Noname manuscript No.
(will be inserted by the editor)

Revisiting the Performance of Automated Approaches for the Retrieval of Duplicate Reports in Issue Tracking Systems that Perform Just-in-Time Duplicate Retrieval

Mohamed Sami Rakha · Cor-Paul Bezemer · Ahmed E. Hassan

the date of receipt and acceptance should be inserted later

Abstract Issue tracking systems (ITSs) allow software end-users and developers to file issue reports and change requests. Reports are frequently duplicately filed for the same software issue. The retrieval of these duplicate issue reports is a tedious manual task. Prior research proposed several automated approaches for the retrieval of duplicate issue reports. Recent versions of ITSs added a feature that does basic retrieval of duplicate issue reports at the filing time of an issue report in an effort to avoid the filing of duplicates as early as possible.

This paper investigates the impact of this just-in-time duplicate retrieval on the duplicate reports that end up in the ITS of an open source project. In particular, we study the differences between duplicate reports for open source projects before and after the activation of this new feature. We show how the experimental results of prior research would vary given the new data after the activation of the just-in-time duplicate retrieval feature. We study duplicate issue reports from the Mozilla-Firefox, Mozilla-Core and Eclipse-Platform projects. In addition, we compare the performance of the state of the art of the automated retrieval of duplicate reports using two popular approaches (i.e., BM25F and REP).

We find that duplicate issue reports after the activation of the just-in-time duplicate retrieval feature are less textually similar, have a greater identification delay and require more discussion to be retrieved as duplicate reports than duplicates before the activation of the feature. Prior work showed that REP outperforms BM25F in terms of Recall rate and Mean average precision. We observe that the performance gap between BM25F and REP becomes even larger after the activation of the just-in-time duplicate retrieval feature. We

Mohamed Sami Rakha, Cor-Paul Bezemer, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL), School of Computing
Queen's University
Kingston, Ontario, Canada
E-mail: {rakha, bezemer, ahmed}@cs.queensu.ca

recommend that future studies focus on duplicates that were reported after the activation of the just-in-time duplicate retrieval feature as these duplicates are more representative of future incoming issue reports and therefore, give a better representation of the future performance of proposed approaches.

1 Introduction

Issue tracking systems (ITSs) [21] are commonly used to manage the maintenance and support processes of a software project. An ITS provides a communication platform for developers and users to report and discuss encountered software issues. An issue report can represent a software bug, a new feature or an improvement request.

An issue report can be reported multiple times, leading to duplicate issue reports. For instance, from the data that we present later in this paper, we calculate that 11-19% of all reported issues are duplicate reports in the Eclipse Foundation and Mozilla Foundation ITSs. Manually retrieving duplicate issue reports is a tedious task for developers [33]. Hence, prior research proposed several automated approaches to retrieve duplicate issue reports [6, 7, 23, 24, 29, 36, 40, 44]. These approaches make use of information retrieval techniques, that make use of the textual information that is available in issue reports, to suggest a list of possible candidate duplicates for each newly-reported issue [38].

To evaluate the performance of these approaches, existing duplicate issue reports from popular ITSs are usually used. The existing duplicate reports are treated equally during the performance evaluation. However, recent versions of popular ITSs (e.g., Bugzilla 4.0 [1] and Jira 6.0 [2]), add a new feature that displays a list of candidate duplicates while a user is filing an issue (*the just-in-time duplicate retrieval feature*). The activation of the just-in-time (JIT) duplicate retrieval feature impacts the assumption that all the duplicate reports selected for the evaluation of automated approaches are equal.

Our paper investigates the impact of the just-in-time duplicate retrieval feature on the data that is used for the experimental evaluation of automated approaches for the retrieval of duplicate issue reports for open source projects. We study duplicate issue reports from three large open source projects (Mozilla-Firefox, Mozilla-Core and Eclipse-Platform). We compare duplicate issue reports that were reported before (*before-JIT duplicates*) and after (*after-JIT duplicates*) the activation of the just-in-time duplicate retrieval feature along two dimensions: 1) their textual similarity and 2) the needed manual effort to retrieve duplicates in terms of identification delay and number of discussion comments. In addition, we evaluate the performance of two popular approaches (BM25F [34] and REP [40]) for the automated retrieval of duplicate reports on before and after-JIT duplicates. In particular, we explore the following research questions:

RQ1: How do the characteristics of duplicates differ before and after the activation of the just-in-time duplicate retrieval feature?

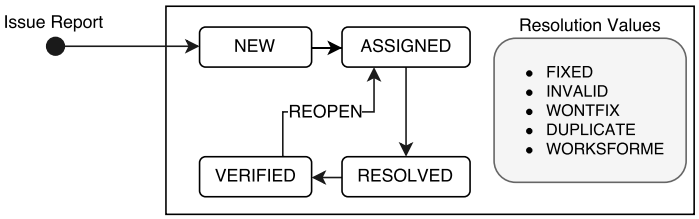


Fig. 1: The typical life cycle of an issue report.

There is a lower proportion of duplicate reports after the activation of the just-in-time duplicate retrieval feature. In addition, after-JIT duplicates are significantly less textually similar and need more effort (i.e., they have a greater identification delay and need more discussion comments) to be manually identified than before-JIT duplicates.

RQ2: How does the just-in-time duplicate retrieval feature impact the performance evaluation of state of the art automated approaches for the retrieval of duplicate issue reports?
The studied approaches (BM25F and REP) achieve a significantly lower performance on after-JIT duplicates than on before-JIT duplicates. Prior work already showed that REP outperforms BM25F in terms of Recall rate and Mean average precision. We show that the performance gap between BM25F and REP is even larger for after-JIT duplicates. Our results show that researchers who study the automated retrieval of duplicate reports should evaluate their approaches using after-JIT duplicates since such duplicates give a better representation of the expected performance of their approaches.

Paper organization. The rest of the paper is organized as follows. Section 2 presents background information and related work. Section 3 provides an overview of our experimental setup. Section 4 presents the results of our study. Section 5 discusses the implications of our study. Finally, we discuss the threats to the validity of our study in Section 6 and we present our conclusion in Section 7.

2 Background

In this section, we discuss popular ITSs, present the typical life cycle of an issue report, provide an overview of approaches for the automated retrieval of duplicate reports, overview the JIT duplicate retrieval feature and finally discuss the related work to our study.

2.1 Issue Tracking Systems (ITSs)

In the ITS of a software project, developers, testers and users can report issues that describe software bugs, new features or improvements. There exist many ITSs [16], of which Bugzilla and JIRA are the most famous (free) ones. Modern source code management platforms, such as GitHub, also offer issue tracking functionality. In this paper, we focus on open source issue tracking systems, in particular Bugzilla, as Bugzilla is used by a wide variety of large projects[1], such as the Linux kernel and Mozilla projects.

2.2 The Typical Life Cycle of an Issue Report

The typical life cycle of an issue report is illustrated in Figure 1. When an issue report is filed, it receives the "NEW" status. Each reported issue must be triaged in order to get processed. After triaging, the issue report's status is changed to "ASSIGNED" [25]. Each triaged issue report is investigated to determine whether it describes a new and valid software issue. Invalid issue reports are rejected and resolved as "INVALID" or "WONTFIX", while duplicate issue reports are resolved as a "DUPLICATE" of an existing issue. Finally, resolved issue reports are verified by the developers. During the verification, an issue report may be reopened by changing its status to "REOPENED". At the end, successfully verified issue reports receive the "VERIFIED" status.

2.3 Duplicate Issue Reports

Duplicate issue reports are reports that describe previously reported software issues. Duplicate issue reports are grouped together by resolving the new report as "DUPLICATE" and linking it to an existing report. In this section, we describe several concepts that are related to duplicate issue reports.

2.3.1 The Master Report

Similar to prior research [7, 8, 24, 28, 36, 40, 44], we group the related duplicate reports. The earliest issue report within a group of duplicate reports is labelled as the "master" report [13]. The goal of "duplicate issue report retrieval" is to identify the master report of a newly-reported duplicate.

2.3.2 Automated Approaches for Duplicate Retrieval

Due to the tediousness of manually retrieving duplicate issue reports, several automated approaches have been proposed to assist developers in identifying newly-reported duplicates [7, 23, 29, 40, 41, 45, 46]. These approaches leverage information retrieval (IR) [38] techniques to suggest candidate duplicate

[1] https://www.bugzilla.org/installation-list/

reports for newly-reported issues. In fact, almost all automated approaches for the retrieval of duplicate reports depend on a derived version of the same base technique, Term Frequency-Inverse Document Frequency (TF-IDF) [18]. The TF-IDF technique extracts the textual content of all previously reported issues into word frequency vectors to measure the importance of a term in a text document. TF-IDF represents the ability of each term to uniquely retrieve a document in the corpus of all documents. The basic formula of TF-IDF is:

$$\text{TF-IDF} = tf(t,d) * idf(t,D) \quad (1)$$

Where $tf(t,d)$ is the frequency of the term t in a document d, while $idf(t,D)$ is the total number of documents in the corpus divided by the number of documents that have the term t. The TF-IDF vector of a document is the vector containing the TF-IDF statistic of all terms in that document. Every document in the corpus has a TF-IDF vector, allowing documents to be compared using the distance between their vectors. A similarity score between the extracted vectors is calculated using similarity measures, such as the cosine similarity [18], Dice similarity [18], BLEU similarity [30] or Jaccard similarity [20]. For each newly-reported duplicate, an automated approach suggests a list of duplicate candidates that are sorted based on the similarity score. The reporter traverses the list and retrieves the accurate duplicate candidate (if any) for the newly-reported issue.

In this paper, we employ two popular similarity measures for short text (BM25F [34] and REP [40]) in our experiments. In the following paragraphs, we will give a brief overview of BM25F and REP.

2.3.3 BM25F

BM25F [34] is an advanced document similarity measure that is based on the TF-IDF vectors of documents [10, 34] (which are in our case, issue reports). The BM25F measure computes the similarity between a query (i.e., the newly-reported issue) and a document (i.e., one of the previously-reported issues) based on the common words that are shared between the two reports. To compute the similarity between the two reports, BM25F uses the TF-IDF vector of each of several fields of the issue reports (e.g., title, header and description). Different degrees of importance in the retrieval process can be given by assigning weights to the vector. For example, words appearing in the summary field of an issue report may be of a greater importance than words appearing in the description field of an issue report.

To find the best matching documents for a query, BM25F ranks the documents based on their TF-IDF statistics for words in the query. The BM25F approach is similar to the full-text search[2] function that is used in Bugzilla, which uses TF-IDF to retrieve duplicate issue reports based on the summary and description fields. BM25F employs several parameters that are automatically optimized using a set of already-known duplicate issue reports, to which we refer throughout this paper as the *tuning data* for BM25F.

[2] http://dev.mysql.com/doc/internals/en/full-text-search.html

2.3.4 REP

The REP similarity measure [40] calculates the similarities between the query and a document based on seven features (two for textual fields and five for categorical fields). A feature is a measurable value of similarity between a specific field of two issue reports, such as the similarity between the summary fields of the reports. REP extends the BM25F measure by also including categorical fields (e.g., the priority) when comparing issue reports. In addition, REP also considers the frequency of shared words in the new issue report and previously-reported issues.

The two textual field features (i.e., summary and description) are calculated based on an extended TF-IDF formula of BM25F. The five categorical field features (i.e., component, priority, product, type, and version) equal one if the field value in reports that are compared is exactly the same, and zero or $\frac{1}{1+(|v_1-v_2|)}$ (for priority and version) if they are not. For example, if the priority fields of two issue reports are 1 and 3, their feature is $\frac{1}{1+(|1-3|)} = \frac{1}{3}$. The REP approach includes a ranking function that combines the textual and categorical features as follows:

$$REP(d,q) = \sum_{i=1}^{7} w_i \times features_i \tag{2}$$

Where d and q are the issue reports that are being compared. The variable w_i is the weight for each feature. These weights are automatically optimized using a stochastic gradient descent algorithm [43], which uses a set of duplicate reports to find these weights. We refer throughout this paper to that set of duplicate reports as the *tuning data* for REP. The $features_i$ variable holds the feature value for each of the textual and categorical fields. For each newly-reported issue, the REP function is used to retrieve a list of possible master issue reports (*candidates*). The correct duplicate candidate (if any) is then selected from this list by the reporter.

While BM25F and REP themselves are not approaches for retrieving duplicate issue reports, several approaches [6, 7, 23, 29, 36, 40, 45, 46] rely heavily on either BM25F or REP. Therefore, we will refer to them in this paper as the BM25F or REP approaches.

2.4 JIT Duplicate Retrieval

Recent versions of ITSs offer a feature that displays a list of candidate duplicates at the time of filing a new issue. The goal of this feature is to prevent users from filing previously reported issues. For example, with the release of version 4.0, Bugzilla started using a full-text search[3] at filing time to assist in the retrieval of possible duplicate reports. Figure 2 shows an example of

[3] http://dev.mysql.com/doc/internals/en/full-text-search.html

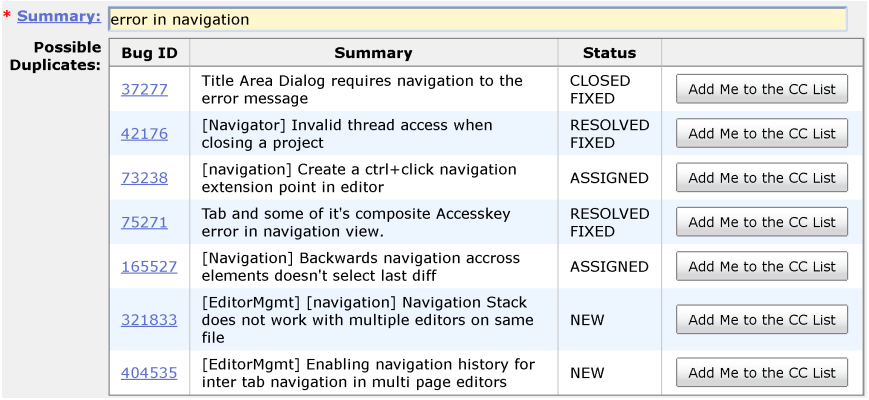


Fig. 2: An example of the JIT duplicate retrieval feature in Bugzilla.

Table 1: A list of popular ITSs and their support for JIT duplicate retrieval.

ITS	JIT duplicate retrieval feature
Bugzilla	Available from version 4.0 [1]
Jira	Available as plugin from version 6.0 [2]
Mantis [3]	-
RedMine [4]	-
Trac [5]	-

the JIT duplicate retrieval feature in Bugzilla. The Bugzilla feature uses the user-inputted text into the summary field of the issue report to find duplicate candidates[4]. Such a feature was frequently requested by Bugzilla users (see #22353[5] in the Bugzilla project). The JIT duplicate retrieval feature in Bugzilla depends only on the contents of the summary field. In this study, we refer to the duplicate reports that are filed after the activation of the JIT duplicate retrieval feature as *after-JIT duplicates*. The duplicate reports that are filed before the activation of the JIT duplicate retrieval feature are referred to as *before-JIT duplicates*. Table 1 shows a list of popular ITSs, together with their support for JIT duplicate retrieval. At the time of writing, only Bugzilla and Jira support JIT duplicate retrieval.

2.5 Related Work

There are several prior studies on duplicate issue reports and the automated retrieval of such reports. In this section, we survey work that is related to our study.

[4] https://github.com/bugzilla/bugzilla/blob/master/Bugzilla/Bug.pm#L599
[5] http://bugzilla.mozilla.org/show_bug.cgi?id=22353

2.5.1 Empirical Studies of Duplicate Issue Reports

Duplicate issue reports represent a large portion of issue reports. Anvik et al. [8] reported that 20-30% of the issue reports in Eclipse and Firefox respectively are duplicates. Cavalcanti et al. [17] found that duplicate reports represent 32%, 43%, and 8% of all the reports in Epiphany, Evolution and Tomcat, respectively. Bettenburg et al. [13] found that merging the information across duplicate reports produces additional useful information over using the information from a single report. In particular, they identify that before reports are identified as duplicates, different reporters with different ideas and suggestions may describe the same problem in different ways. Often, each of those ways contains partly unique information, which can contribute to getting a better description of the issue when merged.

2.5.2 Automated Retrieval of Duplicate Issue Reports

Many automated approaches have been proposed to retrieve duplicate issue reports. Runeson et al. [36] proposed a natural language processing approach to automatically rank duplicate candidates based on textual similarity. Issue reports are considered as text documents. The textual contents of the issue reports are preprocessed (i.e., using tokenization, stemming and stop word removal [32]) then a vector of term (word) frequencies (TF) is calculated for each issue report. Wang et al. [44] extended the work of Runeson et al. by not only considering TF, but also the inverse document frequency (IDF). In addition, Wang et al. considered the execution traces that are contained in the issue reports as one of the features to retrieve duplicate issue reports. Jalbert et al. [24] proposed a tool that computes the similarity between issue reports using textual features. Jalbert et al.'s approach used textual similarity combined with the non-textual features that are included in an issue report such as the product, component, version, and platform on which the reported issue occurs. Jalbert et al.'s approach leverages linear regression to predict the status of duplicate issue reports. Sureka et al. [42] use a character n-gram (see Section 3.3) approach to measure the text similarity between the titles and descriptions of issue reports. Sureka et al.'s approach is evaluated on 2,270 randomly selected reports from the Eclipse project. Sureka et al.'s approach does not use language-dependent preprocessing, such as stemming or stop word removal.

Sun et al. [41] built a discriminative model to determine if two issue reports are duplicates. The output of the model is a probability score which is used to determine the likelihood that an issue is a duplicate. Later, Sun et al. [40] extended the BM25F similarity measure [34] to support long queries (such as the full content of a duplicate issue report). In addition, Sun et al. proposed a ranking function, REP, that combines both the extended BM25F and categorical information for the retrieval of duplicate issue reports. Nguyen et al. [29] proposed an approach called DBTM to measure the similarity between issue reports sharing the same LDA topics [37]. Alipour et al. [7] and Aggarwal et

al. [6] proposed approaches that incorporate software contextual features. The creation of the contextual features includes manual labeling of software text books chapters. A word list is created from each chapter using the labeled Latent Dirichlet Allocation (*labeled-LDA*).

Banerjee et al. [9] use a random forest classifier along with 24 document similarity measures to retrieve duplicate reports. Borg et al. [15] studied if duplicate retrieval can be done by an off-the-shelf search engine, Apache Lucene. They found that Apache Lucene is a good candidate for retrieving duplicate issue reports. In addition, they found that assigning more weight to the title of an issue report can significantly improve duplicate retrieval. Borg and Runeson [14] showed how recommender systems can be used for retrieving duplicate issue reports.

Hindle [22] proposed a JIT duplicate retrieval feature for ITSs that is based on continuous querying (i.e., the feature asks many queries against a system while the user is typing, instead of a single query after entering the report title). He showed that the performance of continuous querying is not as high as that of a single query, but that continuous querying still can be useful for preventing duplicate issue reports from being submitted. Hindle's work is different from ours as we studied the real-world impact of the JIT duplicate retrieval feature over a long period of time, whereas Hindle evaluated his feature in experiments.

2.5.3 Revisiting the Performance Evaluation of Automated Approaches for the Retrieval of Duplicate Issue Reports

In our prior work [33], we conducted an empirical study on the needed effort for the retrieval of duplicate issue reports. We identified that 50% of the duplicate issue reports are straightforward to retrieve manually. Therefore, it is important that an automated approach for retrieving duplicate issue reports is capable of retrieving effort-consuming duplicates. We recommended that automated approaches for retrieving duplicate issue reports should consider the needed effort for the retrieval of duplicates in their evaluation.

In addition, we identified several flaws in the way in which approaches for retrieving duplicate issue reports are evaluated [32]. First, the evaluation is usually done on only a subset of issue reports. Therefore, duplicate issue reports with old master reports, which are more difficult to retrieve, are not included in the evaluation. Hence, the performance of the evaluated approach is often overestimated. Second, prior work usually reports a single performance value when evaluating an approach. Our work [32] showed that a range of performance values needs to be given in order to describe the performance of an approach accurately. We proposed a more realistic evaluation for automated approaches for retrieving duplicate issue reports.

2.5.4 Data Selection for Performance Evaluation

Prior work uses different issue reports from the ITSs for their performance evaluation. Table 2 shows an overview of prior research based on the data that

Study	Before-JIT Data	After-JIT Data
Runeson et al. [36]	✓	
Jalbert et al. [24]	✓	
Wang et al. [44]	✓	
Sun et al. [41]	✓	
Sureka et al. [42]	✓	
Sun et al. [40]	✓	
Nguyen et al. [29]	✓	
Cavalcanti et al. [17]	✓	
Rakha et al. [33]	✓	
Aggarwal et al. [6]	✓	
Banerjee et al. [9]*	✓	✓
Rakha et al. [32]	✓	

* Banerjee et al. [9] mixed Before-JIT and After-JIT in the same evaluation

Table 2: A summary of prior research based on their usage of before-JIT and after-JIT duplicate reports in their evaluation (ordered by publication date).

was used. We observe that almost all studies were applied on issues that were reported before the activation of the JIT duplicate retrieval feature. However, in this paper we show that future research needs to focus on after-JIT duplicates as these duplicates are more representative of the reports with which developers deal nowadays.

In contrast to prior work, we focus on highlighting the differences between before and after-JIT duplicates. In addition, we study how prior work is affected by these differences.

3 Experimental Setup

In this section, we present the projects that we studied and our process for collecting data for our experiments.

3.1 Studied Projects:

In this paper, we selected the studied projects based on the following criteria:

1. **Project uses the JIT duplicate retrieval feature:** the project uses an ITS that has the JIT duplicate retrieval feature activated for at least a year.
2. **Project has a considerable ratio of duplicates:** the project has a considerably large portion of duplicate issue reports (i.e., 20-30% of all the issue reports).

Table 3: The number of analyzed duplicate issue reports in each studied project.

Studied Project	Before-JIT issue reports		After-JIT issue reports	
	Total	Duplicates (%)	Total	Duplicates (%)
Mozilla-Firefox	93,941	28,647 (30%)	56,615	10,312 (18%)
Mozilla-Core	176,742	40,256 (23%)	119,998	11,931 (10%)
Eclipse-Platform	89,876	15,962 (18%)	11,536	1,399 (12%)

For the experiments in this paper, we selected issue reports from the three largest software projects from the ITSs of the Mozilla and Eclipse foundation (Mozilla-Firefox, Mozilla-Core and Eclipse-Platform). Both the Mozilla[6] and Eclipse[7] foundation started using Bugzilla 4.0, including the JIT duplicate retrieval feature, in 2011. We assume that Mozilla uses Bugzilla's latest stable version since Bugzilla is a Mozilla-sponsored project. The studied projects are known to have a considerable portion of duplicate reports [13, 33]. In addition, Mozilla and Eclipse projects' issue reports have been frequently used by prior research when studying automated approaches for the retrieval of duplicate reports [7, 8, 12, 33, 40].

We crawled the issue reports' XML files for the studied projects up to 31-Dec-2015. Then, we parsed the XML files for the further analysis that is presented in this paper. Table 3 shows the number of duplicate reports for each studied software project. We ignored issues that were reported in 2011 to leave a time buffer for the ITS's users to get familiar with the usage of the new JIT duplicate retrieval feature. The issue reports and our scripts are available in our online appendix [31].

3.2 Pre-processing of Issue Reports

In this paper, we executed the same text pre-processing implementation that was used by Sun et al. [40] and Nguyen et al. [29]:

- *Parsing tokens:* parsing the text sentences into words based on a selected delimiter, such as space or comma.
- *Stemming:* reducing the words to their root. For example, the words "developers" and "development" are reverted to the stem base "develop".
- *Removing stop words:* removing common words in the used language (i.e., English) that do not add value to the retrieval of duplicate candidates, such as "the", "are", and "is".

We applied the same pre-processing steps for all studied issue reports. For a more precise description of these steps, we refer to our replication package [31].

[6] https://www.bugzilla.org/news/

[7] https://bugs.eclipse.org/bugs/show_bug.cgi?id=359299

3.3 Studying the Characteristics of Duplicate Reports Before and After the Activation of the JIT Duplicate Retrieval Feature

We examine the characteristics of duplicates using the following metrics:

1. **Ratio of duplicates:** This ratio is the proportion of issue reports that are duplicates within a month [8, 13]. We divide the number of duplicate issues that are reported within one particular month by the total number of reported issues in that month. We study the ratio of duplicates before and after the activation of the JIT duplicate retrieval feature to examine the impact of the feature on the number of duplicates. We expect that the ratio of duplicates decreases after the activation of the JIT duplicate retrieval feature, because the goal of this feature is to reduce this ratio.
2. **Identification delay:** As suggested by our prior work [33], the identification delay is the time in days from triaging to resolving an issue report as a duplicate. We study this time as it is an indication of how long it takes developers to identify a duplicate. We expect that duplicate reports after the activation of the JIT retrieval feature need more time to be retrieved, because more time may be spent on less similar duplicates [33].
3. **Number of Comments:** As suggested by our prior work [33], the number of comments is the total number of comments on an issue report until the time of its resolution as a duplicate report. Auto-generated comments are filtered out from the number of comments (i.e, auto-generated comments for attachments or ****This issue has been marked as a duplicate of #****). We study the number of comments because it is a relatively reasonable proxy of the spent effort on deciding whether an issue report is a duplicate one. We expect that duplicate reports after the activation of the JIT retrieval feature would require more discussion, because after-JIT duplicates may be less textually similar to the master report, making it harder to identify them as a duplicate.
4. **Textual similarity:** Textual similarity plays a significant role in the automated retrieval of duplicate reports, since state of the art automated approaches for retrieval of duplicates, such as BM25F or REP, rely heavily on textual similarity. Generally, the most similar duplicates to their masters are easier to retrieve for current state of the art automated approaches. In addition, dissimilar duplicates need more effort to be manually identified [33]. As suggested by our prior work [33], we reuse the trigram textual similarity [26, 42] to highlight the differences in the duplicate reports' similarity to their master reports. The trigram algorithm works by dividing a piece of text into a vector of strings with three letters each (i.e., also called *trigrams*). For example, the string "new error" has the trigrams "new", "ew_", "w_e", "_er", "err", "rro", and "ror" where an underscore "_" marks a space. At the end, the two compared texts will have two sets of trigrams with no repetitions. The similarity score between the two sets is equal to the number of trigrams in common divided by the total number of trigrams. The similarity score ranges between 0 and 1 (i.e., identical texts have a score of 1). First, we removed all the special characters, dou-

ble spaces and converted all the text to lowercase. Then, we applied the trigram algorithm to calculate the similarity between the description fields of the duplicate issue reports and their masters. We expect that highly similar duplicates would be noted by the JIT duplicate retrieval feature and hence would not be filed and would not exist in the data.

3.4 Performance Evaluation Measures

In this study, we computed two frequently used performance measures to evaluate the studied automated approaches for the retrieval of duplicate issue reports [23, 29, 40, 41]: 1) Recall rate, and 2) Mean Average Precision (MAP). The Recall rate is defined as:

$$\text{Recall}_{\text{topN}} = \frac{retrieved_{topN}}{retrieved_{topN} + missed_{topN}}$$

where $retrieved_{topN}$ is the number of duplicate reports that successfully had their master reports retrieved in the *topN* ranked list of candidates, and $missed_{topN}$ is the number of duplicate reports that did not have their master reports retrieved in the *topN* candidates list. The larger the number of correctly retrieved master report candidates, the higher the Recall rate. The Recall rate value ranges from 0 to 1. In this paper, we study the top-5 and top-10 Recall rates.

The MAP measure indicates in which rank the correctly retrieved duplicate candidate is found in the returned list of candidates. The MAP is measured as follows:

$$\text{MAP} = \frac{1}{Q} \sum_{n=1}^{Q} \frac{1}{rank(n)}$$

where Q is the total number of accurately-found duplicate candidates, and *rank* is the position of the master report in the list. The MAP value ranges from 0 to 1. A MAP value of 1 means that the accurate candidates appear on the first rank of the returned list all the time. To reduce the computational complexity of the experiments in this study, we limited the MAP calculation to a list size of 1,000 candidates.

In this paper, we used the BM25F and REP implementations that were provided by Sun et al[8].

4 Experimental results

In this section, we present the results of our experiments. For each research question, we discuss the motivation, approach and results.

[8] http://www.comp.nus.edu.sg/~specmine/suncn/ase11/index.html

RQ1: How do the characteristics of duplicates differ before and after the activation of the just-in-time duplicate retrieval feature?

Motivation. The manual retrieval of duplicate issue reports is a tedious task [8, 11, 12, 33]. Therefore, many prior studies have proposed automated approaches to assist in the retrieval of duplicate reports [6, 7, 23, 24, 29, 36, 40, 44]. In addition, recent versions of ITSs provide a JIT duplicate retrieval feature (see Section 2.4), which shows a list of candidate duplicates for each report at filing time. In the evaluation of the proposed approaches, prior work never considered the possible impact of the JIT duplicate retrieval feature on the duplicate reports that end up in the ITS. In this paper, we study this impact. First, we compare the characteristics of before and after-JIT duplicates in this RQ. Examining the differences between before and after-JIT duplicates can lead to insights about whether future studies should take into account the impact of the JIT duplicate retrieval feature when collecting data for the performance evaluation of the automated approaches.

Approach. In order to study the impact of the JIT duplicate retrieval feature, we compared the characteristics of the before-JIT and after-JIT duplicates. We divided the issue reports into two groups. The first group includes all the issue reports prior to 2011 (i.e., the before-JIT duplicates), while the second group includes all the issue reports after 2011 (i.e., the after-JIT duplicates). As mentioned in Section 3.1, we do not use the issue reports that were filed in 2011 to leave a time buffer for the ITS's users to get familiar with the JIT duplicate retrieval feature. For all the studied projects, we have four years of issue reports that were reported after the activation of the JIT duplicate retrieval feature. We explored the differences in characteristics between the duplicate reports before and after the activation of the JIT duplicate retrieval feature through the characteristic metrics (Ratio of duplicates, Identification delay, Number of comments and Textual similarity) that were presented in Section 3.3.

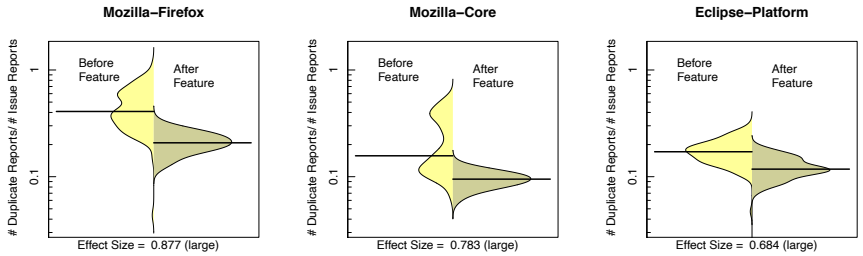
For each metric, we compared the two groups using the *Mann-Whitney U* statistical test [19]. We use the following hypotheses:

H_0= The two groups of duplicate reports have similar metric values.

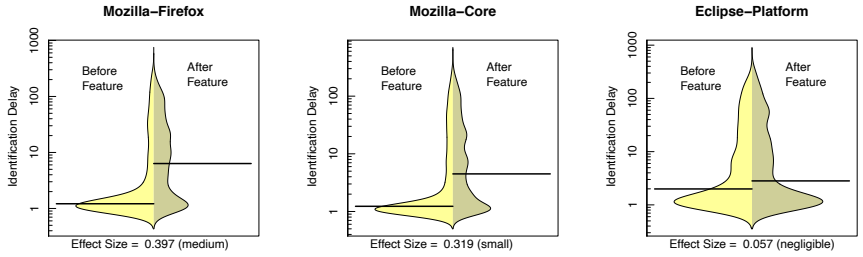
H_1= The two groups of duplicate reports have different metric values.

We reject H_0 and accept H_1, when $p < 0.01$. In addition, we calculated the *effect size*, as the effect size quantifies the difference between the two group distributions. We used *Cliff's Delta* as it does not require the distributions normality assumption to quantify the effect size [27]. The following thresholds are used for *Cliff's Delta (d)* [35]:

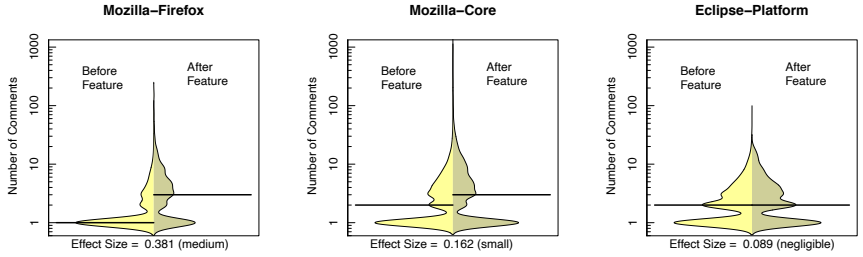
$$\begin{cases} \text{negligible} & \text{for } |d| \leq 0.147 \\ \text{small} & \text{for } 0.147 < |d| \leq 0.33 \\ \text{medium} & \text{for } 0.33 < |d| \leq 0.474 \\ \text{large} & \text{for } 0.474 < |d| \leq 1 \end{cases} \quad (3)$$



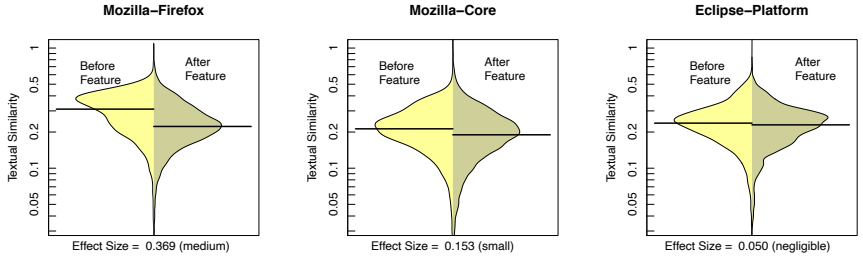
(a) Ratio of duplicates



(b) Identification Delay



(c) Number of Comments



(d) Textual Similarity

Fig. 3: A comparison of the characteristics of duplicate reports before and after the activation of the JIT duplicate retrieval feature for the studied projects. Note that all the axes are in a logarithmic scale.

Results. **Significantly less duplicate reports end up in an ITS after the activation of the JIT duplicate retrieval feature.** Figure 3a shows the distributions of the ratio of before-JIT and after-JIT duplicates. For all the studied projects, the ratio of the after-JIT duplicates is significantly lower than the ratio of the before-JIT duplicates. We observe that the effect size for all projects is *large*. This result indicates that there is a correlation between the activated JIT duplicate retrieval feature and the ratio of duplicates. However, the ratio of the after-JIT duplicates to the overall issues can still reach large values (e.g., 43% for Mozilla-Firefox and 22% for Eclipse-Platform). Therefore, activating a more advanced automated approach as the JIT duplicate retrieval feature is still a worthwhile goal for next generation ITSs.

After-JIT duplicates have a significantly larger identification delay than before-JIT duplicates and need more comments to be manually identified. Figures 3b and 3c show the distributions of the identification delay and number of comments for the studied projects. We observe a significant increase in the needed effort for identifying duplicate reports with small (Mozilla-Core) and medium (Mozilla-Firefox) effect sizes. However, the Eclipse-Platform has a significant difference with a negligible effect size. From our manual investigation, one possible explanation for our finding is that Eclipse-Bugzilla allows the filing of issue reports through a secondary form that uses the default form rather than the JIT duplicate retrieval feature[9].

After-JIT duplicates share a significantly smaller textual similarity with their master reports. Figure 3d shows the distributions of the textual similarity of the duplicate reports of the studied projects. We observe that after-JIT duplicates share a smaller textual similarity with their masters than before-JIT duplicates (with medium and small effect size).

> The JIT duplicate retrieval feature removes the trivial textually-similar duplicates but other duplicates which are hard to retrieve remain. Given that automated approaches rely heavily on textual similarity, we believe that future ITSs need to employ additional techniques to retrieve duplicates. In the following RQ, we delve deeper into our observations about current automated approaches for the retrieval of duplicates.

RQ2: How does the just-in-time duplicate retrieval feature impact the performance evaluation of state of the art automated approaches for the retrieval of duplicate issue reports?

Motivation. As shown in RQ1, the characteristics of after-JIT reports differ significantly from the characteristics of before-JIT duplicates. In this RQ, we

[9] Issue#393235: https://bugs.eclipse.org/bugs/show_bug.cgi?id=393235. We manually verified that this issue still persists.

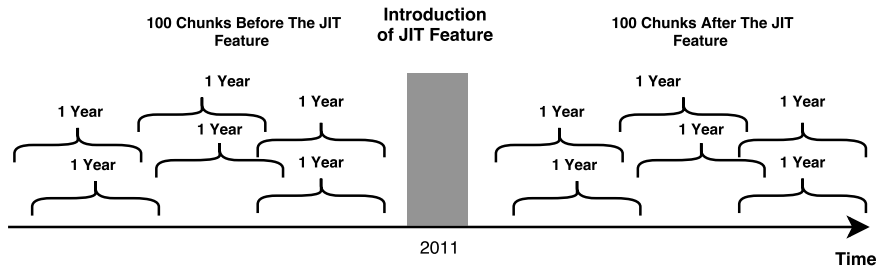
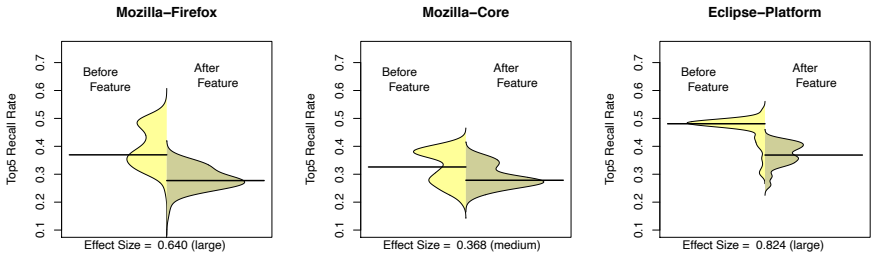


Fig. 4: The selection of data chunks before and after the activation of the JIT duplicate retrieval feature.

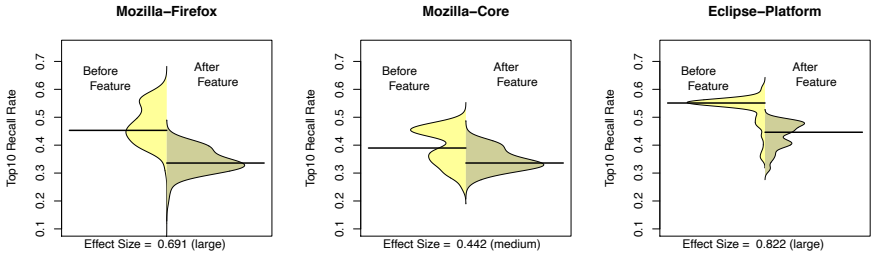
study whether these different characteristics impact the performance of existing automated approaches for the retrieval of duplicate issue reports. If the performance is affected, knowing so is important for the evaluation of future approaches, and would indicate that previously reported results might not hold for feature deployments.

Approach. In this RQ, we compare the performance of the BM25F and REP approaches when evaluated on before-JIT and after-JIT duplicates. First, we randomly picked 100 chunks of before-JIT duplicates and 100 chunks of after-JIT duplicates from the studied ITSs (similar to our prior work [32]). Each chunk of data is of one year-length (see Figure 4) and consists of all issue reports that are reported in one year. For each studied project, we can select from a large number of chunks of data with a one-year length (e.g., January 2010 to December 2010 and February 2010 to January 2011 are considered different chunks). For each studied chunk, we used the first 200 duplicates of that chunk to tune the approach [6, 23, 26, 29, 40], then we calculated the performance of the BM25F and REP approaches on the remaining duplicates in that chunk. We used the same tuning algorithm that was used by prior research [29, 40]. In our prior work [32], we showed that the choice of tuning data does not matter much, as long as the tuning process is executed. After tuning the approaches, we compared the performance before and after the activation of the JIT duplicate retrieval feature from two perspectives:

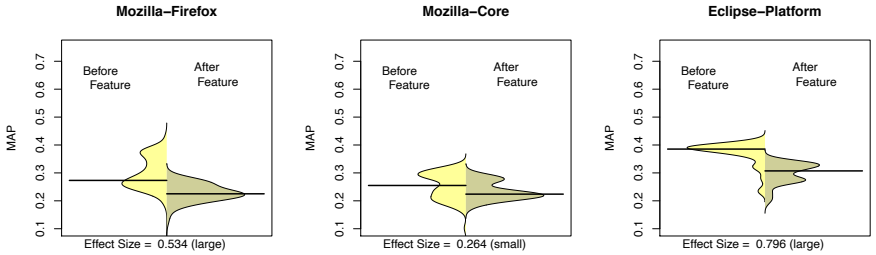
1. **Performance per approach:** for both BM25F and REP, we compared the distributions of the performance measure values. Each distribution of a performance measure has 100 values before and after the activation of the JIT duplicate retrieval feature. This perspective gives researchers an idea of what they should expect when they use after-JIT duplicates to evaluate their proposed approaches.
2. **Relative Improvement Gap:** the performance comparison of a newly-proposed automated approach with an older one is an important part of research in this domain [6, 23, 26, 29, 40, 41, 45, 46]. Prior research has



(a) Top-5 Recall rate



(b) Top-10 Recall rate

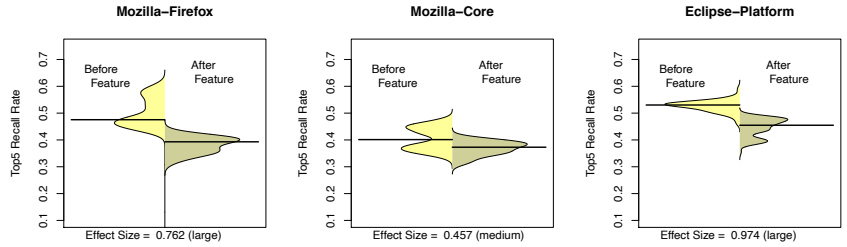


(c) MAP

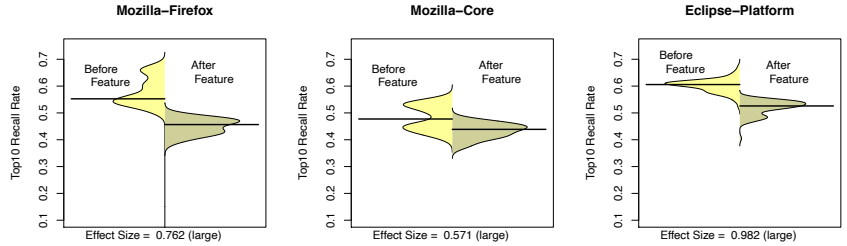
Fig. 5: A comparison of the performance of BM25F before and after the activation of the JIT duplicate retrieval feature for the studied projects.

already shown that REP outperforms BM25F [29, 40, 45]. However, this result followed from an evaluation using before-JIT duplicates. We study whether such prior result still holds for after-JIT duplicates. Similar to prior research [6, 23, 26, 29, 40, 41, 45, 46], we used the relative improvement of the performance to show the difference between the BM25F and REP approaches for each data chunk. The relative improvement is defined as follows:

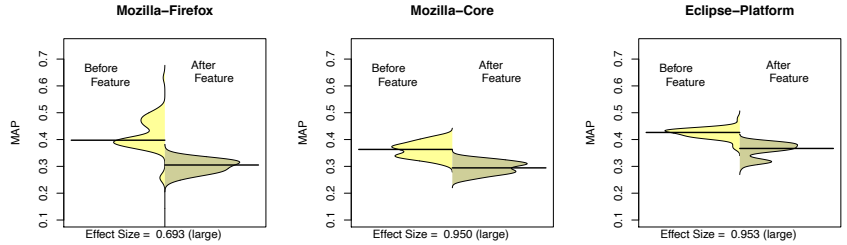
$$Relative\ Improvement = \left(\frac{measure_{REP} - measure_{BM25F}}{measure_{BM25F}}\right) \quad (4)$$



(a) Top-5 Recall rate



(b) Top-10 Recall rate

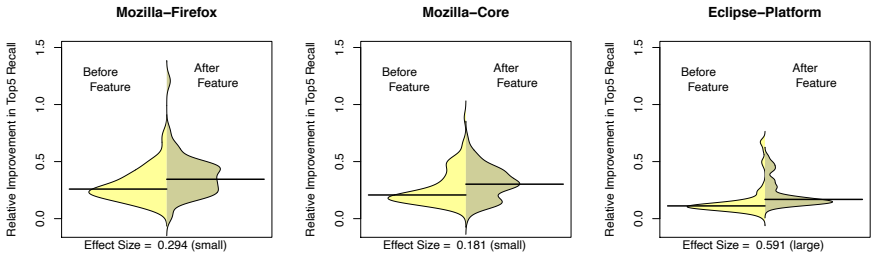


(c) MAP

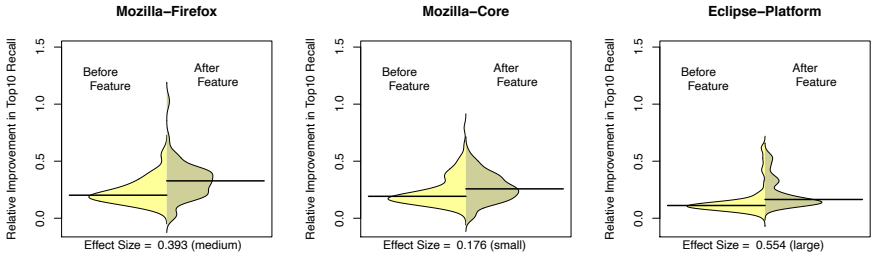
Fig. 6: A comparison of the performance of REP before and after the activation of the JIT duplicate retrieval feature for the studied projects.

where $measure_{REP}$ is the performance of the REP approach for a certain data chunk, while $measure_{BM25F}$ is the performance of the BM25F approach for the same data chunk. The relative improvement is calculated for all the measures that are covered in our study (i.e., $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP). A relative improvement of zero means that both approaches have identical performance. A relative improvement that is larger than zero means that the REP approach is outperforming the BM25F approach and vice versa for a relative improvement that is smaller than zero.

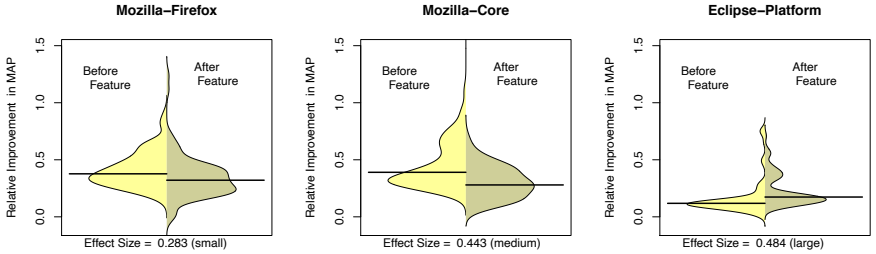
As in RQ1, we compared the distributions of performance measures using the Mann-Whitney U test and Cliff's Delta effect size.



(a) Relative improvement in the top-5 Recall rate



(b) Relative improvement in the top-10 Recall rate



(c) Relative improvement in the MAP

Fig. 7: Comparison of the relative improvement in performance from the BM25F approach to the REP approach before and after the activation of the JIT duplicate retrieval feature (the y-axis represents the relative percentage of the improvement for each performance measure).

Results. **The performance of both BM25F and REP is lower for after-JIT duplicates.** Figures 5 and 6 show the distributions of performance measures (i.e., $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP) before and after the activation of the JIT duplicate retrieval feature for the BM25F and REP approaches, respectively. For all studied projects, the performance after the activation of the JIT duplicate retrieval feature is significantly lower than before. In most cases, the difference has a large effect size. These results indicate that future studies of the automated retrieval of duplicate reports should use after-JIT

duplicates in their evaluations as these duplicates are the most representative duplicates in practice nowadays. Therefore, after-JIT duplicates will give the most accurate and realistic view of the performance of an automated approach in practice.

The relative improvement of the performance of REP over BM25F is significantly larger after the activation of the JIT duplicate retrieval feature. Figure 7 shows the distributions of the relative improvement over the BM25F approach by the REP approach before and after the activation of the JIT duplicate retrieval feature. The only exception is for the MAP for Mozilla-Firefox and Mozilla-Core. The reason for that exception is that the difference in MAP for before and after-JIT duplicates for BM25F is smaller than for REP. Generally, the results show that the relative improvement significantly increases after the activation of the JIT duplicate retrieval feature. The reason for the bigger drop in performance of the BM25F approach in contrast to the REP approach is explained by the differences between the ranking function of each approach. In contrast to the ranking function of BM25F, which relies solely on textual similarity, the ranking function of REP depends on categorical fields as well. Hence, the performance of the REP approach is less susceptible to the lower textual similarity after the activation of the JIT duplicate retrieval feature.

Our results show that duplicate retrieval based on basic textual similarity no longer works, as the JIT duplicate retrieval feature takes care of such duplicates. Instead, more sophisticated approaches are necessary. REP is a first step towards such an approach, as it does not exclusively rely on textual similarity.

5 Implications

In this section, we discuss the implications of our work for researchers and ITS developers that are interested in the automated retrieval of duplicate reports.

The JIT duplicate retrieval feature appears to lower the number of duplicates that end up in the ITS. Our results show that the portion of duplicates in the ITS is significantly lower after the activation of the JIT duplicate retrieval feature. Future studies should investigate how this portion can be made even lower.

To improve the JIT duplicate retrieval feature, researchers need to consider the trade-off between execution complexity and performance. The JIT duplicate retrieval feature involves the filer of an issue report in finding its potential duplicate. The advantages are that: 1) the actual filer knows best what the report describes and 2) the immediate feedback of the feature ensures the freshness of the report in the filer's mind. However, the activation of an automated approach at filing time is restricted by execution complexity. For example, the currently activated feature in Bugzilla relies only on a single textual field (i.e., the summary field) which makes it much

simpler and therefore faster than the approaches that were proposed by prior research [6, 7, 23, 24, 29, 36, 40, 44], including BM25F and REP. The execution complexity is important, as the list of duplicate candidates should be returned to the reporter directly after typing a new word in the summary field. Therefore, future studies are necessary to investigate how the performance of the JIT duplicate retrieval feature can be improved while keeping the computational complexity low. A possible solution is to implement a more sophisticated retrieval approach which runs periodically (e.g., nightly) on recently-filed issue reports and notifies the filer by email about possible duplicates. Such an approach would lower the computational burden of needing to run for every filed report, while it could still deliver relatively fast feedback (i.e., within minutes).

Future studies should evaluate automated approaches for duplicate retrieval using after-JIT duplicates. In RQ1, we showed how after-JIT duplicates differ from before-JIT duplicates. In addition, we illustrated the impact of the new characteristics of such duplicates on the performance of automated approaches in RQ2. These findings give an insight on what can be expected when evaluating on after-JIT duplicates. Generally, after-JIT duplicates are more representative of the issue reports that exist nowadays for the studied projects. Therefore, future studies should perform their evaluations on after-JIT duplicates.

6 Threats to Validity

In this section, we discuss the threats to the validity of our study.

6.1 External Validity

These threats concern the ability of our study to generalize our findings to other software projects. In this study, we investigated duplicate issue reports of three large open-source software projects. However, similar findings may not hold for software projects from other domains such as commercial projects. In order to mitigate this threat, more software projects, preferably commercial software projects need to be analyzed in future studies. For example, Micro Focus' commercial solution ALM for issue tracking requires a manual action to check for duplicate issue reports when reporting a new issue[10]. Future studies should investigate the impact of requiring a manual action to trigger the JIT feature.

In addition, future studies are necessary to investigate the impact of the JIT duplicate retrieval feature on other types of issue reports, such as issue reports for compilers [39].

We only studied the impact of the activation of the JIT duplicate retrieval feature in one ITS system (i.e., Bugzilla) which is a threat to the generality

[10] https://alm-help.saas.hpe.com/en/12.55/online_help/Content/UG/ui_similar_defects.htm

of our study. However, the large majority of the studies on the automated retrieval of duplicate reports focus on Bugzilla [6, 7, 23, 29, 36, 40, 45, 46]. Therefore, our findings should apply to the majority of prior work.

The majority of approaches for retrieving duplicate issue reports are based on a version of the same base technique (TF-IDF). In this paper, we focused on REP and BM25F as these are by far the most used TF-IDF-based approaches for retrieving duplicate issue reports. REP [40] and BM25F [34] have always been treated as two separate approaches in prior research [29, 40, 46]. The REP approach depends on a ranking function that combines the BM25Fext approach (which is an extended version of BM25F by Sun et al. [40]) along with categorical fields of issue reports. Nowadays, even the most recent approaches make only small increments to the REP or BM25F approach [6, 23, 29]. Therefore, studying the REP and BM25F approach covers the majority of the spectrum of the approaches for retrieving duplicate issue reports.

We did not study how the JIT duplicate retrieval feature affects the evaluations of other approaches for duplicate issue retrieval, such as topic modeling-based approaches. We expect that the evaluations of these approaches are impacted by the JIT feature as well. The challenge of retrieving the after-JIT duplicates is that they are not as textually similar as before-JIT duplicates. Therefore, the extracted topics would also be different. Similar reasonings can be given for other approaches for the retrieval of duplicate issue reports. Future studies are necessary to confirm our expectation.

6.2 Internal Validity

In this study, we assumed that the only impactful change in ITSs in 2011 was the activation of the JIT duplicate retrieval feature. In addition, we assumed that the JIT duplicate retrieval feature was not deactivated after its activation in 2011. Future studies need to investigate the possible impact of other new features in that year (or following years) on the automated retrieval of duplicate issue reports.

We can select from a large number of chunks of data with a one-year length (e.g., January, 2010 to December, 2010 and February, 2011 to January, 2012) for the experiments that are applied on each studied ITS. However, running the approaches for retrieving duplicate issue reports is costly in terms of time. Therefore, we limited the number of chunks that we evaluated to 100 randomly-selected chunks of before-JIT and after-JIT duplicates. Evaluating 200 chunks of one-year length in total should give a strict enough confidence interval.

7 Conclusion

In this paper, we explored the impact of the activation of the JIT duplicate retrieval feature on the automated retrieval of duplicate reports. First, we studied the characteristics of the duplicate issue reports of three software projects

(Mozilla-Firefox, Mozilla-Core and Eclipse-Platform) before and after the activation of the JIT duplicate retrieval. In particular, we explored four metrics of characteristics covering the ratio of duplicates, identification delay, number of comments, and textual similarity. Second, we applied two automated approaches for the retrieval of duplicate reports (BM25F and REP) before and after the activation of the JIT duplicate retrieval feature. The highlights of our study are:

1. The portion of duplicate issue reports in an ITS is significantly lower after the activation of the JIT duplicate retrieval feature.

2. The after-JIT duplicates have significantly less textual similarity and need more effort to be manually retrieved than before-JIT duplicates.

3. The changed characteristics of after-JIT duplicates can have a significant impact on the performance of automated approaches for the retrieval of duplicate issue reports. In particular, we showed that both BM25F and REP perform significantly lower on after-JIT duplicates.

Our findings highlight that future studies of the automated retrieved of duplicate issue reports have to focus on after-JIT duplicates, as these duplicates are more representative of new issue reports. In addition, our findings indicate that automated retrieval of duplicates based on basic textual similarity is no longer effective, as many textually similar duplicates are already retrieved by the JIT duplicate retrieval feature. Instead, more sophisticated approaches that do not solely rely on textual similarity are necessary. In addition, future studies should investigate how to further improve the JIT duplicate retrieval feature.

8 Acknowledgments

This study would not have been possible without the High Performance Computing (HPC) systems that are shared by Compute Canada[11] and the Centre for Advanced Computing[12] as well as the tools provided by Sun et al. [40].

References

1. Bugzilla Release notes for Bugzilla 4.0. https://www.bugzilla.org/releases/4.0/release-notes.html. Last visited on 11/12/2017
2. Jira Duplicate Detection. https://marketplace.atlassian.com/plugins/com.deniz.jira.similarissues/server/overview. Last visited on 11/12/2017

[11] https://www.computecanada.ca/
[12] http://cac.queensu.ca/

3. Mantis Bug Tracker. https://www.mantisbt.org/. Last visited on 11/12/2017
4. RedMine Flexible Project Management. https://www.redmine.org/. Last visited on 11/12/2017
5. The Trac Project. https://trac.edgewall.org/. Last visited on 11/12/2017
6. Aggarwal, K., Rutgers, T., Timbers, F., Hindle, A., Greiner, R., Stroulia, E.: Detecting duplicate bug reports with software engineering domain knowledge. In: Proceedings of the 22th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 211–220. IEEE (2015)
7. Alipour, A., Hindle, A., Stroulia, E.: A contextual approach towards more accurate duplicate bug report detection. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), pp. 183–192 (2013)
8. Anvik, J., Hiew, L., Murphy, G.C.: Coping with an open bug repository. In: Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (Eclipse), pp. 35–39. ACM (2005)
9. Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., Cukic, B.: Automated triaging of very large bug repositories. Information and Software Technology **89**(Supplement C), 1–13 (2017)
10. Berry, M.W., Castellanos, M.: Survey of text mining. Computing Reviews **45**(9), 548 (2004)
11. Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., Zimmermann, T.: Quality of bug reports in eclipse. In: Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (Eclipse), pp. 21–25. ACM (2007)
12. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT/FSE), pp. 308–318. ACM (2008)
13. Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.: Duplicate bug reports considered harmful...really? In: Proceedings of the 24th International Conference on Software Maintenance (ICSM), pp. 337–345. IEEE (2008)
14. Borg, M., Runeson, P.: Changes, Evolution, and Bugs, pp. 477–509. Springer Berlin Heidelberg (2014)
15. Borg, M., Runeson, P., Johansson, J., Mäntylä, M.V.: A replicated study on duplicate detection: Using apache lucene to search among android defects. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 8:1–8:4. ACM, New York, NY, USA (2014)
16. Cavalcanti, Y.C., da Mota Silveira Neto, P.A., Machado, I.d.C., Vale, T.F., de Almeida, E.S., Meira, S.R.d.L.: Challenges and opportunities for software change request repositories: a systematic mapping study. Journal of Software: Evolution and Process **26**(7), 620–653 (2014)

17. Cavalcanti, Y.C., Neto, P.A.d.M.S., Lucrédio, D., Vale, T., de Almeida, E.S., de Lemos Meira, S.R.: The bug report duplication problem: an exploratory study. Software Quality Journal **21**(1), 39–66 (2013)
18. Chowdhury, G.: Introduction to modern information retrieval. Facet publishing (2010)
19. Gehan, E.A.: A generalized Wilcoxon test for comparing arbitrarily singly-censored samples. Biometrika **52**(1-2), 203–223 (1965)
20. Hamers, L., Hemeryck, Y., Herweyers, G., Janssen, M., Keters, H., Rousseau, R., Vanhoutte, A.: Similarity measures in scientometric research: The Jaccard index versus Salton's cosine formula. Information Processing and Management **25**(3), 315–318 (1989)
21. Hassan, A.E.: The road ahead for mining software repositories. In: Proceedings of the Frontiers of Software Maintenance (FoSM), pp. 48–57. IEEE (2008)
22. Hindle, A.: Stopping duplicate bug reports before they start with Continuous Querying for bug reports. PeerJ Preprints **4**, e2373v1 (2016)
23. Hindle, A., Alipour, A., Stroulia, E.: A contextual approach towards more accurate duplicate bug report detection and ranking. Empirical Software Engineering **21**(2), 368–410 (2016)
24. Jalbert, N., Weimer, W.: Automated duplicate detection for bug tracking systems. In: Proceedings of the 38th International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), pp. 52–61. IEEE (2008)
25. Koponen, T.: Life cycle of defects in open source software projects. In: Open Source Systems, pp. 195–200. Springer (2006)
26. Lazar, A., Ritchey, S., Sharif, B.: Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), pp. 308–311. ACM (2014)
27. Long, J.D., Feng, D., Cliff, N.: Ordinal analysis of behavioral data. Handbook of psychology (2003)
28. Nagwani, N.K., Singh, P.: Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs. In: Proceedings of the 1st International Conference on Advances in Computing, Communication and Control (ICAC3), pp. 202–207. ACM (2009)
29. Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., Lo, D., Sun, C.: Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 70–79. ACM (2012)
30. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pp. 311–318. Association for Computational Linguistics (2002)
31. Rakha, M.S., Bezemer, C.P., Hassan, A.E.: Revisiting the Performance of Automated Approaches for the Retrieval of Duplicate Reports in Is-

sue Tracking Systems that Perform Just-in-Time Duplicate Retrieval: Online Appendix. https://github.com/SAILResearch/replication-jit_duplicates. Last visited on 11/12/2017
32. Rakha, M.S., Bezemer, C.P., Hassan, A.E.: Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Transactions on Software Engineering (TSE) **PP**(99), 1–27 (2017)
33. Rakha, M.S., Shang, W., Hassan, A.E.: Studying the needed effort for identifying duplicate issues. Empirical Software Engineering (EMSE) **21**(5), 1960–1989 (2016)
34. Robertson, S., Zaragoza, H., Taylor, M.: Simple BM25 extension to multiple weighted fields. In: Proceedings of the 13th International Conference on Information and Knowledge Management (CIKM), pp. 42–49. ACM (2004)
35. Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J., Devine, L.: Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and Cohens'd indices the most appropriate choices. In: Annual Meeting of the Southern Association for Institutional Research (2006)
36. Runeson, P., Alexandersson, M., Nyholm, O.: Detection of duplicate defect reports using natural language processing. In: Proceedings of the 29th International Conference on Software Engineering (ICSE), pp. 499–510. IEEE Computer Society (2007)
37. Somasundaram, K., Murphy, G.C.: Automatic categorization of bug reports using Latent Dirichlet Allocation. In: Proceedings of the 5th India Software Engineering Conference (ISEC), pp. 125–130. ACM (2012)
38. Strzalkowski, T., Lin, F., Wang, J., Perez-Carballo, J.: Evaluating natural language processing techniques in information retrieval. In: Natural language information retrieval, pp. 113–145. Springer (1999)
39. Sun, C., Le, V., Zhang, Q., Su, Z.: Toward understanding compiler bugs in GCC and LLVM. In: Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA), pp. 294–305. ACM, New York, NY, USA (2016)
40. Sun, C., Lo, D., Khoo, S.C., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 253–262. IEEE (2011)
41. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of the 32th ACM/IEEE International Conference on Software Engineering (ICSE), pp. 45–54. ACM (2010)
42. Sureka, A., Jalote, P.: Detecting duplicate bug report using character n-gram-based features. In: Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC), pp. 366–374. IEEE Computer Society (2010)

43. Taylor, M., Zaragoza, H., Craswell, N., Robertson, S., Burges, C.: Optimisation methods for ranking functions with multiple parameters. In: CIKM 2006: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, pp. 585–593. ACM (2006)
44. Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering (ICSE), pp. 461–470. ACM (2008)
45. Zhou, J., Zhang, H.: Learning to rank duplicate bug reports. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM), pp. 852–861. ACM (2012)
46. Zou, J., Xu, L., Yang, M., Zhang, X., Zeng, J., Hirokawa, S.: Automated duplicate bug report detection using multi-factor analysis. IEICE Transactions on Information and Systems **E99.D**(7), 1762–1775 (2016)