An Empirical Study of the Effects of Expert Knowledge on Bug Reports

Da Huo*, Tao Ding†, Collin McMillan*, and Malcom Gethers†
*Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46545
Email: {dhuo, cmc}@nd.edu
†Information Systems Department
University of Maryland Baltimore County, Baltimore, MD 21250
Email: {tding1027, mgethers}@umbc.edu

***Abstract*—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 significance difference exists, and that this difference has a significant impact on the results from a state-of-the-art feature location tool. Our recommendation is that researchers evaluate maintenance tools using different sets of bug reports for experts and non-experts.**

I. 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 tracability [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, to communicate those failures to maintainers. Or, 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 text 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 compute the textual difference between 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 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 that textual differences 1) did not affect performance of the bug triage technique that is based on text classification for different developers, but 2) did affect performance of the feature location technique based on textual similarity analysis.

II. 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 difference in bug reports written by persons with expert knowledge about the program, and without that knowledge. At present, this textual difference 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]. Tracebility 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.

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

A. 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 is 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 short natural language documents [24], [25], such as bug reports.

B. Developer Recommendation

Open source software is developed by 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.

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 the 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 machine and an unsupervised generative model based on LDA. Tamrawi proposed Bugzie for triaging based on fuzzy set-based modeling of 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 at al. [4] utilized support vector machine to classify bug and improve precision up to 64% by refining data set, which filtered out developers who did not make enough contribution in recent 3 months.

C. 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 analysis 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] present an approach based on a complimentary information retrieval method which used information flow and co-occurrence information derived from non source code artifact 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 [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, and the technique has significant advantages over LSI and pLSI. Rao and Kak [46] found the performance of VSM model in bug localization is worse than SUM but better than LDA and LSI.

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 Vector space model to rank all files based on text similarity between the initial bug report and the source code ($rVSMScore$), and 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.

$$FinalScore = (1-\alpha)*N(rVSMScore)+\alpha*N(SimiScore) \tag{1}$$

, 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 empirical study section, we use BugLocator to evaluate how textual difference will affect feature location.

IV. Empirical Study Design

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

A. 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 What is the degree of textual similarity between bug reports written by experts and duplicates of those bug reports written by non-experts?

RQ_2 What is the degree of textual similarity among duplicate bug reports written by experts?

RQ_3 What is the degree of textual similarity among duplicate bug reports written by non-experts?

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 highly-cited software maintenance tool when the inputs to that tool are bug reports written solely by experts?

RQ_5 What is the performance of a highly-cited 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 highly-cited 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. 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 the distinction as well (see Section II).

B. 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 language.

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

C. 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.

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 III-A) 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.

2) Developer Recommendation: The second empirical study is to evaluate how textual differences impact textual analysis based automated approaches to bug triaging. In this study, we collect bug reports from two bug repositories: Eclipse [1] and Mozilla[2]. Table I shows the number of bug reports, period and the number of developers involved. The time period starts from the time when the first bug was fixed. We extracted those bugs which were fixed and tagged as ASSIGNED, RESOLVED, FIXED, VERIFIED, and CLOSE. 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. Description of bugs includes two parts in each report: the title which briefly summarized the issue, and long description which provided details about the bug. In the study, we study the impact of textual difference considering the case where only the summary is used as well as the case where the summary and short description is used. Two sub groups are produced for each group: bug description containing title only (E, NE), or bug description containing title and comment 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 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 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 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. We search the top 3 recommended developers based on probability, if one is correct developer, we consider the bug as being assigned correctly. In this study, we evaluate the impact of textual difference on the accuracy of the developer recommendation technique in order to answer RQ_4, RQ_5, and RQ_6.

TABLE I. INFORMATION OF TWO PROJECTS USED IN STUDIES

	# Bug Report	Period	# of Developer	# of Developer after refine
Eclipse	214560	10/2001-07/2002	1517	21
Mozilla	195963	04/1998-04/2003	1562	24

3) Feature Location: In this empirical study, we evaluated how the textual difference impacts a textual analysis based feature location technique, namely BugLocator [26]. We used the sample data set of SWT (98 bugs) and Eclipse (3070 bugs) from BugLocator[3]. Note that this is a subset of the data used in our other studies. 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 paper [26], when α equals 0.2 for SWT and α equals 0.3 for Eclipse, the results are the best, 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.

TABLE II. SIZE OF BUG REPORT FROM TWO PROJECTS($\alpha = 0.2$)

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

D. 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.

$$Accuracy = \frac{\#\ correct\ predicted\ bug}{\#\ predicted\ bug} \quad (2)$$

[1]https://bugs.eclipse.org/bugs/

[2]https://bugzilla.mozilla.org/

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

To measure the effectiveness of feature location method, we use the following metrics in 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 results contains at least one file at which the bug should be fixed, we consider the bug located.
- MHR (Mean Highest Rank), which is mean of highest ranks for all known relevant files when testing n bug reports.

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

E. Research Subjects

We used two bug repositories for our data. We obtained the bug reports via public databases for Eclipse[4] and Mozilla[5]. These repositories are extensive and include several years of data over several versions of different software products, including contributions by many programmers and users. Table III presents details about the repositories. In total we extraced over 400,000 bug reports, of which 268,000 were duplicates suitable for our empirical study.

TABLE III. BUG REPOSITORIES USED FOR THIS STUDY.

Repository	Eclipse	Mozilla	*Total*
Number of Projects	50	30	80
Number of Bug Reports	200k	200k	400k
Number of Duplicates			
Expert vs. Expert	17k	27k	42k
Expert vs. Non-Expert	5k	53k	58k
Non-Expert vs. Non-Expert	16k	152k	168k

In the third study, a subset of the data in Table II is downloaded from the project site of BugLocator, which was experimental data used in paper [4]. The sample data is also subset of Table III.

F. 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 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 similaries 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.

G. 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 guarentee that our results are consistent for every repository. 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 datasets on the tool, and can draw a conclusion related to these datasets 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.

[4]https://bugs.eclipse.org/bugs/

[5]https://bugzilla.mozilla.org/

H. 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/

V. TEXTUAL SIMILARITY RESULTS

This section describes the results of our Empirical Study for RQ_1, RQ_2, RQ_3, and AQ_1. 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.

A. 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.

B. Detailed Results - RQ_1

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 X 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 III-A). This is an important distinction because the repositories contain bug reports of varying lengths.

C. Detailed Results - RQ_2

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 X and Figure 1.

D. Detailed Results - RQ_3

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 X and Figure 1.

VI. Evaluation of Impacts on Software Maintenance

The section describe the results of our two empirical studies: developer recommendation and feature location for answering RQ_4,RQ_5,RQ_6 and AQ_2.

TABLE IV. Developer Recommendation of Eclipse

	Sample Size(Training/Testing)	Accuracy
All	9000/1000	0.8
AllLong	9000/1000	0.75
E	9000/1000	0.81
EL	9000/1000	0.74
NE	9000/1000	0.81
NEL	9000/1000	0.75

TABLE V. Developer Recommendation of Mozilla

	Sample Size(Training/Testing)	Accuracy
All	9000/1000	0.68
AllLong	9000/1000	0.70
E	9000/1000	0.64
EL	9000/1000	0.58
NE	9000/1000	0.64
NEL	9000/1000	0.58

A. *Detailed Results - RQ_4*

In developer recommendation study, when we used bug description without comments and only expert bugs are considered, for Eclipse in Table IV, accuracy for development recommendation in Eclipse is 0.81, for Mozilla Table V shows accuracy is 0.64. When comments are added in bug description, both accuracy is decreased 0.75 and 0.58 respectively.

In Feature location study, for SWT when α equals 0.2, the accuracy is 35% at top 1 when we used bug description without comments(E), which is higher than accuracy(32%) when comments is considered(EL). As 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($\alpha = 1$),that means longer description is helpful when only considering similarity with bugs that had fixed before.

TABLE VI. Feature Location of SWT ($\alpha = 0.2$)

	MFR	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%)

B. *Detailed Results - RQ_5*

In developer recommendation study, when we used bug description without comments and only non-expert bugs are considered, for Eclipse in Table IV, accuracy for development recommendation in Eclipse is 0.74, for Moziila in Table V, accuracy is 0.64. When comments is added in bug description, both accuracy is decreased 0.75 and 0.58 respectively.

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

TABLE VII. Feature Location of Eclipse ($\alpha = 0.3$)

	MFR	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%)

C. *Detailed Results - RQ_6*

From Table IX MHR of Eclipse, all the bug reports types with long description (AllLong, EL and NEL) have relevantly equal ranking performance which is much better than the types without long description (AllLong, E and NE). And also the NE type (non-expert bug description) has the worst MHR performance among all types.

TABLE VIII. MHR of SWT

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 IX. MHR of Eclipse

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(0-0.5)	398.15	140.97	406.49	140.17	512.25	144.71

VII. Comparison Analysis

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

A. *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.

TABLE X. 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.	T	T_{expt}	T_{vari}	p	Decision
H_1	STASIS	Expert-Expert NonEx.-NonEx.	44717 169305	0.609 0.624	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	0.609 0.609	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	0.624 0.609	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

The procedure we used to obtain the evidence is as follows. Consider the statistical data in Table X. 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 The difference between STASIS similarities for Experts-Experts and NonEx.-NonEx. is not statistically signficant.

H_2 The difference between STASIS similarities for Experts-Experts and Experts-NonEx. is not statistically signficant.

H_3 The difference between STASIS similarities for NonEx.-NonEx. and Experts-NonEx. is not statistically signficant.

H_4 The difference between LSS similarities for Experts-Experts and NonEx.-NonEx. is not statistically signficant.

H_5 The difference between LSS similarities for Experts-Experts and Experts-NonEx. is not statistically signficant.

H_6 The difference between LSS similarities for NonEx.-NonEx. and Experts-NonEx. is not statistically signficant.

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

B. AQ$_2$ - Effects on Tools

We compare the mean for ranking score for correct developer from different groups (All, E, NE). Each groups contains 1000 bugs in our study. To find statistically significant evidence for RQ$_3$,RQ$_4$,RQ$_5$ in developer recommendation, we produced three hypotheses:

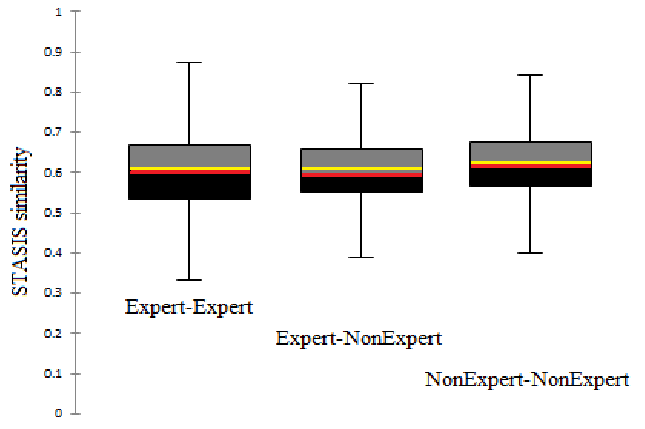
H_7 The difference between accuracy in developer recommendation from Experts reports and NonEx. reports is not statistically significant.

We perform a u-test for H_7 in two projects: Eclipse and Mozilla. We rejected hypothesis when p exceeded 0.5. Table XI presents the hypothesis in Eclipse when comments is considered in bug report.

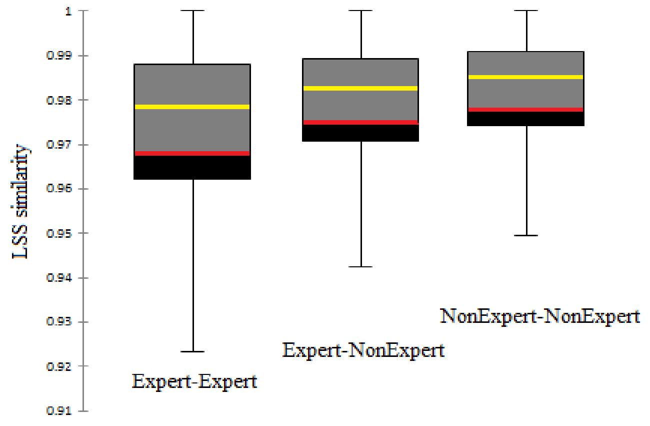
We also compare the mean for final similarity(FinalScore Section III) between bug report and relevant source code files from each group. To find statistically significant evidence for RQ$_3$,RQ$_4$,RQ$_5$ in feature location, we produced three hypotheses:

H_8 The difference between final similarities in feature location study for Experts reports and NonEx. reports is not statistically significant.

We performed a u-tests for H_8 to compare Expert and NonEx (see Section IV-F). At .05 level, we reject hypothesis H_8.



(a)



(b)

Fig. 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

TABLE XI. MANN-WHITNEY TEST RESULT OF RANKING SCORE IN DEVELOPER RECOMMENDATION TEST

Project	H	Comparison	Sample	$\tilde{x}$	μ	Vari.	U	Z	$Z_{critical}$	p(two-tail)	Decision
Mozilla	H_7	E.	1000	3.262	3.262	24.013	497995	-0.155	1.949964	0.877	Fail to reject
		NE.	1000	3.248	3.248	24.111					
		EL.	1000	3.985	3.985	30.764	499633	-0.028	1.949964	0.978	Fail to reject
		NEL.	1000	3.99	3.99	30.654					
Eclipse	H_7	E.	1000	1.716	1.665	11.08386	499927.5	-0.006	1.949964	0.996	Fail to reject
		NE.	1000	1.72	1.716	11.795					
		EL.	1000	2.223	2.198	15.198	459685.5	0.001	1.949964	0.02	Reject
		NEL.	1000	2.215	15.162	2.223					

TABLE XII. MANN-WHITNEY TEST RESULTS OF SIMI SCORE IN FEATURE LOCATION TEST

Project	H	Comparison	Sample	$\tilde{x}$	μ	Vari.	U	Z	$Z_{critical}$	p(two-tail)	Decision
SWT	H_8	E.	201	0.269	0.269	0.098	4555.5	-3.514	1.949964	<0.001	Reject
		NE.	64	0.387	0.387	0.096					
		EL.	201	0.383	0.383	0.087	4864	-2.936	1.949964	0.003	Reject
		NEL.	64	3.99	3.99	30.654					
Eclipse	H_8	E.	8447	0.260	0.26	0.065	6489003.5	-2.253	1.949964	0.024	Reject
		NE.	1593	0.278	0.278	0.056					
		EL.	8447	0.335	0.335	0.067	6311668	-3.924	1.949964	<0.001	Reject
		NEL.	1593	0.352	0.352	0.054					

VIII. 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.

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. This means that the similarity metrics did not agree, which leaves us unable to draw a strong conclusion about the textual similarities.

A further finding 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 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.

IX. 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]. 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" [51]. Researchers' answers to this problem have analyzed past changes to the system to identify which developers possess relevant expertise [52], [53], [54], [55], [56], [57], [58]. Other approaches which take into account textual information from bug reports, commit logs, and source code have also been proposed [59], [51], [60], [61], [62], [63], [64]. 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 [65], [66], [67], [68], [69], [70]. Note that in certain cases, other information sources, such as execution traces, are also used to identify duplicate bug reports [67]. Research have also conducted several empirical studies to investigate the impact of authorship on code quality [71], [72], developer contributions and working habits [73], [74], [75], [76], [77] as well as other properties related to quality, time to fix issues, severity and classification [78], [1], [79], [80], [81], [82], [83], [34], [84], [85].

X. 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 has 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.

ACKNOWLEDGMENT

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

REFERENCES

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16, 2008, pp. 308–318.

[2] S. Bugde, N. Nagappan, S. Rajamani, and G. Ramalingam, "Global software servicing: Observational experiences at microsoft," *2013 IEEE 8th International Conference on Global Software Engineering*, vol. 0, pp. 182–191, 2008.

[3] H. Kagdi and D. Poshyvanyk, "Who can help me with this change request?" in *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, 2009, pp. 273–277.

[4] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06, 2006, pp. 361–370.

[5] M. Gethers, H. Kagdi, B. Dit, and D. Poshyvanyk, "An adaptive approach to impact analysis from change requests to source code," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11, 2011, pp. 540–543.

[6] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 234–243.

[7] D. Poshyvanyk, M. Gethers, and A. Marcus, "Concept location using formal concept analysis and information retrieval," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, pp. 23:1–23:34, Feb. 2013.

[8] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 972–990, Sep. 2010.

[9] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, Oct. 2002.

[10] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

[11] S. Jarzabek, *Effective Software Maintenance and Evolution: A Reuse-Based Approach*, 2007.

[12] J. T. Nosek and P. Palvia, "Software maintenance management: Changes in the last decade," *Journal of Software Maintenance: Research and Practice*, vol. 2, no. 3, pp. 157–174, 1990.

[13] S. Letovsky, "Cognitive processes in program comprehension," *Journal of Systems and Software*, vol. 7, no. 4, pp. 325 – 339, 1987.

[14] L. L. Levesque, J. M. Wilson, and D. R. Wholey, "Cognitive divergence and shared mental models in software development project teams," *Journal of Organizational Behavior*, vol. 22, no. 2, pp. 135–144, 2001.

[15] A. Begel and B. Simon, "Struggles of new college graduates in their first software development job," in *39th SIGCSE technical symposium on Computer science education*, 2008, pp. 226–230.

[16] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 712–721.

[17] R. Hofman, "Behavioral economics in software quality engineering," *Empirical Softw. Engg.*, vol. 16, no. 2, pp. 278–293, Apr. 2011.

[18] A. Ko, "Mining whining in support forums with frictionary," in *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '12, 2012, pp. 191–200.

[19] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful... really?" in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 2008, pp. 337–345.

[20] B. Dit, D. Poshyvanyk, and A. Marcus, "Measuring the semantic similarity of comments in bug reports," *Proc. of 1st STSM*, vol. 8, 2008.

[21] E. M. Voorhees, "Using wordnet to disambiguate word senses for text retrieval," in *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '93, 1993, pp. 171–180.

[22] R. Krovetz and W. B. Croft, "Lexical ambiguity and information retrieval," *ACM Trans. Inf. Syst.*, vol. 10, no. 2, pp. 115–141, Apr. 1992.

[23] M. Sanderson and C. J. Van Rijsbergen, "The impact on retrieval effectiveness of skewed frequency distributions," *ACM Trans. Inf. Syst.*, vol. 17, no. 4, pp. 440–465, Oct. 1999.

[24] Y. Li, Z. A. Bandar, and D. McLean, "An approach for measuring semantic similarity between words using multiple information sources," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, no. 4, pp. 871–882, Jul. 2003.

[25] D. Croft, S. Coupland, J. Shell, and S. Brown, "A fast and efficient semantic short text similarity metric," in *Computational Intelligence (UKCI), 2013 13th UK Workshop on*, Sept 2013, pp. 221–227.

[26] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *Software Engineering (ICSE), 2012 34th International Conference on*, June 2012, pp. 14–24.

[27] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 522–531.

[28] M. Revelle and D. Poshyvanyk, "An exploratory study on assessing feature location techniques," in *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, May 2009, pp. 218–222.

[29] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11, 2011, pp. 15–25.

[30] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold, "An empirical comparison of dynamic impact analysis algorithms," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04, 2004, pp. 491–500.

[31] X. Ren, B. G. Ryder, M. Stoerzer, and F. Tip, "Chianti: A change impact analysis tool for java programs," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05, 2005, pp. 664–665.

[32] M. Torchiano and F. Ricca, "Impact analysis by means of unstructured knowledge in the context of bug repositories," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10, 2010, pp. 47:1–47:4.

[33] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, May 2007, pp. 9–9.

[34] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, 2007, pp. 1–.

[35] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, 2011, pp. 481–490.

[36] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.

[37] G. Canfora and L. Cerulo, "How software repositories can help in resolving a new change request," in *In Workshop on Empirical Studies in Reverse Engineering*, 2005.

[38] M. Linares-Vasquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Sept 2012, pp. 451–460.

[39] D. ubrani, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering Knowledge Engineering*, 2004, pp. 92–97.

[40] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation," in *Proceedings of the 5th India Software Engineering Conference*, ser. ISEC '12, 2012, pp. 125–130.

[41] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 884–887.

[42] G. Jeong, S. Kim, and T. Zimmermann, "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*, ser. ESEC/FSE '09, 2009, pp. 111–120.

[43] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[44] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic, "An information retrieval approach to concept location in source code," in *WCRE*, 2004.

[45] B. Cleary, C. Exton, J. Buckley, and M. English, "An empirical analysis of information retrieval based concept location techniques in software comprehension," *Empirical Softw. Engg.*, vol. 14, no. 1, pp. 93–130, Feb. 2009.

[46] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, 2011, pp. 43–52.

[47] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location," in *Software Maintenance, 2009. IEEE International Conference on*, 2009, pp. 351–360.

[48] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 420–432, Jun. 2007.

[49] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972 – 990, 2010.

[50] M. D. Smucker, J. Allan, and B. Carterette, "A comparison of statistical significance tests for information retrieval evaluation," in *CIKM*, 2007, pp. 623–632.

[51] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06, 2006, pp. 361–370.

[52] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, ser. CSCW '00, 2000, pp. 231–240.

[53] S. Minto and G. C. Murphy, "Recommending emergent teams," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, 2007, pp. 5–.

[54] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02, 2002, pp. 503–512.

[55] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, 2007, pp. 2–.

[56] D. Ma, D. Schuler, T. Zimmermann, and J. Sillito, "Expert recommendation with usage expertise," vol. 0, 2009, pp. 535–538.

[57] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proceedings of the 2006 ACM symposium on Applied computing*, ser. SAC '06, 2006, pp. 1767–1772.

[58] J. Anvik and G. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, p. 10, 2011.

[59] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "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*, 2011, pp. 365–375.

[60] X. Song, B. Tseng, C.-Y. Lin, and M.-T. Sun, "Expertisenet: Relational and evolutionary expert modeling," in *User Modeling 2005*, ser. Lecture Notes in Computer Science, L. Ardissono, P. Brna, and A. Mitrovic, Eds., 2005, vol. 3538, pp. 99–108.

[61] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR '09, 2009, pp. 131–140.

[62] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 2011.

[63] M. Linares-Vasquez, H. Dang, K. Hossen, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *28th IEEE International Conference on Software Maintenance (ICSM'12)*, Riva del Garda, Italy, 2012.

[64] D. Čubranić and G. Murphy, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

[65] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07, 2007, pp. 499–510.

[66] L. Hiew, "Assisted detection of duplicate bug reports," Ph.D. dissertation, The University Of British Columbia, 2006.

[67] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08, 2008, pp. 461–470.

[68] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. IEEE International Conference on*, 2008, pp. 52–61.

[69] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, 2010, pp. 45–54.

[70] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11, 2011, pp. 253–262.

[71] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, 2011, pp. 491–500.

[72] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ser. ESEC/FSE '11, 2011, pp. 4–14.

[73] D. M. German, "A study of the contributors of postgresql," in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06, 2006, pp. 163–164.

[74] M. Tsunoda, A. Monden, T. Kakimoto, Y. Kamei, and K.-i. Matsumoto, "Analyzing oss developers' working time using mailing lists archives," in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06, 2006, pp. 181–182.

[75] P. Weissgerber, M. Pohl, and M. Burch, "Visual data mining in software archives to detect how developers work together," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, 2007, pp. 9–.

[76] D. M. German, "An empirical study of fine-grained software modifications," *Empirical Softw. Engg.*, vol. 11, no. 3, pp. 369–393, Sep. 2006.

[77] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *International Conference on Software Maintenance (ICSM'03)*, 2003, pp. 23–.

[78] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ser. ASE '07, 2007, pp. 34–43.

[79] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03, 2003, pp. 465–475.

[80] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 28 2008-oct. 4 2008, pp. 346 –355.

[81] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," in *2007 OOPSLA workshop on eclipse technology eXchange*, 2007, pp. 21–25.

[82] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?" in *Software Maintenance, 2008. IEEE International Conference on*, 2008, pp. 337 –345.

[83] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16, 2008, pp. 308–318.

[84] S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06, 2006, pp. 173–174.

[85] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 11:1–11:8.