. Eng.* Jun 2007; **33**(6):420–432, doi:10.1109/TSE.2007.1016.
49. Lukins SK, Kraft NA, Etzkorn LH. Bug localization using latent dirichlet allocation. *Information and Software Technology* 2010; **52**(9):972 – 990, doi:http://dx.doi.org/10.1016/j.infsof.2010.04.002.
50. Smucker MD, Allan J, Carterette B. A comparison of statistical significance tests for information retrieval evaluation. *CIKM*, 2007; 623–632.
51. Kochhar PS, Tian Y, Lo D. Potential biases in bug localization: Do they matter? *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, ACM: New York, NY, USA, 2014; 803–814, doi:10.1145/2642937.2642997. URL `http://doi.acm.org/10.1145/2642937.2642997`.
52. Anvik J, Hiew L, Murphy GC. Who should fix this bug? *Proceedings of the 28th international conference on Software engineering*, ICSE '06, 2006; 361–370, doi:10.1145/1134285.1134336.
53. McDonald DW, Ackerman MS. Expertise recommender: a flexible recommendation system and architecture. *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, 2000; 231–240, doi:10.1145/358916.358994.
54. Minto S, Murphy GC. Recommending emergent teams. *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, 2007; 5–.
55. Mockus A, Herbsleb JD. Expertise browser: a quantitative approach to identifying expertise. *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, 2002; 503–512.
56. Anvik J, Murphy GC. Determining implementation expertise from bug reports. *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, 2007; 2–, doi:10.1109/MSR.2007.7.
57. Ma D, Schuler D, Zimmermann T, Sillito J. Expert recommendation with usage expertise. 2009; 535–538, doi:http://doi.ieeecomputersociety.org/10.1109/ICSM.2009.5306386.
58. Canfora G, Cerulo L. Supporting change request assignment in open source development. *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, 2006; 1767–1772, doi:10.1145/1141277.1141693.
59. Anvik J, Murphy G. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology* 2011; **20**(3):10.
60. Tamrawi A, Nguyen TT, Al-Kofahi JM, Nguyen TN. Fuzzy set and cache-based approach for bug triaging. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011; 365–375, doi:10.1145/2025113.2025163.
61. Song X, Tseng B, Lin CY, Sun MT. Expertisenet: Relational and evolutionary expert modeling. *User Modeling 2005*, *Lecture Notes in Computer Science*, vol. 3538, Ardissono L, Brna P, Mitrovic A (eds.). 2005; 99–108, doi:10.1007/11527886_14.
62. Matter D, Kuhn A, Nierstrasz O. Assigning bug reports using a vocabulary-based expertise model of developers. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, 2009; 131–140, doi:10.1109/MSR.2009.5069491.
63. Kagdi H, Gethers M, Poshyvanyk D, Hammad M. Assigning change requests to software developers. *Journal of Software Maintenance and Evolution: Research and Practice (JSME)* 2011; .
64. Linares-Vasquez M, Dang H, Hossen K, Kagdi H, Gethers M, Poshyvanyk D. Triaging incoming change requests: Bug or commit history, or code authorship? *28th IEEE International Conference on Software Maintenance (ICSM'12)*, Riva del Garda, Italy, 2012.
65. Čubranić D, Murphy G. Automatic bug triage using text categorization. *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.
66. Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, 2007; 499–510, doi:10.1109/ICSE.2007.32.

67. Hiew L. Assisted detection of duplicate bug reports. PhD Thesis, The University Of British Columbia 2006.
68. Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. *Proceedings of the 30th international conference on Software engineering*, ICSE '08, 2008; 461–470, doi:10.1145/1368088.1368151.
69. Jalbert N, Weimer W. Automated duplicate detection for bug tracking systems. *Dependable Systems and Networks With FTCS and DCC, 2008. IEEE International Conference on*, 2008; 52–61.
70. Sun C, Lo D, Wang X, Jiang J, Khoo SC. A discriminative model approach for accurate duplicate bug report retrieval. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, 2010; 45–54, doi:10.1145/1806799.1806811.
71. Sun C, Lo D, Khoo SC, Jiang J. Towards more accurate retrieval of duplicate bug reports. *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, 2011; 253–262, doi:10.1109/ASE.2011.6100061.
72. Rahman F, Devanbu P. Ownership, experience and defects: a fine-grained study of authorship. *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, 2011; 491–500, doi:10.1145/1985793.1985860.
73. Bird C, Nagappan N, Murphy B, Gall H, Devanbu P. Don't touch my code!: examining the effects of ownership on software quality. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, 2011; 4–14, doi:10.1145/2025113.2025119.
74. German DM. A study of the contributors of postgresql. *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, 2006; 163–164, doi:10.1145/1137983.1138022.
75. Tsunoda M, Monden A, Kakimoto T, Kamei Y, Matsumoto Ki. Analyzing oss developers' working time using mailing lists archives. *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, 2006; 181–182, doi:10.1145/1137983.1138031.
76. Weissgerber P, Pohl M, Burch M. Visual data mining in software archives to detect how developers work together. *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, 2007; 9–, doi: 10.1109/MSR.2007.34.
77. German DM. An empirical study of fine-grained software modifications. *Empirical Softw. Engg.* Sep 2006; **11**(3):369–393, doi:10.1007/s10664-006-9004-6.
78. Fischer M, Pinzger M, Gall H. Populating a release history database from version control and bug tracking systems. *International Conference on Software Maintenance (ICSM'03)*, 2003; 23–.
79. Hooimeijer P, Weimer W. Modeling bug report quality. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, 2007; 34–43, doi:10.1145/1321631.1321639.
80. Podgurski A, Leon D, Francis P, Masri W, Minch M, Sun J, Wang B. Automated support for classifying software failure reports. *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, 2003; 465–475.
81. Menzies T, Marcus A. Automated severity assessment of software defect reports. *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 2008; 346 –355, doi:10.1109/ICSM.2008.4658083.
82. Bettenburg N, Just S, Schröter A, WeißC, Premraj R, Zimmermann T. Quality of bug reports in eclipse. *2007 OOPSLA workshop on eclipse technology eXchange*, 2007; 21–25, doi:10.1145/1328279.1328284.
83. Bettenburg N, Premraj R, Zimmermann T, Kim S. Duplicate bug reports considered harmful ... really? *Software Maintenance, 2008. IEEE International Conference on*, 2008; 337 –345, doi:10.1109/ICSM.2008.4658082.
84. Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T. What makes a good bug report? *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, 2008; 308–318, doi:10.1145/1453101.1453146.
85. Kim S, Whitehead EJ Jr. How long did it take to fix bugs? *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, 2006; 173–174, doi:10.1145/1137983.1138027.
86. Marks L, Zou Y, Hassan AE. Studying the fix-time for bugs in large open source projects. *7th International Conference on Predictive Models in Software Engineering*, 2011; 11:1–11:8, doi:10.1145/2020390.2020401.

AUTHORS' BIOGRAPHIES



Paige Rodeghero is a Computer Science Ph.D. student at the University of Notre Dame working under Dr. Collin McMillan. She completed her Bachelor's Degree at Ball State University with a major in Computer Science and a minor in Dance Performance. Her research focuses primarily on program comprehension and source code summarization.

Da Huo is a graduate student at the University of Notre Dame. He received his bachelors degree in 2013 at DePauw University, with a double major in Computer Science and Mathematics. He worked with Dr. Collin McMillan during his first year with the study of the effects of expert knowledge on bug reports. Now, he is focusing on research projects about data preservation, ontology, and cognitive computing.

Tao Ding received a bachelors degree in Software Engineering from China, and a masters degree in Computer and Information Science from Gannon University, Pennsylvania, US. She is currently a Ph.D. student in the Department of Information Systems at the University of Maryland, Baltimore County. Her research interests are software engineering, natural language processing, and applied machine learning.

Dr. Collin McMillan is an Assistant Professor at the University of Notre Dame. He completed his Ph.D. in 2012 at the College of William and Mary, focusing on source code search and traceability technologies for program reuse and comprehension. Since joining Notre Dame, his work has focused on source code summarization and efficient reuse of executable code. Dr. McMillan's work has been recognized with the National Science Foundation's CAREER award.

Dr. Malcom Gethers was an Assistant Professor in the Information Systems Department at the University of Maryland, Baltimore County until 2014. He completed his Ph.D. in 2012 at William and Mary where he was a member of the SEMERU research group. His research interests include software engineering, software maintenance and evolution, mining of software repositories, feature location, software measurement, and traceability link recovery and management.

DOI: 10.1002/smr

Table I. Information on the bug repositories used for the textual similarities and developer recommendation studies.

Repository	Eclipse	Mozilla	*Total*
Number of Projects	50	30	80
Number of Developers	1517	1562	3079
Number of Filtered Developers	21	24	45
Number of Bug Reports	214k	196k	410k
Number of Duplicates	37k	231k	268k
Expert vs. Expert	16k	26k	42k
Expert vs. Non-Expert	5k	53k	58k
Non-Expert vs. Non-Expert	16k	152k	168k

Table II. Amount of reports from the two projects from BugLocator used for the feature location study (Historical Weighting Factor $\alpha = 0.2$)

Project	Type	Amount	# of source files
SWT	All	98	265
	Expert	65	201
	NonExpert	38	64
Eclipse	All	3070	10040
	Expert	2497	8447
	NonExpert	573	1593

Table III. Developer Recommendations for Eclipse and Mozilla. Training sample size was 9000. Testing sample size was 1000.

	Sample Size(Training/Testing)	Eclipse Accuracy	Mozilla Accuracy
All	0.8	0.68	
AllLong	0.75	0.70	
E	0.81	0.64	
EL	0.74	0.58	
NE	0.81	0.64	
NEL	0.75	0.58	

Table IV. Feature Location of SWT (Historical Weighting Factor $\alpha = 0.2$)

	MHR	Top 1	Top 5	Top 10	Top 20	Top 50
All	12.4	35/98 (36%)	64/98 (65%)	72/98 (73%)	84/98 (86%)	94/98 (96%)
AllLong	8.46	36/98 (37%)	69/98 (70%)	80/98 (82%)	89/98 (91%)	96/98 (98%)
E	8.49	23/65 (35%)	41/65 (63%)	46/65 (71%)	56/65 (86%)	64/65 (98%)
EL	10.63	21/65 (32%)	44/65 (68%)	50/65 (77%)	59/65 (91%)	63/65 (97%)
NE	21.68	11/33 (33%)	20/33 (61%)	24/33 (73%)	29/33 (88%)	29/33 (88%)
NEL	5.27	15/33 (45%)	22/33 (67%)	29/33 (88%)	30/33 (91%)	33/33 (100%)

Prepared using smrauth.cls
DOI: 10.1002/smr

Table V. Feature Location of Eclipse (Historical Weighting Factor $\alpha = 0.3$)

	MHR	Top 1	Top 5	Top 10	Top 20	Top 50
All	380.6	668/3070 (22%)	1231/3070 (40%)	1479/3070 (48%)	1709/3070 (56%)	2039/3070 (66%)
AllLong	135.46	987/3070 (32%)	1726/3070 (56%)	2030/3070 (66%)	2265/3070 (74%)	2518/3070 (82%)
E	393.77	558/2497 (22%)	1726/2497 (41%)	2030/2497 (49%)	2265/2497 (56%)	2518/2497 (67%)
EL	129.28	795/2497 (32%)	1427/2497 (57%)	1669/2497 (67%)	1859/2497 (74%)	2048/2497 (81%)
NE	426.58	107/573 (19%)	180/573 (31%)	227/573 (40%)	276/573 (48%)	344/573 (60%)
NEL	211	165/573 (29%)	275/573 (48%)	337/573 (59%)	393/573 (69%)	443/573 (77%)

Table VI. Mean Highest Rank of SWT with α as the Historical Weighting Factor. Means do not include $\alpha = 1$.

α	All	AllLong	E	EL	NE	NEL
0	13.92	9.38	6.56	11.15	22.51	5.88
0.1	13.07	8.82	9.01	10.72	22.06	5.39
0.2	12.44	8.46	8.49	10.63	21.58	5.27
0.3	11.88	8.47	8.45	10.75	21.33	5.3
0.4	11.71	8.8	8.28	11.09	21.18	5.3
0.5	11.83	9.26	8.29	11.86	21.27	6.03
0.6	12.01	9.84	8.49	12.65	21.33	6.42
0.7	12.17	10.71	8.63	13.33	21.39	7.36
0.8	12.45	11.72	9.01	14.36	21.36	8.24
0.9	13.08	13.59	9.71	15.23	21.36	9.45
1	140.52	88.91	156.32	90.32	200.06	160.58
Mean(0-0.9)	12.46	9.91	8.49	12.18	21.54	6.46

Table VII. Mean Highest Rank of Eclipse with α as the Historical Weighting Factor. Means do not include $\alpha = 1$.

α	All	AllLong	E	EL	NE	NEL
0	420.96	149.21	417.5	149.26	477.25	152.93
0.3	393.05	144.44	384.02	129.71	508.9	134.14
0.5	380.43	129.27	417.94	141.53	550.61	147.06
1	2857.71	2293.75	2885.21	2230.18	3894.08	3436.27
Mean(0,0.3,0.5)	398.15	140.97	406.49	140.17	512.25	144.71

DOI: 10.1002/smr

Table VIII. Statistical summary of the results for AQ_1. Mann-Whitney test values are U, U_{expt}, and U_{vari}. Decision criteria is p. A "Sample" is a similarity value for one pair of duplicate bug reports.

H	Metric	Method Area	Samples	$\tilde{x}$	μ	Vari.	U	U_{expt}	U_{vari}	p	Decision
H_1	STASIS	Expert-Expert NonEx.-NonEx.	44717 169305	0.607 0.619	0.607 0.619	0.020 0.011	3.450×10^9	-5.096×10^8	1.350×10^{14}	<0.0001	Reject
H_2	STASIS	Expert-Expert Expert-NonEx.	44717 59565	0.607 0.597	0.607 0.597	0.020 0.008	1.346×10^9	-8.166×10^8	2.315×10^{13}	<0.0001	Reject
H_3	STASIS	NonEx.-NonEx. Expert-NonEx.	169305 59565	0.619 0.597	0.619 0.597	0.011 0.008	5.620×10^9	7.474×10^8	1.923×10^{14}	<0.0001	Reject
H_4	LSS	Expert-Expert NonEx.-NonEx.	44644 169264	0.979 0.985	0.968 0.978	0.002 0.001	2.963×10^9	-5.167×10^8	1.347×10^{14}	<0.0001	Reject
H_5	LSS	Expert-Expert Expert-NonEx.	44644 59526	0.979 0.983	0.968 0.975	0.002 0.001	1.167×10^9	-8.187×10^8	2.307×10^{13}	<0.0001	Reject
H_6	LSS	NonEx.-NonEx. Expert-NonEx.	169264 59526	0.985 0.983	0.978 0.975	0.001 0.001	5.583×10^9	7.428×10^8	1.921×10^{14}	<0.0001	Reject

Table IX. Mann-Whitney Test Result of Ranking Score in Developer Recommendation Test. Critical Z value is 1.949964.

Project	H	Comparison	Sample	$\tilde{x}$	μ	Vari.	U	Z	p(two-tail)	Decision
Mozilla	H_7	E. NE.	1000 1000	3.262 3.248	3.262 3.248	24.013 24.111	497995	-0.155	0.877	Fail to reject
		EL. NEL.	1000 1000	3.985 3.99	3.985 3.99	30.764 30.654	499633	-0.028	0.978	Fail to reject
Eclipse	H_7	E. NE.	1000 1000	1.716 1.72	1.665 1.716	11.08386 11.795	499927.5	-0.006	0.996	Fail to reject
		EL. NEL.	1000 1000	2.223 2.215	2.198 15.162	15.198 2.223	459685.5	0.001	0.02	Reject

Table X. Mann-Whitney Test Results of Similarity Score in Feature Location Test. Critical Z value is 1.949964.

Project	H	Comparison	Sample	$\tilde{x}$	μ	Vari.	U	p(two-tail)	Decision	
SWT	H_8	E. NE.	201 64	0.269 0.387	0.269 0.387	0.098 0.096	4555.5	-3.514	<0.001	Reject
		EL. NEL.	201 64	0.383 0.506	0.383 0.506	0.087 0.074	4864	-2.936	0.003	Reject
Eclipse	H_8	E. NE.	8447 1593	0.260 0.278	0.26 0.278	0.065 0.056	6489003.5	-2.253	0.024	Reject
		EL. NEL.	8447 1593	0.335 0.352	0.335 0.352	0.067 0.054	6311668	-3.924	<0.001	Reject

Table XI. Statistical summary of the results for RQ_7. Mann-Whitney test value is U, and decision criteria is p. A "Sample" is a similarity value for one bug report paired with its relevant source code. Critical Z value is 1.96.

H	Metric	Method Area	Samples	$\tilde{x}$	μ	Vari.	U	Z	$p(two-tailed)$	Decision
H_9	STASIS	Expert-Code NonEx.-Code	2497 573	0.3968 0.4074	0.4023 0.4073	0.008159 0.008041	753536	1.91054	0.0550236	Fail to Reject
H_{10}	LSS	Expert-Code NonEx.-Code	2497 573	0.7931 0.7880	0.8205 0.8134	0.0134 0.01367	741614	1.28853	0.1974036	Fail to Reject

Prepared using smrauth.cls
DOI: 10.1002/smr