REVISITING THE EXPERIMENTAL DESIGN CHOICES FOR APPROACHES FOR THE AUTOMATED RETRIEVAL OF DUPLICATE ISSUE REPORTS

by

MOHAMED SAMI RAKHA

A thesis submitted to the

Graduate Program in Computing

in conformity with the requirements for

the degree of Doctor of Philosophy

Queen's University

Kingston, Ontario, Canada

November 2017

Abstract

Issue tracking systems, such as Bugzilla, are commonly used to track reported bugs and change requests. Duplicate reports have been considered as a hindrance to developers and a drain on their resources. To avoid wasting developer resources on previously-reported (i.e., duplicate) issues, it is necessary to identify such duplicates as soon as they are reported. In recent years, several approaches have been proposed for the automated retrieval of duplicate reports. These approaches leverage the textual, categorical, and contextual information in previously reported issues to determine whether a newly-reported issue has been previously-reported. In general, studies that are designed to evaluate these approaches treat all the duplicate issue reports equally, make use of data chunks that span a relatively short period of time, and ignore the impact of newly-activated features (e.g., just-in-time lightweight retrieval of duplicates at filing time) in the recent issue tracking systems.

This thesis revisits the experimental design choices of such prior studies along three

perspectives: 1) Used performance measures, 2) Evaluation process, and 3) Experiment's data choice. For the performance measures, we highlight the need for effort-aware evaluation of such approaches, since the identification of a considerable amount of duplicate reports (over 50%) appears to be a relatively trivial task.

For the evaluation process, we show that the previously-reported performance of such approaches is significantly overestimated.

Finally, recent versions of ITSs perform just-in-time lightweight retrieval of duplicate issue reports at the filing time of an issue report. The aim of such just-in-time retrieval is to avoid the filing of duplicates. We show that future studies of the automated retrieval of duplicate reports have to focus on after-JIT duplicates, as these duplicates are more representative of issue reports in practice nowadays.

Our results through this thesis highlight the current state of progress in the automated retrieval of duplicate reports while charting directions for future research efforts.

Acknowledgments

I wish to express my gratitude to Allah (God) for providing me the blessings to complete this work. I also grateful to my supervisor Professor Ahmed E. Hassan who offered me the opportunity to continue my graduate studies at Queen's University and guided me through my research work. I acknowledge his patience and his continuous support during my study.

I am very honored to have the chance to work and collaborate with the brightest researchers during my Ph.D. career. I would like to thank all of my collaborators, Dr. Cor-Paul Bezemer, Dr. Weiyi Shang, Dr. Shane McIntosh, and Dr. Nasir Ali.

My appreciation extends to my fellow labmates: Dr. Daniel Da Costa, Safawt Hassan, Suhas Kabinna, Dr. Mark D. Syer, Hammam AlGhamdi, Heng Li, Baljinder Ghotra, Dr. Gustavo Ansaldi Oliva, Ravjot Singh, Dayi Lin, Dr. Yasutaka Kamei, Dr. Chakkrit Tantithamthavorn, and Dr. Patanamon Thongtanunam. Also, I thank my friends: Dr. Shadi Khalifa, Dr. Yehia Elshater, Dr. Fahim Imam, Prashant Agrawel, Dima Liashenko,

Hassan Nouri, Ahmed Youssef, Ranjan Rahul and Mahmoud Ragab for all the joyful memories we had in the last four years.

I also thank Compute Canada and the Centre for Advanced Computing at Queen's University for providing me access to the High Performance Computing (HPC) systems. Such systems were essential to perform the work that is presented in this thesis.

A special thanks to my family, Sami Rakha, Olfat Hassan, Ahmed Sami and Tamer Sami. Thank you for all the sacrifices that you made for me during my study. Your precious and invaluable love have been my greatest support in my life.

Co-authorship

Earlier versions of the work in the thesis were published as listed below:

1. Studying the Needed Effort for Identifying Duplicate Issues.
   Mohamed Sami Rakha, Weiyi Shang and Ahmed E. Hassan. Empirical Software Engineering Journal (EMSE), 2015.

   My contribution: Drafting the research plan, collecting the data, analyzing the data, writing and polishing the paper drafts.

2. Revisiting the Performance Evaluation of Automated Approaches for the Retrieval of Duplicate Issue Reports
   Mohamed Sami Rakha, Cor-Paul Bezemer and Ahmed E. Hassan. IEEE Transactions on Software Engineering (TSE), 2017.

   My contribution: Drafting the research plan, collecting the data, analyzing the data, writing and polishing the paper drafts.

3. 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 and Ahmed E. Hassan. Empirical Software Engineering Journal (EMSE)[Under review], 2017.

   My contribution: Drafting the research plan, collecting the data, analyzing the data, writing and polishing the paper drafts.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Issue tracking systems (ITSs) are widely used in practice to track a wide array of reported customer requests and concerns (Hassan, 2008). Examples of ITSs include BugZilla[1],Fogbugz[2], JIRA[3], and Trac[4]. An ITS is an essential project component that allows users and project members to report and track software issues during the development of a large software project (Bertram et al., 2010). For example, users can receive email updates when issue reports are resolved or assigned. Furthermore, users can discuss issues directly with team members. Mining the data in the repositories of such systems often leads to significant insights and improved decisions (Hassan, 2006).

[1] https://bugzilla.mozilla.org/
[2] http://www.fogcreek.com/Fogbugz/
[3] https://issues.apache.org/jira/
[4] http://trac.edgewall.org

Newly submitted issue reports to an ITS usually go through a filtration process before they are assigned to developers. One of the common filtering steps is to determine whether a new-reported issue is a duplicate of an existing issue. Two issue reports are considered duplicates of each other, if they refer to a similar software problem or propose a similar feature. The retrieval (i.e., identification and presentation) of duplicate issue reports has been mostly a manual procedure carried out by developers. Considering the large number of issues that are reported daily for popular software projects, the manual retrieval of duplicates requires a significant amount of effort and time. As an example, Mozilla reports in 2005 that "everyday, almost 300 bugs appear that need triaging[5]" (Anvik et al., 2006).

Duplicate issue reports are often considered a waste of developers' effort (Bettenburg et al., 2008b). For instance, imagine a developer who after days of debugging and verification realizes that she or he had been working on a duplicate issue report which was already fixed by another developer. Prior studies show that up to 20-30% of the issue reports are duplicates (Jalbert and Weimer, 2008). Such large amounts of duplicate issue reports have motivated extensive studies on the automated retrieval of duplicate issue reports (Anvik et al., 2005; Runeson et al., 2007; Wang et al., 2008; Jalbert and Weimer, 2008; Nagwani and Singh, 2009; Sun et al., 2011; Alipour et al., 2013; Hindle et al., 2015; Zou et al., 2016). The task of duplicate reports retrieval is a challenging task due to many reasons such as the natural language ambiguities in such reports. Various advanced approaches, such as Natural Language Processing approaches, are commonly used to determine whether a new issue report is a duplicate of a previously reported issues (Runeson et al., 2007).

[5]Issue triaging is the task of determining if an issue report describes a meaningful new problem or enhancement, so it can be assigned to an appropriate developer for further handling (Anvik et al., 2006).

1.1 Thesis Statement

Existing approaches for the automated retrieval of duplicate reports primarily focus on increasing their performance (i.e., accuracy). However, the evaluation of these approaches treat all the duplicates equally, apply experiments on unrealistic data and ignore the impact of newly activated features in ITSs. A revisiting of the experimental design choices for studies of these approaches is needed to gain a better understanding of the state of the field. Therefore, I propose the following thesis statement:

> **Thesis statement:** Revisiting the prior studies on automated approaches for retrieving duplicate issue reports can promote insights about the benefits and limitations of such studies' experimental design choices.

This thesis does not attempt to modify the algorithmic composition of previously proposed automated approaches. Instead, the thesis revisits how the experiments for these automated approaches are applied (e.g., the selection of the studied data and the used measures to capture the performance of such approaches).

1.2 Thesis Overview

Figure 1.1 shows an overview of this thesis's focus in contrast to the prior work. The thesis focuses on a set of experimental design choices for studies of approaches for the automated retrieval of duplicate reports. The thesis studies four design choices as illustrated in Figure 1.1. The studied choices are: 1) Needed effort, 2) Evaluation Process , 3) Data Filtration and 4) Data Changes. The contribution in each revisited

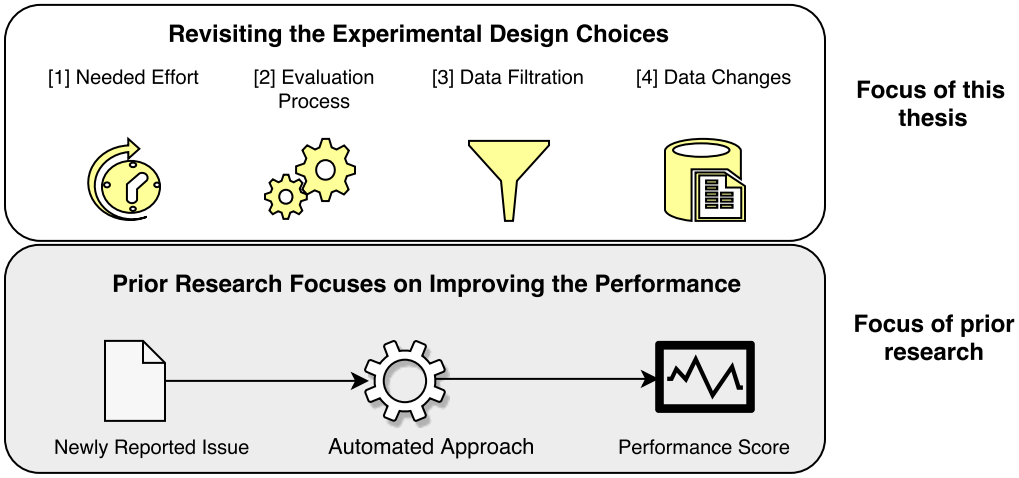


Figure 1.1: Thesis overview.

design choice is mentioned at Section 1.3. Below, I give an overview of the chapters in this thesis:

1.2.1 Chapter 2: Background on the Automated Retrieval of Duplicate Reports

First, I give an introduction to the usage of issue tracking systems. Then, I provide background on some key information retrieval concepts and how they are used for the automated retrieval of duplicate reports. In addition, I give a brief introduction about the automated approaches that I leveraged through the studies of this thesis. At the end of this chapter, I provide an overview of how the performance of such automated approaches are evaluated.

1.2.2 Chapter 3: Related Research

In this chapter, I present a literature review of the studies that focused on the problem of retrieving duplicate reports. Studies are grouped based on the following criteria:

- Empirical Studies of Duplicate Issue Reports.
- Automated Classification of Duplicate Issue Reports.
- Automated Retrieval of Duplicate Issue Reports.

From the surveyed related work, I gather the common experimental design choices that I believe might impact the reported findings of prior studies. First, I observe that all prior research treats all duplicate reports equally in terms of the needed effort. Secondly, a large majority of prior studies use an evaluation process that intentionally drops a large part of data leading to unrealistic performance measures. Finally, the prior studies totally ignore the impact of the newly activated features ITSs (i.e., just-in-time lightweight retrieval).

1.2.3 Chapter 4: A Study on Effort Needed for Identifying Duplicates

In this chapter, I empirically examine the effort that is needed for manually identifying duplicate reports. My results show that: (i) More than 50% of the duplicate reports are identified within half a day. Most of the duplicate reports are identified without any discussion and with the involvement of very few people: (ii) A classification model built using a set of factors that are extracted from duplicate issue reports classifies duplicates according to the effort that is needed to retrieve them with a precision of 0.60 to 0.77, a Recall of 0.23 to 0.96, and an area under the curve for Receiver Operating Characteristic

(ROC) (Hanley and McNeil, 1982) of 0.68 to 0.80, (iii) Factors that capture the developer awareness of the duplicate issue's peers (i.e., other duplicates of that issue) and textual similarity of a new report to prior reports are the most influential factors in my models. In short, I highlight the need for effort-aware evaluation of approaches that retrieve duplicate issue reports, since the identification of a considerable amount of duplicate reports (over 50%) appear to be a relatively trivial task for developers.

1.2.4 Chapter 5: Revisiting the Performance Evaluation of Automated Approaches for the Retrieval of Duplicate Issue Reports

In this chapter, I show that many prior studies leverage an evaluation that tends to overestimate the performance of automated approaches for retrieving duplicate issue reports. Instead, I propose a realistic evaluation using all the reports that are available in the ITS of a software project. I conduct experiments in which I evaluate prior approaches for the automated retrieval of duplicate reports using the classical and my proposed realistic evaluations. I highlight that the realistic evaluation shows that the previously proposed approaches perform considerably lower than previously reported using the classical evaluation. As a result, I conclude that the reported performance of approaches for retrieving duplicate issue reports is significantly overestimated in literature. In order to improve the performance of prior approaches, I propose to leverage the resolution field of issue reports. My experiments show a large relative improvement in the performance according to my proposed realistic evaluation.

1.2.5 Chapter 6: The Performance Impact of Just-in-time Duplicate Retrieval

This chapter investigates the impact of the "just-in-time duplicate retrieval feature" that has been activated in recent versions of ITSs. In particular, I study the differences between duplicate reports before and after the activation of this new feature. I 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. I find that duplicate issue reports after the activation of the just-in-time feature are less textually similar, have a greater identification delay and require more discussion to be identified as duplicate reports than duplicates before the activation of the feature. I observe that the performance gap between the automated approaches of prior research becomes even larger after the activation of the just-in-time duplicate retrieval feature. I recommend that future studies focus on duplicates that were reported after the activation of the just-in-time 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.

1.3 Thesis Contributions

In this thesis, I revisit a set of common experimental design choices of prior studies in order to highlight important aspects when evaluating an automated approach for the retrieval of duplicate issue reports. The results of the thesis experiments highlight the current state of progress in this area while charting directions for future research efforts. In particular, the thesis contributions are as follows:

1. Show the importance of considering the needed effort for retrieving duplicate reports in the performance measurement of automated duplicate retrieval approaches.

2. Propose a realistic evaluation of such approaches for the automated retrieval of duplicate reports, and analyze prior findings using this realistic evaluation.

3. Propose a genetic algorithm-based approach for filtering out the old issue reports to improve the performance of duplicate retrieval approaches.

4. Highlight the impact of the testing data changes after the “just-in-time duplicate retrieval feature” on performance evaluation of duplicate retrieval approaches.

CHAPTER 2

Background on the Automated Retrieval of Duplicate Reports

This chapter provides an introduction to the general usage of ITSs, duplicate issue reports and information retrieval concepts. This chapter also presents the performance measures that are used to commonly evaluate the performance of automated retrieval of duplicates.

2.1 Usage of Issue Tracking Systems

The general workflow of an issue tracking system is illustrated in Figure 2.1. Each issue has a unique id, reporter, status, attachments, report date, and other relevant data. Each reported issue must be triaged in order to get processed. After triaging, each issue is closely investigated to determine whether it is really

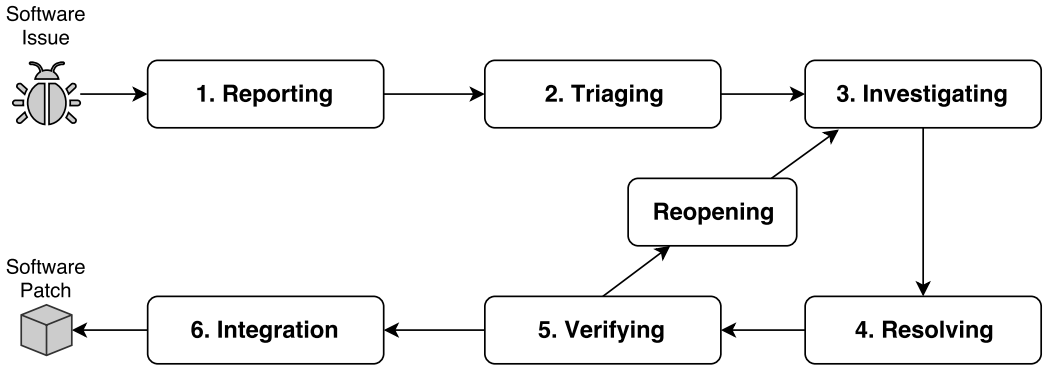


Figure 2.1: General workflow of an issue tracking system.

describing an actual problem and to assign the issue to the most appropriate developer for further investigation. Otherwise, the issue can be flagged as an invalid one or a duplicate of an existing issue. After resolving an issue, developers can reopen the issue during the verification step. If an issue is finally resolved and verified, it can be integrated into a future software release.

Prior research covers different steps in the workflow of issue tracking systems. I briefly below describe the various steps that are involved in the workflow of issue tracking systems while highlighting research efforts that are associated with each step.

1. Reporting. When an issue is reported, its *severity* is set to indicate the impact of the issue on the successful execution of the software system. A high severity issue signifies a fatal problem and a low severity issue implies a trivial one (BugzillaFields, 2010). The severity of an issue implicitly defines how urgently it needs to be resolved. Also, team leaders and project managers may set the severity based on their business view and internal plans. The severity of an issue plays an important role in deciding the handling order of issues. Lamkanfi et al. (2010) use a set of text mining algorithms on Mozilla, Eclipse and GNOME projects to automatically determine the severity of issues reports with a precision of 0.65-0.75 and a Recall of 0.70-0.85. Lamkanf *et al.* find that

the length of an issue's description does not impact its severity.

2. Triaging. Issue triaging is time consuming and researchers have proposed automation techniques to assist in triaging issues. Čubranić (2004) use text categorization to suggest the issue assignee using a Bayesian classifier. Čubranić's approach correctly suggests 30% of the issue report assignee on a collection of 15,859 issues. In addition, Anvik et al. (2005) use a Support Vector Machine(SVM) (Steinwart and Christmann, 2008) classifier to automate triaging. Anvik *et al.* report that an issue's assignee can be correctly suggested 57% of the time. Jeong et al. (2009) address the issue triaging problem from a different perspective which is the reassignment of the issue reports (i.e., also called issue tossing). Jeong *et al.* propose a Markov-based graph model which leverages the history of issue tossing to reduce tossing by up to 72% in the Mozilla and Firefox projects.

3. Investigating. During the investigation of issues, developers put issues into known categories and examine whether each issue is a duplicate of an existing issue. Issue categorization has taken considerable attention in recent research. Somasundaram and Murphy (2012) study the categorization of issues based on the components of the software. Somasundaram and Murphy compare different machine learning techniques to automatically determine to which software component each issue belongs. Somasundaram and Murphy find that Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and Kullback Leibler Divergence(KL) (Latecki et al., 2006) are the best performing models for determining the component of an issue. Thung et al. (2012) propose a model to categorize issues into the IBM Orthogonal Defect Classification (ODC) (IBM, 2016) with accuracy of 77%. Thung *et al.* rely on text mining of a report and code changes associated with an issue to automatically determine the category of an issue (e.g., a GUI

defect or logical defect).

Duplicate issue report retrieval is the task of matching a newly entered issue with an existing one. See Section 2.3 for more details.

4. Resolving. During the resolution of an issue, developers often find that some issues are given a higher priority, and/or that they block the fixing of other issues. (i.e., Blocking issues). Valdivia Garcia and Shihab (2014) detect blocking issues using features that are proposed by Shihab et al. (2013). Valdivia Gracia *et al.* find that blocking issues take significantly more time to fix in comparison to non-blocking issues. On six open source projects, their proposed model achieves an F-measure of 0.15 to 0.42.

5. Verifying. After resolving an issue, developers need to verify whether the issue is addressed correctly. Developers need to re-open the issue, if they find out that the issue is not addressed. Prior research investigates the reasons for the re-opening of issues. Zimmermann et al. (2012) study various factors that are associated with issue reopening in commercial systems. Zimmerman *et al.* found a wide array of factors affecting the likelihood of reopening an issue. Shihab et al. (2013) are the first to study issue reopening in open source software projects, namely Eclipse, Apache, and OpenOffice. Shihab *et al.* find that the comment text of an issue report is the most important factor for predicting issue reopening for Eclipse and OpenOffice projects, while the last status of an issue report is the most important factor for the Apache project.

6. Integration. A fix to an issue is integrated into a future release of the software. However, there is always integration delay of fixed issues, i.e., from the time when an issue is fixed to the time when the issue is integrated in the official releases of a software system. Integration delay is a point of great interest to software users. Costa et al. (2014) study the delay in the integration of the fixed issues. Costa *et al.* find that 34-98% of

fixed issues are delayed. They built a model to predict whether a fixed issue will be integrated to the next release, after the next release, after two releases, or more. They find that the integrator workload has the highest impact on the integration delay.

2.2 Duplicate Issue Reports

In the ITS of a software project, developers, testers and users can report issues that describe software bugs, new features or improvements. To report an issue, users fill in a form with details about that issue. As this form contains several free text fields, users can describe the same issue in different ways. Thus, the same issue may be reported multiple times. To avoid wasting resources, all duplicate issue reports must be marked as DUPLICATE by a triager. Then, developers that want to address an issue report do not have to worry about working on an issue that is already resolved or assigned to another developer.

In general, developers select one issue report of a set of duplicate reports to survive, i.e., the master report. This master report is often the report that contains the most detailed information about the issue, such as developer discussions. Hence, the goal of retrieving duplicate issue reports is to link all newly-reported issues to their corresponding master report if they are duplicates. In contrast to manual retrieval, prior work on automated approaches for retrieving duplicate issue reports classifies the oldest issue report of a set of duplicate reports as the master report (Runeson et al., 2007; Sun et al., 2010, 2011; Nguyen et al., 2012).

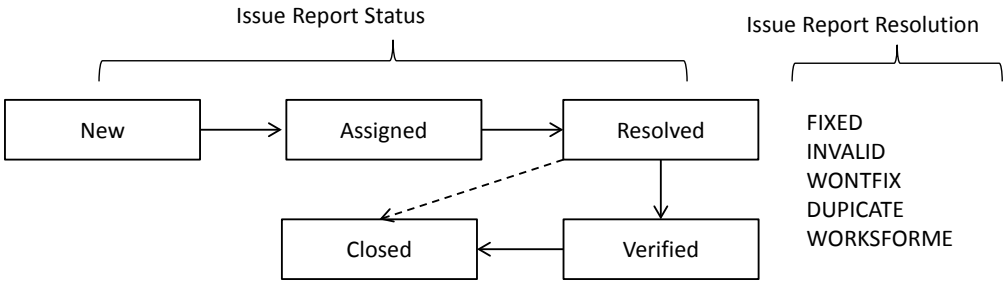


Figure 2.2: The life cycle of an issue report in Bugzilla.

2.2.1 Linking of Duplicate Issue Reports

In practice, the retrieval of duplicate reports is done manually. Whenever developers retrieve a duplicate issue report, they change the resolution status of a report to *DUPLICATE*, and add a reference (i.e., *Id-Number*) to the previously reported issue. An automatically-generated comment is attached to the discussion of the issue report in order to indicate that this issue report is a duplicate of another one. I use such automatically generated comments to retrieve duplicate issue reports. For example, if I find a message like ****This issue has been marked as a duplicate of B **** in the discussions of issue report A, I consider issue report A and B as duplicates of each other. I consider the latest *Id-Number* in case it was changed. I group duplicate issue reports. For example, if I find that issue report A and B are duplicates of each other and issue report B and C are duplicates of each other, I consider issue reports A, B and C as duplicates of each other. I repeat this grouping process, until I cannot add additional duplicate reports to a group.

2.2.2 Pre-processing of Duplicate Issue Reports

Figure 2.2 shows the life cycle of an issue report in Bugzilla (Koponen, 2006). First an issue is reported with status *NEW*. Next, a triaging process determines if an issue report should be assigned to a developer to fix. If the issue is assigned to a developer the status of the issue is changed to *ASSIGNED*. After the developer fixes the issue, the report status is changed to *RESOLVED-FIXED*. In some cases, an issue is not fixed because it is invalid (i.e., *RESOLVED-INVALID*), not reproducible (i.e., *RESOLVED-WORKSFORME*) or duplicate (i.e., *RESOLVED-DUPLICATE*). Finally, the issue is verified by another developer or by a tester (i.e., *VERIFIED-FIXED* or *CLOSED-FIXED*).

Figure 4.1 presents the typical process for managing a duplicate report. First, a duplicate report is triaged. Then, it gets retrieved as a duplicate. The un-duplication happens when the *DUPLICATE* resolution is removed. If a developer finds that a duplicate report is actually not a duplicate, the *DUPLICATE* resolution is removed. On average 2.5% of the duplicates in the studied projects are un-duplicated. I remove the un-duplicated issue reports from my analysis of duplicate reports. The removal is performed by ensuring that the last resolution status is one of the following statuses: *RESOLVED-DUPLICATE, VERIFIED-DUPLICATE* and *CLOSED-DUPLICATE*.

2.3 Automated Retrieval of Duplicate Issue Reports

To assist triagers with the tedious process of retrieving duplicate issue reports, automated approaches have been proposed (Runeson et al., 2007; Wang et al., 2008; Sun et al., 2011; Nguyen et al., 2012; Alipour et al., 2013; Aggarwal et al., 2015; Jalbert and Weimer, 2008; Hindle et al., 2015). Most of these approaches use information retrieval

techniques (Baeza-Yates and Frakes, 1992). Information retrieval is the activity of finding the needed information from a set of information sources (the *corpus*), such as text documents, images and audio. Most information retrieval approaches calculate a score that is based on the similarity of the available information sources to a query.

To calculate the similarity score of text documents, the documents are usually separated into terms, as in the vector space model (VSM) (Salton et al., 1975). A term can be a single word or a longer phrase. Each term has a value which represents its importance in a document. The terms within a document are ranked by their importance and the similarity score of a document is calculated based on that ranking.

Term Frequency-Inverse Document Frequency (**TF-IDF**) (Chowdhury, 2010) is a common approach 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. The basic formula of TF-IDF is:

$$\text{TF-IDF} = \text{TF}(t,d) * \text{IDF}(t,D) \tag{2.1}$$

Where $\text{TF}(t,d)$ is the frequency of the term t in a document d, while $\text{IDF}(t,D)$ 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. Several approaches are used to measure the distance between two TF-IDF vectors, such as the cosine similarity (Chowdhury, 2010), Dice similarity (Chowdhury, 2010), BLEU similarity (Papineni et al., 2002) and Jaccard similarity (Niwattanakul et al., 2013). In this

chapter, I perform my experiments using two popular approaches for retrieving duplicate issue reports that are based on TF-IDF: Best Matching 25F (BM25F) and REP (Sun et al., 2011; Nguyen et al., 2012; Zou et al., 2016).

BM25F (Robertson et al., 2004) is an advanced document similarity approach that is based on the TF-IDF vectors of documents (Berry and Castellanos, 2004; Robertson et al., 2004). The BM25F approach 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 them. BM25F considers the retrieval of structured documents which are composed of several fields (e.g., title, header and description). Each field in turn corresponds to a TF-IDF vector, to which 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.

In order 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[1] 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 I refer throughout this thesis as the *tuning data* for BM25F. In some parts of the thesis, I use the term *training data* when referring to the data that is needed to train the classifiers such as random forest (Breiman, 2001) (see Chapter 4) or the data used by genetic algorithm for optimization (Deb et al., 2002) (see Chapter 5).

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

REP (Sun et al., 2011) extends the BM25F approach into the BM25F$_{ext}$ by considering the frequency of the shared words in the new issue report and previously-reported issues, when searching for the best-matching reports. In addition, the REP approach includes the categorical fields of issue reports in the retrieval process, while BM25F does not include such fields. The REP approach calculates the similarities between the query and a document based on seven features. A feature is a measurable value of similarity between two issue reports, such as the similarity between the summary fields of the reports. Two features for the textual fields (i.e., summary and description) are calculated based on an extended TF-IDF formula of BM25F. The other five features represent the categorical fields (i.e., component, priority, product, type, and version). These features equal one if the field value in reports that are compared is 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:

$$\text{REP}\,(d,q) = \sum_{i=1}^{7} w_i \times \mathit{feature}_i \tag{2.2}$$

Here, d and q are two issue reports that are compared. The variable w_i is the weight for each feature. These weights are automatically optimized using a stochastic gradient descent algorithm (Taylor et al., 2006), which uses a set of duplicate reports to find these weights. I refer throughout this thesis to that set of duplicate reports as the *tuning data* for REP. The $\mathit{feature}_i$ 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*). Table 2.1 shows an example of a REP

Table 2.1: Example of a list of results returned by REP for OpenOffice.

	ID	Title	REP
Query	90892	Copying and pasting appends line break	
Rank1	55631	Cannot copy from Writer to any other software unless pasted	22.38
Rank2	90511	Linefeed added by copy & paste	20.45 (*)
Rank3	67683	Can't paste from clipboard after copying from certain programs	20.02
Rank4	45970	Copying outline entry to clipboard copies things that weren't highlighted	19.57
Rank5	81023	cropped image loses crop when copy/pasted	19.46

(*) The correct duplicate candidate for the query.

ranking for a newly-reported duplicate issue #90892 in OpenOffice[2]. The result is a list of duplicate candidates based on their REP score. In this example, REP returned the correct master report #90511 marked by $(*)$ at rank 2 in the top 5 list.

2.3.1 Performance Evaluation Measures

In this thesis, I compute two frequently used performance measures to evaluate the studied automated approaches for the retrieval of duplicate issue reports (Sun et al., 2010, 2011; Nguyen et al., 2012; Hindle et al., 2015): 1) Recall rate, and 2) Mean Average Precision (MAP). The *Recall rate* is defined as:

$$\text{Recall}_{\text{topN}} = \frac{retrieved_{topN}}{retrieved_{topN} + missed_{topN}} \tag{2.3}$$

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,

[2]Apache OpenOffice Webpage: https://www.openoffice.org/

the higher the Recall rate. The Recall rate value ranges from 0 to 1. In this thesis, I 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:

$$\text{MAP} = \frac{1}{Q}\sum_{n=1}^{Q}\frac{1}{rank(n)} \tag{2.4}$$

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 thesis, I limited the MAP calculation to a list size of 1,000 candidates.

2.4 Chapter Summary

This chapter provides a background on the general usage of issue tracking systems in software engineering. In addition, I illustrate the basic concepts of information retrieval that are leveraged for the automated retrieval of duplicate issue reports.

CHAPTER 3

Literature Review

In this chapter, I explore and classify prior studies that are related to this thesis. I present the main contribution of each study and compare it with its prior studies.

3.1 Revisiting Prior Research

There are several studies that revisit concepts and findings of prior research.

3.1.1 Revisiting Studies on Defect Prediction

Defect prediction uses machine-learning and statistical analysis to try to guess whether a piece of code has a defect (i.e., buggy) or not. Kamei et al. (2010) revisit the performance measures of bug prediction models using additional measures which capture software quality assurance effort. In particular, Kamei *et al.* study bug prediction using effort awareness. Similar to the prior research, Kamei et al show that the process metrics outperform product metrics when the effort is taken into consideration in file level models of bug prediction. However, unlike prior research, package level predictions do not outperform the file-level predictions when considering effort awareness. Ghotra et al. (2015) argue that classification techniques can have a large impact on the performance of defect predictions. Different from the prior research (Lessmann et al., 2008), Ghotra et al.s' results suggest that some classification techniques produce defect prediction models that outperform others.

3.1.2 Revisiting Studies on Mobile Apps

Syer et al. (2013) analyze 15 open source apps to investigate the differences between mobile apps and five desktop/server applications. Syer *et al.* use two dimensions in their comparison: the defect resolution time and the size of each app. Syer *et al.*'s results suggest that mobile apps are similar to UNIX utilities in terms of size of the code and the development team.

3.1.3 Revisiting Studies on Code Review

Code review is a phase in the software development process in which the developers and authors get together to review code and correct its errors. Thongtanunam et al. (2016) revisit the relationship between code ownership and software quality from a code review perspective. On six releases of QT and OpenStack systems, Thongtanunam *et al.* find that non active developers still contributes to the reviewing of code changes. The study results illustrate that including the code review features along with the code ownership features exhibits a relationship with the software quality. Thongtanunam *et al.* suggest that code reviewing activities capture an important aspect of code ownership and should be considered by future research efforts.

3.1.4 Revisiting Studies on Development Teams

Yamashita et al. (2015) revisit the implication of the Pareto principle (Goeminne and Mens, 2011) on set of open-source projects from GitHub. Yamashita *et al.* find that the Pareto principle does not apply for 40%-87% of GitHub projects. Yamashita *et al.* suggest that the Pareto principle is not compatible with many GitHub projects.

In this thesis, I focus on revisiting prior research that proposes new automated approaches for duplicate reports retrieval by examining their used evaluation. In the next section, I survey studies that are related to duplicate issue reports.

3.2 Studies on Duplicate Issue Reports

Newly submitted issue reports usually go through a filtration process before they are assigned to developers. One of the common filtering steps is to determine whether a

new-reported issue is a duplicate of an existing issue. Two issue reports are considered duplicates of each other, if they refer to a similar software problem or propose a similar feature. Duplicate issue reports are considered a waste of developers' effort. In this subsection I survey prior studies on duplicate issue reports.

3.2.1 Characteristics of Duplicate Issue Reports

Anvik et al. (2005) report that 20-30% of the issue reports in the Eclipse and Firefox projects respectively are duplicates. Cavalcanti et al. (2013) find that duplicate reports represent 32%, 43% and 8% of all the reports in the Epiphany, Evolution and Tomcat projects, respectively.

Duplicate reports are not always harmful. Bettenburg et al. (2008b) find that merging the information across duplicate reports produces additional useful information over using the information from a single report. In particular, Bettenburg *et al.* find that such additional information improves the accuracy of automated approaches for issue triaging by up to 65%.

Several prior studies explored the effort that is associated with duplicate reports. Davidson et al. (2011) and Cavalcanti et al. (2010, 2013) examine the effort that is associated with closing a duplicate report: Davidson *et al.* look at the time that is needed for closing a duplicate report after it was triaged, and Cavalcanti *et al.* examine the total time that is needed to close a duplicate report. Cavalcanti *et al.* also investigate the time that is spent by report submitters to check whether they are about to file an issue that is a duplicate of previously-reported issues.

3.2.2 Determining Whether a Newly-Reported Issue was Previously-Reported

Hiew (2006) proposes an approach that labels a newly-reported issue as 'duplicate' or 'unique'. His approach extracts a TF-IDF vector from the summary and description field of the newly-reported issue, and groups similar issues together in so-called 'centroids'. Based on a threshold for the similarity of the vectors within a centroid, a new issue report is labeled as a duplicate or unique report.

Feng et al. (2013) propose an approach to determine whether two issue reports are duplicates of each other. Their approach uses a classifier (such as Naive Bayes, Decision Tree or SVM) to determine whether a newly-reported issue is a duplicate, based on the (1) profile of the reporter and (2) the list of candidate reports that were retrieved while entering the report. The intuitions behind using this data are (1) some reporters may tend to submit more duplicate reports, as they do not use the search functionality, and (2) due to the writing style of a reporter, the candidate reports that are retrieved may be similar but not duplicates of the newly-reported issue.

Banerjee et al. (2016) use an approach that is similar to Feng et al.'s work. However, Banerjee et al. use a random forest classifier along with 24 document similarity measures and evaluate their approach on a much larger dataset.

3.2.3 Automated Retrieval of Duplicate Issue Reports

The Retrieval of Duplicate Issue Reports is the task of matching a newly-reported issue with one or more previously reported ones. Prior research proposed various duplicate retrieval approaches that leverage textual and non-textual information that are derived from issue reports. Below, I group the previously proposed automated approaches for

duplicate reports retrieval based on the type of used features.

Approaches Based On Textual Similarity.

Anvik et al. (2005) propose an approach that uses cosine similarity to classify a newly-reported issue as either a new issue or a duplicate one. Their approach can correctly retrieve 28% of the duplicate issue reports in Firefox 1.0. Runeson et al. (2007) propose an approach to retrieve duplicate reports based on Natural Language Processing (NLP) and evaluated the approach on issue reports from Sony Ericsson. The prototype tool correctly classified 40% of the duplicate issue reports. Nagwani and Singh (2009) use string similarity measures, such as TF-IDF similarity, and text semantics measures, to retrieve duplicate issue reports. Prifti et al. (2011) propose an approach based on information retrieval, while limiting the search for duplicates to the most recently filed reports. Prifti *et al.*'s approach is able to retrieve around 53% of the duplicate issue reports in their case study.

Sun et al. (2010) build 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. Sun *et al.*'s approach outperforms the prior state of the art approaches Runeson et al. (2007); Jalbert and Weimer (2008); Wang et al. (2008) by 17-31%, 22-26%, and 35-43% on OpenOffice, Firefox, and Eclipse projects, respectively. Sureka and Jalote (2010) use a character n-grams approach (Kanaris et al., 2007) to measure the text similarity between the titles and descriptions of issue reports. Sureka *et al.*'s approach is language independent and achieves a Recall rate of around 62% for 2,270 randomly selected reports from the Eclipse project.

Wang et al. (2008) combine textual information and execution traces to retrieve duplicate issue reports. The execution information (i.e., stack traces) has low influence from the variety of natural language (i.e., the stack traces are generated by the software itself and not by users) and can provide additional technical details for an issue. For two issue reports, Wang *et al.* calculate two separate similarity measures: one measure for textual similarity and another measure to capture the similarity of the execution traces. By combining both similarity measures, a list of possible duplicate issue reports is suggested for each new-reported issue. Wang et al. (2008)'s approach achieves an accuracy of 67% to 93%. Lerch and Mezini (2013) propose an automated duplicate retrieval approach that is only based on the stack traces that are attached to issue reports. Lerch and Mezini (2013) use the TF-IDF similarity of stack traces for retrieving duplicate reports (around 10% of reports contain stack traces).

Approaches Based On Textual and Categorical Similarity.

Other information in issue reports, such as the component, version, Bug_Id, and platform, help improve the accuracy of approaches (Shah and Croft, 2004). Jalbert and Weimer (2008) propose an approach that combines textual similarity with categorical features that are derived from issue reports. Their approach is able to filter out 8% of the duplicate issue reports while allowing at least one report for each unique issue to reach developers. Jalbert *et al.*'s approach achieves a Recall of 51%. Kaushik and Tahvildari (2012) compare the performance of word-based and topic-based information retrieval models using categorical features (e.g., the component field of an issue report) and textual features. For the word based models, Kaushik *et al.* apply different term weighting approaches including those proposed by Runeson et al. (2007) and Jalbert

and Weimer (2008). While for the topic based models, Kaushik *et al.* apply Latent Semantic Indexing (LSI) (Deerwester et al., 1990), Latent Dirichlet Allocation (LDA) (Blei et al., 2003), and Random Indexing (Kanerva et al., 2000) approaches. Kaushik *et al.*'s approach is applied to the Mozilla and Eclipse projects. The results show that the word-based approaches outperform the topic-based approaches. Sun et al. (2011) leverage a BM25F model (Robertson et al. (2004)) to combine both textual and categorical information for retrieving duplicate issue reports. Sun *et al.* achieve an improvement of 10-27% in Recall in comparison to Sureka and Jalote (2010)'s approach.

Approaches Based On Topic Modeling

Nguyen et al. (2012) propose an approach called DBTM to measure the similarity between issue reports sharing the same topics in addition to BM25F similarity. Nguyen *et al.* propose an LDA-based approach called T-Model which extracts the topics for the issue reports. The T-Model parameters are estimated from the duplicate reports relations in train phase. For each new issue reports, T-Model finds the possible duplicates that have the most similarity in terms of topics. To combine BM25F similarity and topic based similarity a machine learning approach called Ensemble Averaging is applied. Nguyen *et al.* approach improves the accuracy of Sun et al. (2011)'s approach by up to 20% in accuracy.

Alipour et al. (2013) propose an approach which uses software dictionaries and word lists to extract the implicit context of each issue report. Alipour *et al.* show that the use of contextual features improves the accuracy of duplicate retrieval by 11.55% over Sun et al. (2011)'s approach on a large dataset of Android projects. However,

Table 3.1: A summary of the surveyed automated approaches for duplicate retrieval.

Works	Features	Approach	Results	Used Datasets	Limitations
Runeson et al. (2007)	Textual Only.	Used a vector space approach along with three text similarities which are Cosine, Dice, Jaccard to compute the text similarity between issue reports	Achieved a Recall rate of 40%	Sony Ericsson dataset	Approach achieves low accuracy and is evaluated only on one dataset
Jalbert and Weimer (2008)	Textual and Categorical	Used a TF-IDF vector space approach for textual features and categorical features	Achieved a 1% improvement over Runeson et al. (2007)	29,000 issue reports from Mozilla	Small improvement in duplicate retrieval accuracy
Wang et al. (2008)	Textual Only	Used a TF-IDF vector space model for textual features and execution information that are derived from issue reports	Achieved a Recall rate of 67-93%	200 issue reports from Eclipse and 1749 issue reports from Mozilla	Depends on the availability of execution information and evaluated on a small set of issue reports
Sun et al. (2010)	Textual Only	Used a TF-IDF vector space approach along with support vector machine (SVM) classifier	Achieved an average of 29% relative improvement over Runeson et al. (2007), Jalbert and Weimer (2008), and Wang et al. (2008)	Datasets from Mozilla, Eclipse and OpenOffice in various periods of years	Ignores categorical features which are used in previous research Runeson et al. (2007); Jalbert and Weimer (2008); Wang et al. (2008)
Sun et al. (2011)	Textual and Categorical	Leveraged a BM25F approach to combine textual and categorical information for retrieving duplicate issue reports	Achieved an average of 19% relative improvement over their previous work Sun et al. (2010)	Datasets from Mozilla, Eclipse and OpenOffice in a period of one to three years	Requires more parameters tunning in contrast to previous work by Sun et al. (2010)
Nguyen et al. (2012)	Topics	Combined the BM25F approach with the similarity between issue reports which share the same topics	Achieved on average of 20% relative Recall rate improvement over Sun et al. (2011)'s approach	Same datasets as Sun et al. (2011)	Needs optimization for the number of LDA topics
Alipour et al. (2013)	Topics	Used the BM25F approach along with contextual similarity for duplicate vs non-duplicate pair classification	Adding contextual features achieved an improvement of 11.55% over the features used by Sun et al. (2011)	Android dataset in a period of two years	Approach requires manual effort to create word lists in order to leverage the contextual features. The newly proposed features are only tested on a randomly selected set of non-duplicate reports

Alipour *et al.*'s approach only classifies the issue reports into duplicates or randomly selected non-duplicates and does not suggest a list of top possible duplicates such as in Sun et al. (2011). Aggarwal et al. (2015) propose a duplicate retrieval approach based on the contextual information that is extracted from software-engineering textbooks and project documentation. Aggarwal *et al.*'s approach achieves a lower accuracy than the approach by Alipour et al. (2013). However, Aggarwal *et al.*'s approach is simpler with up to six times less computation time on the same Android dataset. Lazar et al. (2014) propose a duplicate retrieval approach based on new textual features that capture WordNet augmented word-overlap and normalized sentence-length differences. These additional features are generated using TakeLab (Šarić et al., 2012). Lazar *et al.* also use categorical features such as component and open_date. The added TakeLab features achieve an accuracy improvement between 3.25% and 6.32% over the Alipour et al. (2013)'s approach in classifying duplicate issues from a randomly selected set of non-duplicate issues.

Practical Usage of Automated Approaches for Duplicate Retrieval

Some issue tracking systems started to use filing-time filtering approaches for duplicate retrieval. For example, with version 4.0 Bugzilla started using Full-Text Search[1] at filing time. The Full-Text Search uses the classic Vector Space Model with a variant formula to weight the words (terms). Additionally, other tools were developed as plugins to automatically retrieve duplicates. Thung et al. (2014) propose a tool named DupFinder that implements the state-of-the art unsupervised approach by Runeson

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

et al. (2007). Thung *et al.* use the issue reports' summary and description fields as features to measure the similarity of a newly filed issue report. The tool is available on Github as a Bugzilla extension and is implemented in Perl. Rocha et al. (2015) propose a tool named NextBug for recommending similar issue reports based on the approach by Runeson et al. (2007) similar to DupFinder tool. However, the goal of NextBug tool is to increase the productivity of developers by taking advantage of the similar context across issues. NextBug is available as a plug-in for Bugzilla.

Conclusion

This chapter presents a literature survey of prior research on duplicate issue reports. I find that the existing studies treat all the duplicate issue reports equally regardless of how hard they might be to retrieve. In addition, a large portion of recent studies use small time periods (or datasets) to test their approaches while completely ignoring all the previous issue reports from those time periods without any empirical support on the impact of such decisions. Finally, prior studies do not take into consideration recent features in ITSs and the impact of such feature on the problem of duplicate issue reports.

CHAPTER 4

Studying the Needed Effort for Identifying Duplicates

In this chapter, I empirically examine the effort that is needed for manually identifying duplicate reports. My results show that: (i) More than 50% of the duplicate reports are retrieved within half a day. Most of the duplicate reports are identified without any discussion and with the involvement of very few people; (ii) A classification model built using a set of factors that are extracted from duplicate issue reports classifies duplicates according to the effort that is needed to identify them with a precision of 0.60 *to* 0.77*, a Recall of* 0.23 *to* 0.96*, and an ROC area of* 0.68 *to* 0.80*; and (iii) Factors that capture the developer awareness of the duplicate issue's peers (i.e., other duplicates of that issue) and textual similarity of a new report to prior reports are the most influential factors in my models. My findings highlight the need for effort-aware evaluation of approaches that retrieve duplicate issue reports, since the identification of a considerable amount of duplicate reports (over 50%) appear to be a relatively trivial task for developers. To better assist developers, research on automatically retrieving duplicate issue reports should put greater emphasis on assisting developers in identifying effort-consuming duplicate issues.*

An earlier version of this chapter is published in the Empirical Software Engineering Journal (EMSE) (Rakha et al., 2015).

4.1 Introduction

While prior research aims to reduce the needed efforts for identifying duplicate issue reports, there exist no studies that examine the actual effort that is spent on manually identifying the duplicate issues in practice. In fact, the needed effort for identifying duplicate issue reports may vary considerably. For example, a Mozilla developer needed less than 15 minutes to identify that issue #312782[1] is a duplicate of a previously reported issue, while the duplicate identification of issue #65305[2] required around 44 days, and involved 20 comments from 11 people. To better understand the effort that is involved in identifying duplicate issue reports in practice, this chapter studies duplicate reports from four open source projects (Firefox, SeaMonkey, Bugzilla and Eclipse platform). In particular, I explore the following research questions:

RQ1: How much effort is needed to identify a duplicate issue report?

> Half of the duplicate reports are identified in less than half a day. 50 - 60% of the duplicates are identified without any discussion.

RQ2: How well can we model the needed effort for identifying duplicate issue reports?

> Random forest classifiers trained using factors derived from duplicate issue reports achieve a precision of 0.60 to 0.77, and a Recall of 0.23 to 0.96, along with an ROC area of 0.68 to 0.80.

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=312782
[2] https://bugzilla.mozilla.org/show_bug.cgi?id=65305

RQ3: What are the most influential factors on the effort that is needed for identifying duplicate issue reports?

> The textual similarity between a new report and previously reported issues plays an important role in explaining the effort that is needed for identifying duplicate reports. Another influential factor is the team's awareness of previously reported issues. Such awareness is measured using: a) The experience of a developer in identifying prior duplicates, b) The recency of the duplicate report relative to the previously filed reports of that issue, and c) The number of people that are involved in the previously filed reports of that issue.

My findings show that developers are able to identify a large portion (over 50%) of duplicate reports with minimal effort. On the other hand, there exist duplicate reports that are considerably more difficult to identify. These findings highlight the need for effort-aware evaluation of automated approaches for duplicate identification. To better assist developers, research on identifying duplicate issue reports should put greater emphasis on assisting developers in identifying effort-consuming duplicate issues.

4.2 The Main Contributions of this Chapter

This chapter highlights the differences between the duplicate issue reports in terms of the needed effort for manual identification. The empirical study in this chapter shows that the duplicate reports can be classified into two groups (i.e., hard and easy) based on the effort needed for manual identification. In particular, my results emphasize that

future studies should consider the effort needed for identification when evaluating the automated approaches for duplicates retrieval.

4.3 Related Work

In this section, I present prior research that relates to this study.

4.3.1 Automated Retrieval of Duplicate Issue Reports

There are several automated approaches for duplicate reports retrieval proposed by prior research (see Chapter 3). Although prior research sought to accurately retrieve duplicate issue reports with advanced approaches, there exists no research which examines whether retrieving duplicate issue reports is indeed consuming a large amount of developer effort. Hence in this chapter, I do not aim to propose yet another duplicate retrieval approach. Instead, I study the needed effort for identifying duplicate issue reports in order to understand how and whether developers can make good use of previously proposed automated approaches for duplicate retrieval in practice.

4.4 Case Study Setup

In this section, I present the studied projects and the process of preparing the data that is used for my case studies. I share my data and scripts as an online replication package[3].

[3]Replication package: http://sailhome.cs.queensu.ca/replication/EMSE2015_DuplicateReports/

Table 4.1: An overview of the issue reports in the studied projects.

Project	# Issues	# Duplicates	Period
Firefox	90,128	27,154 (30.1%)	Jun 1999 - Aug 2010
SeaMonkey	88,049	35,827 (40.7%)	Nov 1995 - Aug 2010
Bugzilla	15,632	3,251 (20.0%)	Sep 1994 - Aug 2010
Eclipse-Platform	85,382	15,044 (17.6%)	Oct 2001 - Jun 2010

4.4.1 Studied Projects

I use the issue reports that are stored in the Bugzilla issue tracking system from four open-source projects: Firefox, SeaMonkey, Bugzilla, and Eclipse-Platform. All four projects are mature projects with years of development history. Table 4.1 shows an overview of the studied projects. Firefox, SeaMonkey and Bugzilla are open-source projects from the Mozilla foundation. These projects have been frequently studied by prior research for evaluating the automated approaches for duplicate identification (Anvik et al., 2005). Eclipse-Platform is a sub-project of Eclipse, which is one of the most popular Java Integrated Development Environments (IDEs). The duplicate issue reports in the Eclipse project have also been studied by prior research. For example, Bettenburg et al. (2008a) used the Eclipse dataset to demonstrate the value of merging information across duplicate issue reports. I studied the projects up to the end of 2010 because Bugzilla 4.0 (released in February 2011) introduced an automated duplicate retrieval feature when filing an issue[4]. Such an automated retrieval feature may bias my measurement of the needed effort for identifying duplicate reports. For example, many trivial duplicate reports might not be filled since Bugzilla would warn about them at the filing time of the report.

[4]Release notes for Bugzilla 4.0: https://www.bugzilla.org/releases/4.0/release-notes.html

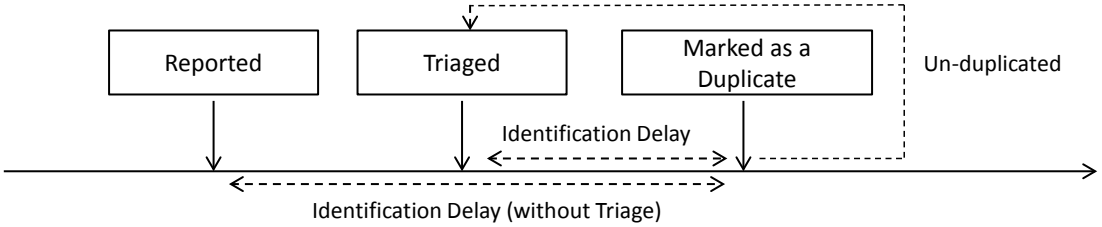


Figure 4.1: A typical timeline for managing duplicate issue reports.

4.5 Results

In this section, I present the results of my research questions. The presentation of each research question is composed of three parts: the motivation of the research question, the approach that I used to address the research question, and my experimental results.

RQ1: How much effort is needed to identify a duplicate issue report?

Motivation. There exist a considerable amount of duplicate reports. Prior research shows that around 20-30% of the issue reports in Mozilla and Eclipse are duplicates (Bettenburg et al., 2007). Duplicate issue reports are a waste of developers' time and resources. Thus, many researchers have proposed automated approaches for duplicate report retrieval. However, the effort that is needed for manually identifying duplicate reports has never been investigated. Therefore, I will examine whether identifying duplicate reports consumes a considerable amount of developers' effort in practice.

Approach. To measure the effort of identifying duplicate issue reports, I calculate the following three measures:

- *Identification Delay.* I calculate the identification delay for each duplicate report (*see* Figure 4.1). This measure provides us a rough estimate of the needed effort. The more time it takes to identify a duplicate report, the more developers' effort

is needed. For reports that are triaged, I measure the time in days between the triaging of the report and when the report is marked as a duplicate. For reports that are never triaged, I count the number of days between the reporting of the report and when the report is marked as a duplicate.

- *Identification Discussions.* I count the number of comments posted on an issue report before it is identified as a duplicate report. The more discussions about a particular report, the more complex and effort consuming it is to identify a duplicate. I ignore the comments that are not representing the issue report discussions. For example, all auto-generated comments, such as *Created an attachment*, are filtered. Then, in the case of duplicates with one remaining comment, I filter that comment if it is posted by the same reporter of the issue in order to add missing information.
- *Involved People.* I count the number of unique people that are involved in discussing each issue other than the reporter before the issue is marked as a duplicate. More people discussing a duplicate report indicates that more team time and resources are spent on understanding that particular duplicate report.

Results. **Most of the duplicate reports are identified within a short time.** I find that the median identification delay for the studied projects is between 0.18 to 0.83 days (*see* Table 4.2). Such results show that more than 50% of the duplicate reports are identified in less than one day. I do note that my identification delay metric is an over estimate of the actual time that is spent on identifying a duplicate report. Developers are not likely to start examining a report as soon as it is reported. Instead, there are many reasons for such activity to be delayed (e.g., reports are filed while developers are away from their desks, or developers are busy with other activities).

Table 4.2: Mean and Five number summary of the three effort measures.

Project	Metric	Mean	Min	1st.Qu	Median	3rd Qu.	Max
Firefox	Identification Delay (Days)	34.74	0.00	0.02	0.20	2.84	2042
	Identification Discussion (Count)	1.20	0.00	0.00	0.00	1.00	230
	Involved People (Count)	1.59	0.00	1.00	1.00	2.00	37
SeaMonkey	Identification Delay (Days)	35.59	0.00	0.03	0.18	3.51	3245
	Identification Discussion (Count)	1.52	0.00	0.00	0.00	2.00	111
	Involved People (Count)	1.75	0.00	1.00	1.00	2.00	75
Bugzilla	Identification Delay (Days)	92.84	0.00	0.02	0.32	27.41	3132
	Identification Discussion (Count)	1.82	0.00	0.00	0.00	2.00	135
	Involved People (Count)	1.62	0.00	1.00	1.00	2.00	43
Eclipse-Platform	Identification Delay (Days)	57.49	0.00	0.06	0.83	13.83	2931
	Identification Discussion (Count)	1.60	0.00	0.00	0.00	2.00	58
	Involved People (Count)	1.80	0.00	1.00	1.00	2.00	46

Most of the duplicate reports are identified without any discussion. I find that over 50% of the issue reports are marked as duplicates with no discussions, and over 75% of the issue reports are marked as duplicates with just one or two comments (*see* Table 4.2). Figure 4.2 shows the cumulative density function (CDF) plot of the discussions count for the four projects. The curve shows that at least 50% of the duplicates are identified without any discussions. I only show one CDF plot (the one for identification discussions) since the other CDF plots follow the same pattern.

Very few people are involved in identifying duplicates. Table 4.2 shows that for over half of the duplicate reports, the identification effort involves one person without counting the reporter of the issue.

Discussion. The results in Table 4.2 show that it takes little effort (i.e., time, discussions, and people) to identify the majority of the duplicate reports. However, it is also interesting to note that there are issue reports taking thousands of days to identify, with up to 230 comments, involving up to 75 people. Such issue reports consume a considerable amount of developers' efforts until developers realize that the issue reports are duplicates of previously-reported issues. Although various advanced approaches

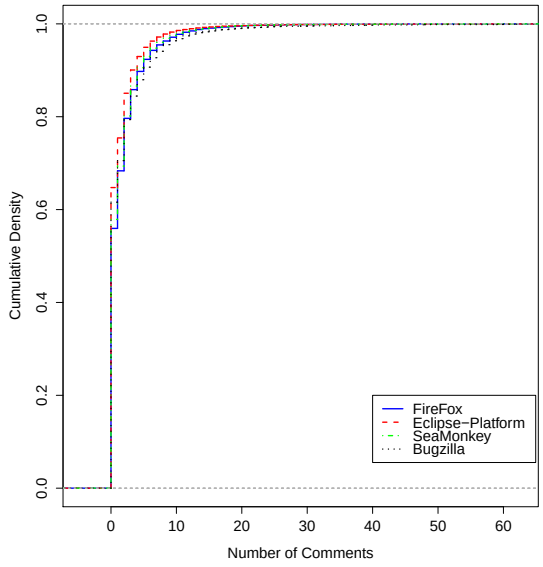


Figure 4.2: Cumulative Density Function (CDF) of identification discussions for the four studied projects. The x-axis shows the identification discussions, and y-axis shows the cumulative density

are proposed by prior research to automatically identify duplicate reports, those approaches do not differentiate between the easily-identifiable duplicate issue reports and the ones that are more effort-consuming to identify. Automated duplicate retrieval approaches should focus on helping developers identify duplicate reports that consume more effort to identify.

> Over half of the duplicate reports are identified in less than one day, with no discussion and the involvement of no one other than the reporter of the issue. On the other hand, there are duplicate reports that consume a large amount of effort to identify. Automated approaches for retrieving duplicates should focus on the ones that are hard to identify in order to better benefit practitioners.

Dimension	**Factors**	**Value**	**Definition**(d)|**Rationale**(r)
Peers	Peers Count	Numeric	d: The total number of peers that an issue report has. I only consider the peers that existed before the reporting of the new issue.
			r: An issue report with many existing peers might be easier to identify as a duplicate.
	Title Similarity	Numeric (0-1)	d: The maximum similarity between the title text of an issue report and title text of its peers. The similarity is measured using trigram matching.
			r: Similar title text as a previously reported issue may assist in identifying that an issue report is a duplicate of an existing one.
	Description Similarity	Numeric (0-1)	d: The maximum similarity between the description text of an issue report and the description text of its peers. The similarity is measured using trigram matching.
			r: Similar description text as a previously reported issue may assist in identifying that an issue report is a duplicate of an existing one.
	Peers CC List	Numeric	d: The sum of the unique persons in the CC lists of an issue report's peers.
			r: A larger peers' CC lists could be a sign of better awareness of an issue across the development team.

	Peers Comments	Numeric	d: The total number of comments that belong to the peers of an issue report. I only consider the comments that existed before the reporting of the issue report.
			r: If an issue report has well discussed peers (with more comments), the issue report may be easier to identify as a duplicate.
	Reporting Recency of Peers	Numeric	d: The smallest difference in days between the triage or report date of an issue report to the date when one of its peers are reported.
			r: If a developer has seen a similar issue that is reported recently, it could speed up the duplicate identification process.
	Recency of Peer Identification	Numeric	d: The smallest difference in days between the triage or report date of an issue report to the date when one of its peers is identified as a duplicate.
			r: If a developer has seen a similar issue that is identified recently, it could speed up the duplicate identification process.
	Recency of Closed Issue	Numeric	d: The smallest difference in days between the triage or report date of a duplicate report to the date when any issue report closed within the same project and component is closed.

			r: If a developer has recently fixed an issue within the same project and component, it could indicate the developer's activity and it could speed up the duplicate identification process within the same project and component.
Workload	Waiting Load	Numeric	d: The total number of other issues reports that are open at the time when that issue is reported within the same component and project.
			r: A large number of open issues could make developers too busy to recognize whether an issue is a duplicate.
Issue Report	Severity	Numeric	d: A number representing the severity of an issue report.
			r: More severe issue reports are likely to attract attention from developers, thus such high severity issues are likely to be identified as a duplicate faster.
	Priority	Numeric	d: A number representing the priority of an issue report.
			r: Issue report with higher priority are likely to attract attention from developers, thus such high severity issues are likely to be identified as a duplicate faster.
	Component	Nominal	d: The component specified in an issue report.
			r: Particular components may be more important than others. Hence issues associated with them are more likely to be identified as duplicates faster.
	Title Size	Numeric	d: The number of words in the title text.

			r: Issues with longer title might provide more useful information which might ease the duplicate identification.
	Description Size	Numeric	d: The number of words in the description of an issue report.
			r: Issues with longer description could be easier to understand and might ease the duplicate identification.
	isBlocking	Boolean	d: A boolean value indicating whether an issue report is blocking other issues.
			r: A blocking issue could be more important to address and this might lead to faster identification of duplicates.
	isDependable	Boolean	d: A boolean value indicating whether an issue report depends on other issues.
			r: An issue depending on other issues may have more attention from developers and get identified as a duplicate easier.
	Reproduce Steps	Boolean	d: A boolean value indicating whether the description of an issue report contains steps for reproducing the issue.
			r: Issues with reproduce steps should be easier to understand and might ease the identification of duplicates.

Work Habits	isWeekend	Boolean	d: A boolean value representing whether an issue is reported on a weekend day (i.e., Saturday and Sunday) or not.
			r: The duplicate reports submitted on weekends might take longer time to be identified than the ones on week days.
Identifier Experience	Total Duplicates Identified	Numeric	d: The total number of duplicate reports that are previously identified by the same identifier per each duplicate report.
			r: An identifier with more experience in finding duplicate reports might affect the effort needed for identifying a new duplicate report.
	Duplicate Peers Identified	Numeric	d: The total number of peers duplicate reports that are previously identified by the same identifier.
			r: An identifier that has seen several peer duplicates might identify duplicates faster.
	Fixed Issues per Identifier	Numeric	d: The total number of issue reports that are previously marked as FIXED by the same identifier.
			r: An identifier with more experience in fixing issues might identify duplicates faster.

Table 4.3: The factors used to model the effort needed for identifying duplicate reports.

RQ2: How well can we model the needed effort for identifying duplicate issue reports?

Motivation. The results of RQ1 show that while the majority of the duplicated reports are identified with little effort, the duplicate identification of some reports requires considerable amount of efforts. Understanding the effort that is needed to identify duplicate reports is important for managing project resources. In this RQ, I will build classifiers to determine the needed efforts to identify duplicate reports.

Approach. In order to build a classifier for the effort that is needed of identifying duplicates, I compute a set of factors from the duplicate reports. The factors are from five dimensions: *Peers*, *Workload*, *Issue Report*, *Work Habits*, and *Identifier Experience*. I describe each dimension below:

- **Peers:** Peers of a duplicate report are the issue reports that are duplicates of that report. In my work, I only include the peers that are reported before the reporting of an issue report, since only such peers can provide information to help identify that an issue report is a duplicate of previously-reported issues. The more information available from the peers of an issue report, the easier it is for developers to identify that an issue report is a duplicate.

- **Workload:** Workload refers to the number of open issues at the time of reporting or triaging of a duplicate report. If there is a high workload at the time of reporting an issue, the duplicate identification of a new-reported issue may be delayed.

- **Issue Report:** Issue report refers to the information that can be derived from an issue report at the time of its reporting. Better information in the issue report should help reduce the effort that is needed to identify that the issue report has

duplicates.

- **Work Habits:** Work habit refers to the time when an issue is reported. If an issue is reported during the weekends or at night, identifying that the report is a duplicate may be delayed.

- **Identifier Experience:** Refers to the possible information that a duplicate report identifier has. Duplicate reports identified by experienced persons might require less effort.

Table 4.3 presents the list of my studied factors along with their descriptions and my rationale for studying such factors. To calculate the factors *Title Similarity* and *Description Similarity* from the *Peers Dimension*, I first apply simple pre-processing steps, such as removing special characters, long spaces and converting all text to lowercase. Then, I apply a trigram algorithm (Lazar et al., 2014) to calculate the similarity between texts. 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. For the *Reproduce Steps* factor, I use a similar approach as Bettenburg et al. (2008a) by searching the description text for certain keywords, such as *steps to reproduce*, *steps*, *reproducible* and *reproduce*. I build individual models for every project because each project may have different ways for handling duplicate issue reports. For example, the triaging processes for the Mozilla and Eclipse projects are different. For Mozilla, triaged issue reports are not initially triaged to any developer. On the other hand, Eclipse issue reports are triaged at filing to a sub-group of developers

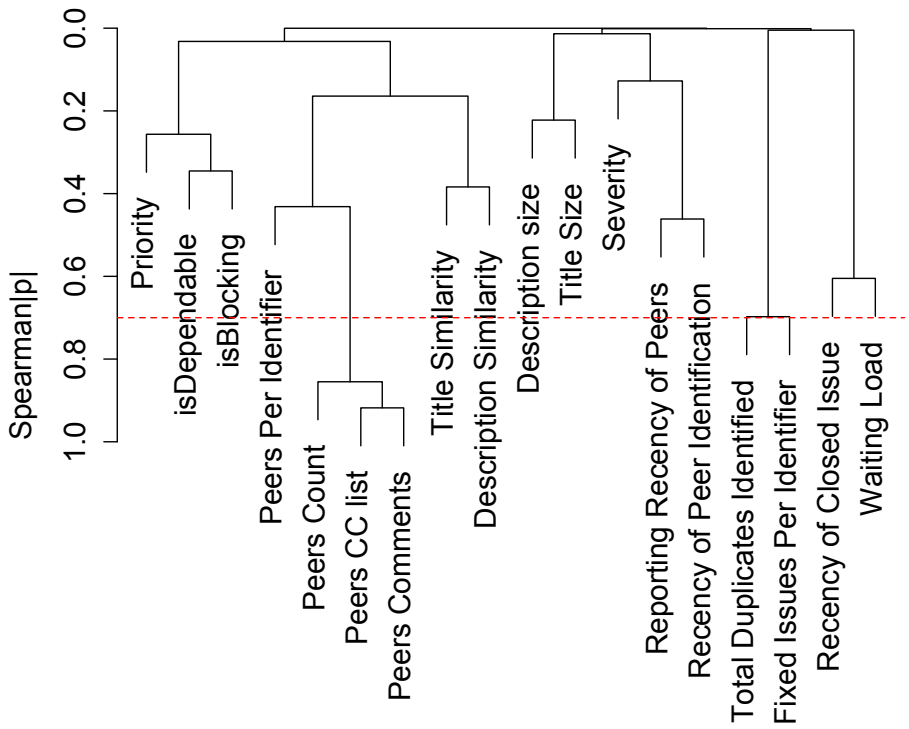


Figure 4.3: Spearman hierarchical cluster analysis for the Firefox dataset.

based on the project component. Only the developers responsible for a certain component receive the notifications about any new-reported issue. Based on this triaging process, Eclipse's developers may have a better chance to see a new-reported issue earlier than the developers of Mozilla (though the Eclipse developers might be overwhelmed with many irrelevant reports).

Correlation analysis. I use the varclus package (Marie Chavent and Benoit Liquet, 2016) in *R* to display Spearman rank correlations (ρ) (Zar, 1972) between my factors

Table 4.4: Spearman correlations between the three effort measures in the four studied projects.

Correlation	Firefox	SeaMonkey	Bugzilla	Eclipse-Platform
Identification Delay vs. Identification Discussions	0.54	0.55	0.56	0.45
Identification Delay vs. Involved People	0.54	0.52	0.53	0.36
identification Discussions vs. Involved People	**0.78**	**0.81**	**0.81**	**0.68**

given the large number of factors in my study. Varclus does a hierarchical cluster analysis on the factors using Spearman correlations. If the correlation between two factors is equal to or greater than 0.7, I ignore one of them since I consider a correlation of 0.7 as a high correlation (McIntosh et al., 2015; Mockus and Weiss, 2000). Figure 4.3 shows the output of varclus where the red dotted line marks the 0.7 Spearman correlation. Looking at Figure 4.3, I notice that the factors *Peers Counts*, *Peers CC List*, and *Peers Comments* are highly correlated with a correlation range of $0.85-0.91$. Hence, I need to pick just one of these three factors and discard the other two. Based on the results of Varclus (*see* Figure 4.3), I remove *Peers Comments*, *Peers Counts*, and *Fixed Issues per Identifier* factors due to the high correlation especially in cases such as between *Peers CC List* and *Peers Comments*. After this step, 18 of 21 factors remain.

Modeling technique. I build two classifiers to understand the effort that is needed for identifying duplicate reports. In RQ1, I have three measures of the effort that is needed for identifying duplicate reports. However, the *Involved People* and the *Identification Discussions* factors are highly correlated. Table 4.4 presents the Spearman correlations between the three effort factors for the studied projects. Hence, I only build two classifiers, one classifier for *Identification Delay* and another one for *identification Discussions*.

Table 4.5: Criteria for each class based on Identification Delay and Identification Discussions

	Identification Delay Classes
Slow	includes all duplicate reports with identification delay that is more than one day.
Fast	includes all duplicate reports with identification delay that is less than or equal to one day.
	Identification Discussions Classes
Not-Discussed	includes all duplicate reports that did not have any comments.
Discussed	includes all duplicate reports with one or more comments.

I divide the issue reports into two classes (*Slow* and *Fast*) based on the *Identification Delay*. All studied projects are open-source projects, where developers may not be able to examine an issue as soon as it is reported. In RQ1, I find that over half of the duplicate reports are identified within one day. Therefore, I choose one day to be the threshold to determine whether a duplicate report is identified *Slow* or *Fast*.

Similarly, I divide the issue reports into two classes (*Not-Discussed* and *Discussed*) based on the number of *Identification Discussions*. I consider that the issue reports that are identified as a duplicate without discussion as the ones requiring minimal effort.

Table 4.5 summarizes the criteria for each class based on the *Identification Delay* (Slow, Fast) and *Identification Discussions* (Not-Discussed, Discussed). Table 4.6 presents the distribution of duplicate reports for each class per each project.

Table 4.6: The number of duplicate reports from each project that belongs to each class.

Project	Not-Discussed	Discussed	Total
Firefox	16,686 (64.7%)	9,088 (35.3%)	25,774
SeaMonkey	19,535 (57.9%)	14,214 (42.1%)	33,749
Bugzilla	1,909 (61.6%)	1,190 (38.4%)	3,099
Eclipse-Platform	8,057 (55.6%)	6,421 (44.4%)	14,478
	Fast	**Slow**	**Total**
Firefox	17,859 (69.3%)	7,915 (30.7%)	25,774
SeaMonkey	22,980 (68.1%)	10,769 (31.9%)	33,749
Bugzilla	1,875 (60.5%)	1,224 (39.5%)	3,099
Eclipse-Platform	7,647 (52.8%)	6,831 (47.2%)	14,478

I build my classifiers using a random forest classifier (Breiman, 2001). Random forest classifiers are known to be robust to data noise and often achieve a high accuracy in software engineering research (Robnik-Šikonja, 2004; Jiang et al., 2008; Lessmann et al., 2008; Kamei et al., 2010; Ghotra et al., 2015). I use the random forest algorithm provided by the *randomForest* R-package (Liaw and Wiener, 2016). In this study, I train the random forest classifier using 100 decision trees.

An important benefit of a random forest classifier is that it has a mechanism to examine the relative influence of the underlying factors in contrast to other machine learning approaches (e.g., Neural Networks). Thus, I would like to point out that although random forest classifier has been used to build accurate classifiers for prediction, my purpose of using a random forest classifier in this chapter is not for predicting the needed effort for identifying duplicate issues. My purpose is to study the factors that influence the needed effort for identifying duplicate issues. Nevertheless, I first need to determine whether the problem at hand (i.e., understanding duplicate identification effort) is non-random in nature (and hence easy to model), before I can start examining the influential factors in RQ3. To minimize the risks of over-fitting, I use

the 10-fold cross validation technique. A k-fold cross validation divides the data into k equal parts. For each fold, k-1 parts are used as training data to build a classifier and the remaining part is used as testing data.

Classification evaluation metrics. To measure the performance of my classifiers, I use precision, Recall, F-score and area under ROC curve to evaluate my classifier. I describe each measure below:

Precision(P) is the percentage of correctly classified duplicate reports per effort class (e.g., Slow). Precision ranges between 0 and 1. A correct prediction is counted if the classified effort class of a duplicate report matches its actual class. The number of correctly labeled reports is called *true positives (TP)*. The number of incorrectly labeled duplicate reports is called *false positives (FP)*. Precision equals to $(TP)/((TP)+(FP))$.

Recall(R) is the percentage of correctly classified duplicate reports for an effort class over all the reports belong to that class. Recall ranges between 0 and 1. Recall is equal to one for a certain effort class if all the duplicate reports belonging to that class are addressed. The number of duplicate reports that were not labeled to a certain effort class even though they should be labeled to it is called *false negatives (FN)*. Recall equals to $(TP)/((TP)+(FN))$.

F-score is the harmonic mean that considers both precision and Recall to calculate an accuracy score. F-score equals to $(2*P*R)/(P+R)$. Where P is the precision and R is the Recall. F-score reaches the best value at 1 and worst at zero.

ROC area value measures the discriminative ability of a classifier in selecting instances of a certain class from a dataset. The ROC metric ranges between 0 and 1. ROC curve shows the relation between the true positive rate (also called *Sensitivity*) and false positive rate (also called *1-Specificity*). The closer the ROC value to 1, the less trade-off

exists. The calculation of the ROC area is based on the probabilities of a test dataset for each class. A classifier that randomly guesses the effort class without any training has an ROC area of 0.5. I use the *pROC* R package (Xavier Robin et al., 2016) to calculate the ROC area.

Results. **The classifiers for *Identification Discussions* achieve an average precision between** 0.60 **to** 0.72 **and an average Recall between** 0.23 **to** 0.93**.** Table 4.7 shows the precision, Recall, F-score and ROC area for each class per project. Note that the highest precision for Bugzilla and Firefox are for the *Not-Discussed* class, whereas the *Discussed* class has the highest precision in SeaMonkey. The Recall value is highest in SeaMonkey for the class *Not-Discussed*, whereas in Eclipse-platform the highest precision and Recall are for the class *Discussed*. A possible reason for the high Recall for the Eclipse-platform is that the dataset is more balanced (*see* Table 4.6 which shows for example that the number of *Not-Discussed* duplicates is twice as many as the number of *Discussed* duplicates in the Firefox dataset). The F-score for the *Non-Discussed* class ranges between 0.69 to 0.78 where as for the *Discussed* class it ranges from 0.34 to 0.57. The ROC areas are between 0.68 to 0.72. These results indicate that the Identification Discussions classifier is better than random guessing (ROC of 0.5) and indicate that my factors can be used to model and understand the discussion effort that is needed for identifying duplicate reports.

The classifier for *Identification Delay* achieves an average precision between 0.63 **to** 0.77 **and average Recall between** 0.23 **to** 0.96**.** From Table 4.7, the highest precision and Recall for the *Fast* class are for Firefox and SeaMonkey, while both projects have low Recall values (0.37 for Firefox and 0.23 for SeaMonkey) for the *Slow* class. The F-score for the *Fast* class ranges between 0.70 to 0.83, whereas for the *Slow* class it ranges

Table 4.7: Evaluation results for all the studied projects.

Project	Class	Precision	Recall	F-Score	ROC area
Firefox	Not-Discussed	0.71	0.88	0.78	0.68
	Discussed	0.60	0.34	0.43	
SeaMonkey	Not-Discussed	0.62	0.93	0.74	0.69
	Discussed	0.69	0.23	0.34	
Bugzilla	Not-Discussed	0.72	0.84	0.78	0.72
	Discussed	0.65	0.47	0.55	
Eclipse-Platform	Not-Discussed	0.66	0.72	0.69	0.69
	Discussed	0.61	0.54	0.57	
	Class	**Precision**	**Recall**	**F-Score**	**ROC area**
Firefox	Fast	0.76	0.90	0.83	0.74
	Slow	0.63	0.37	0.46	
SeaMonkey	Fast	0.73	0.96	0.83	0.77
	Slow	0.74	0.23	0.35	
Bugzilla	Fast	0.77	0.86	0.81	0.80
	Slow	0.74	0.60	0.66	
Eclipse-Platform	Fast	0.68	0.69	0.69	0.74
	Slow	0.65	0.64	0.65	

from 0.35 to 0.66. My *Identification Delay* classifier has ROC areas between 0.74 to 0.80 which are better than random guessing (ROC of 0.5). These results provide a sound starting point for modeling the effort that is needed for identifying duplicate reports, and for examining the important (i.e., influential) factors in the models.

> Both of my classifiers outperform random guessing, achieving an ROC of 0.68-0.80.

RQ3: What are the most influential factors on the effort that is needed for identifying duplicate issue reports?

Motivation. I wish to understand the factors that most influence the effort that needed for identifying duplicate reports. By knowing the influential factors, I can shed some light on whether current state of the art automated approaches are identifying duplicate issue reports that require little effort.

Approach. To find the factors that most influence the effort that is needed for identifying duplicate reports, I compute the *variable importance* for each of the factors that are used in the RandomForest classifiers that are built in RQ2. In order to calculate the *variable importance*, I use the same *randomForest* R package. A Random Forest (Breiman, 2001) classifier consists of a number of decision trees. Each tree is constructed using a different sample from the original data. Around a third of the data is left out from the construction sample per tree. At the end of each iteration the left out data is used to test the classification and estimate what is called the *out-of-bag* (oob) error (Mitchell, 2011). Random Forest follows a fitting process where the target is to minimize the out-of-bag error. To measure the importance of each factor after building the random forest, the values of each factor are permuted one at a time within the training set and the out-of-bag error is calculated again. The importance of a factor is computed by calculating the average difference in out-of-bag error. I compute the factor importance values based on the change in the out-of-bag error. I take the average of the 10-folds classifiers for all the *variable importance* values for each factor. The factor with the highest rank is identified to be the most influential factor for modeling the identification effort.

To compute a statistically stable ranking of the factors, I use the Scott-Knott test (Scott

and Knott, 1974; Tantithamthavorn et al., 2015) (with $p < 0.05$). The Scott-Knott test is a statistical multi-comparison cluster analysis technique. The Scott-Knott test clusters the factors based on the reported importance values across the ten folds. The Scott-Knott test divides the factors into two different clusters, if the difference between the two clusters is statistically significant. The Scott-Knott test runs recursively until no more clusters can be created.

Results. **Factors from the Identifier Experience and Peers dimensions have the largest influence on modeling the needed effort for identifying duplicates.** Table 4.8 shows the results from Scott-Knott test for the four projects. The test groups the 18 factors into 11 significant groups on average. The clusters show that the factors from the *Identifier Experience* and *Peers* dimensions are in the top groups of influencing factors for the needed effort for identifying duplicate reports across all the studied projects. For Bugzilla, I find that the Priority factor has a large influence in both effort classifiers.

Identifier Experience dimension has the most influential factors in the identification discussions classifier. The results show that the factor *Total Duplicates identified* in the *Identifier Experience* dimension has the most influential effect. Peers dimension factors such as the *Description Similarity*, *Reporting Recency of Peers/Recency of Peer identification*, and *Peers CCList* play a high influential role in determining the effort that is needed for identifying duplicates in Firefox, SeaMonkey and Eclipse-platform. The role of identifier experience in identifying duplicates highlights the importance of dedicating experienced developers for triaging new issue reports.

Peers dimension has the most influential factors in the identification delay classifier. The results show that the factors in the Peers dimension are the most influential. Factors such as the *Reporting Recency of Peers/Recency of Peer identification*, *Peers*

CCList and *Description Similarity*, are most influential in determining the identification delay of duplicates in Firefox, SeaMonkey and Eclipse-platform. These results show that a new duplicate issue may get identified faster if it has a recent peer (i.e., fresh peer). In Bugzilla, the *Total Duplicates Identified* and *Priority* factors are still the most influential factors in the identification discussions classifier.

Discussion.

On the impact of the Priority of an Issue. My results show that the *Priority* of a report is an important factor only for the Bugzilla project in contrast to the other projects. By looking at the data, I found that 98.6% of the duplicate reports for Firefox have an empty priority value. The high percentages of empty values imply that the *Priority* field is rarely used in Firefox. Even though SeaMonkey and Bugzilla have fewer empty priority values (83.7% and 85%), SeaMonkey's priority values are barely used, since around 88% of the non-empty priority values at SeaMonkey are set to *P*3 (the default priority). On the other hand, only 54% of the non-empty priority values in Bugzilla are *P*3. The Eclipse-Platform dataset does not support empty priority values, but around 91% of the filled priority values are set to *P*3. These data explain the unique role of the *Priority* factor in the Bugzilla classifiers.

Figure 4.4 shows the quadrants for the Identification Delay versus Identification Discussion in the Firefox project. As we can observe in the figure there are 14.3% of duplicate reports with fast Identification Delay and long Identification Discussion. Similar results hold for the other projects. Possible reason for such a high percentage (i.e., 14.3%) of duplicates is the importance of the software issues described within the duplicate reports. However, based on 10 duplicate reports randomly selected from the Firefox project, there could be two other reasons for the long Identification Discussion.

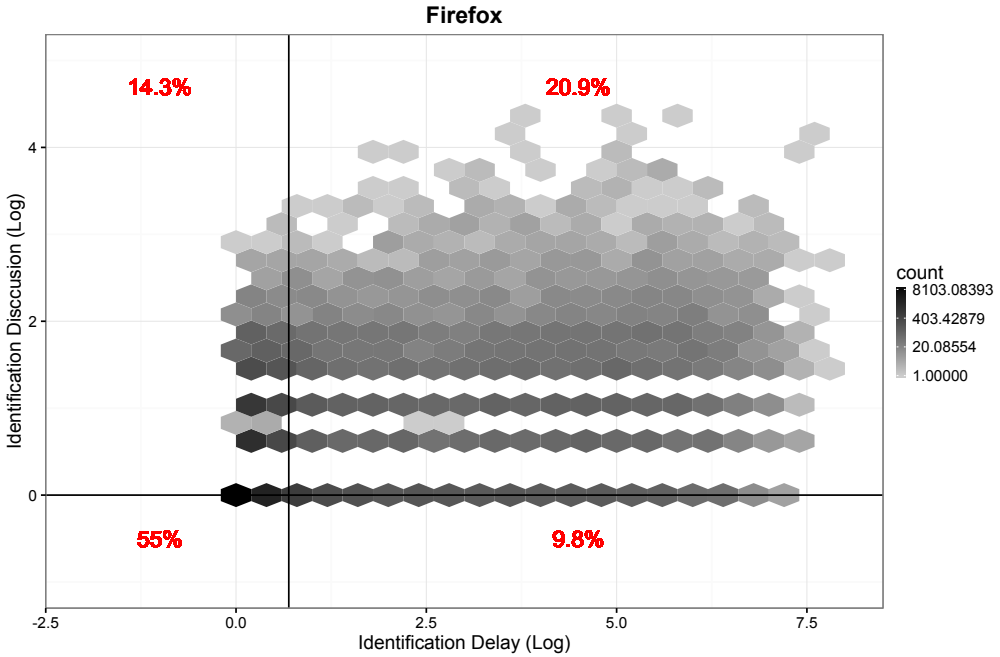


Figure 4.4: Quadrants for the Identification Delay versus Identification Discussion in the Firefox project.

First is that the developers can reach towards a fast consensus on identifying these duplicates. The second reason could be that the developers with higher authority can override the discussions and decide on identifying the duplicate reports fast ()i.e., less than one day).

> Factors from the Identifier Experience and Peers dimensions, such as Total Duplicates Identified, Recency of Peer Identification, Reporting Recency of Peers, Description Similarity and Peers CC List, are the most influential factors on the needed effort for identifying duplicate issue reports. Priority plays an important role in modeling the effort of identifying duplicate reports in Bugzilla.

4.6 Discussion

4.6.1 On the Textual Similarity of Duplicate Issue Reports

Description Similarity is one of the most important factors to influence the effort that is needed for identifying a duplicate report. The majority of the automated approaches for duplicate reports retrieval (Anvik et al., 2005; Runeson et al., 2007; Jalbert and Weimer, 2008; Nagwani and Singh, 2009; Sureka and Jalote, 2010; Prifti et al., 2011; Sun et al., 2011; Alipour et al., 2013; Lazar et al., 2014) are based on textual similarity (e.g., description similarity). My results imply that the Description Similarity is one of the most important factors to influence the needed effort for identifying a duplicate report. However, my results imply that the more similar the description, the easier developers can identify the duplicate issue report. Thus, the duplicate reports retrieved by automated approaches may be the ones that require the least effort to identify. Figure 4.5 shows a Beanplot (Kampstra et al., 2008) of the description similarity distributions for the 10% hardest and 10% easiest duplicate reports for the identification delay metric. The similarity score ranges between 0 (very dissimilar) to 1 (very similar). The closer the score to 1 the more similar the issue reports. Figure 4.5 shows that the hardest 10% duplicate reports have a lower text similarity median with a peak at the lowest similarity (i.e., no textual similarity), while the easiest 10% have a peak at the highest similarity (i.e., identical text). The same observation holds for identification discussions.

The results in Figure 4.5 show that duplicates that require more effort to identify tend to be less similar. Such a finding implies that current state of the art automated approaches are likely to miss effort-consuming duplicates.

Table 4.8: Scott-Knott test results when comparing the importance per factor for the four studied projects. Factors are divided into groups that have a statistically significant difference in the importance mean ($p < 0.05$).

Identification Discussions Classifier: Scott-Knott test results

Firefox			SeaMonkey			Bugzilla			Eclipse-Platform		
Group	**Factor**	**Mean**	**Group**	**Factor**	**Mean**	**Group**	**Factor**	**Mean**	**Group**	**Factor**	**Mean**
1	Total Duplicates Identified	0.0163	**1**	Total Duplicates Identified	0.0115	**1**	Total Duplicates Identified	0.0181	**1**	Reporting Recency of Peers	0.0152
2	Description Similarity	0.0112	**2**	Description Similarity	0.0101	**2**	Priority	0.0159	**2**	Description Similarity	0.0133
	Reporting Recency of Peers		**3**	Reporting Recency of Peers	0.0095	**3**	isDependable	0.0114	**3**	Component	0.0125
3	Recency of Peer Identification	0.0100	**4**	Recency of Peer Identification	0.0081		Title Similarity			Title Similarity	
4	Peers CC List	0.0091		Title Similarity		**4**	Description Similarity	0.0083		Total Duplicates Identified	
5	Severity	0.0080	**5**	Severity	0.0073	**5**	Recency of Peer Identification	0.0064	**4**	Priority	0.0074
6	Title Similarity	0.0066		Peers CC List			isBlocking			Waiting Load	
7	Component	0.0056	**6**	Component	0.0056		Reporting Recency of Peers	0.0051	**5**	Recency of Peer Identification	0.0058
	Waiting Load		**7**	Duplicate Peers Identified	0.0041	**6**	Peers CC List			Duplicate Peers Identified	
8	isBlocking	0.0046	**8**	isDependable	0.0031	**7**	Component	0.0037		Description Size	
9	isDependable	0.0038		isBlocking			Waiting Load		**6**	Peers CC List	0.0028
	Recency of Closed Issue		**9**	Waiting Load	0.0024		Description Size			Severity	
10	Duplicate Peers Identified	0.0028		Priority		**8**	Recency of Closed Issue	0.0020	**7**	Recency of Closed Issue	0.0021
11	Description Size	0.0018		Description Size			Severity			isBlocking	
12	Priority		**10**	Recency of Closed Issue	0.0014		Duplicate Peers Identified			isDependable	
12	Title Size	0.0008	**11**	Reproduce Steps	0.0003		Title Size		**8**	Title Size	0.0012
	Reproduce Steps			Title Size		**9**	isWeekend	0.0014		isWeekend	
13	isWeekend	0.0002	**12**	isWeekend	0.0001		Reproduce Steps		**9**	Reproduce Steps	0.0000

Identification Delay Classifier: Scott-Knott test results

Firefox			SeaMonkey			Bugzilla			Eclipse-Platform		
Group	**Factor**	**Mean**	**Group**	**Factor**	**Mean**	**Group**	**Factor**	**Mean**	**Group**	**Factor**	**Mean**
1	Reporting Recency of Peers	0.0316	**1**	Reporting Recency of Peers	0.0309	**1**	Total Duplicates Identified	0.0406	**1**	Reporting Recency of Peers	0.0309
2	Recency of Peer Identification	0.0274	**2**	Total Duplicates Identified	0.0247	**2**	Priority	0.0312	**2**	Recency of Peer Identification	0.0203
3	Total Duplicates Identified	0.0245	**3**	Recency of Peer Identification	0.0236	**3**	Recency of Peer Identification	0.0160	**3**	Component	0.0137
4	Peers CC List	0.0119	**4**	Peers CC List	0.0118	**4**	Reporting Recency of Peers	0.0141	**4**	Total Duplicates Identified	0.0128
5	Description Similarity			Description Similarity		**5**	Description Similarity	0.0105	**5**	Description Similarity	0.0106
	Waiting Load			isDependable			Component		**6**	Waiting Load	0.0080
6	Component			Duplicate Peers Identified		**6**	Peers CC List	0.0088	**7**	Recency of Closed Issue	0.0063
	Title Similarity	0.0079	**5**	Waiting Load	0.0059		Title Similarity			Severity	
	Duplicate Peers Identified			Title Similarity			isDependable			Peers CC List	
7	isBlocking	0.0055	**6**	Component	0.0047	**7**	isBlocking	0.0066	**8**	Priority	0.0054
	isDependable	0.0035	**7**	isBlocking	0.0032		Waiting Load			Title Similarity	
8	Recency of Closed Issue	0.0033	**8**	Recency of Closed Issue	0.0028	**8**	Recency of Closed Issue	0.0035		Duplicate Peers Identified	
	Severity			Severity			Description Size		**9**	Title Size	0.0029
9	Description Size			Description Size		**9**	Severity	0.0021		isWeekend	
	Priority	0.0019	**9**	Priority	0.0022		Duplicate Peers Identified		**10**	Description Size	0.0023
10	Title Size	0.0009	**10**	Title Size	0.0013		Title Size		**11**	isBlocking	0.0017
	Reproduce Steps			isWeekend		**10**	isWeekend	0.0006	**12**	isDependable	0.0007
11	isWeekend	0.0001	**11**	Reproduce Steps	0.0003		Reproduce Steps		**13**	Reproduce Steps	0.0001

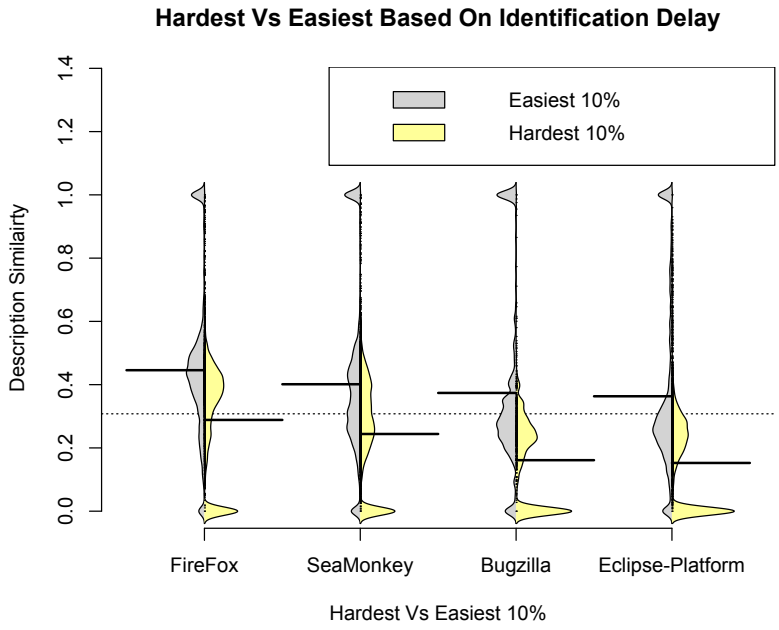


Figure 4.5: Description Similarity Beanplot for the Hardest and Easiest duplicate reports in all the studied projects.

4.6.2 On the Difficulty of Duplicate Issue Reports

I manually examined the top 20 reports for both effort metrics (i.e, Identification Delay and Identification Discussions) across the four studied projects (160 issues in total). I observed that these reports correspond to difficult issues. In particular, many of the reports correspond to issues that take a long time to resolve and around 50% of the corresponding issues are blocker issues. Figure 4.6 shows the difference between blocker and non-blocker duplicate reports based on Identification Discussions. To define the blocker issues, I use the *Blocks* field that is provided by Bugzilla. Blocker duplicate reports have a larger Identification Discussions median in all the studied projects ($p < 0.001$ for Student T-test).

To understand the relationship between duplicate report Identification Delay and

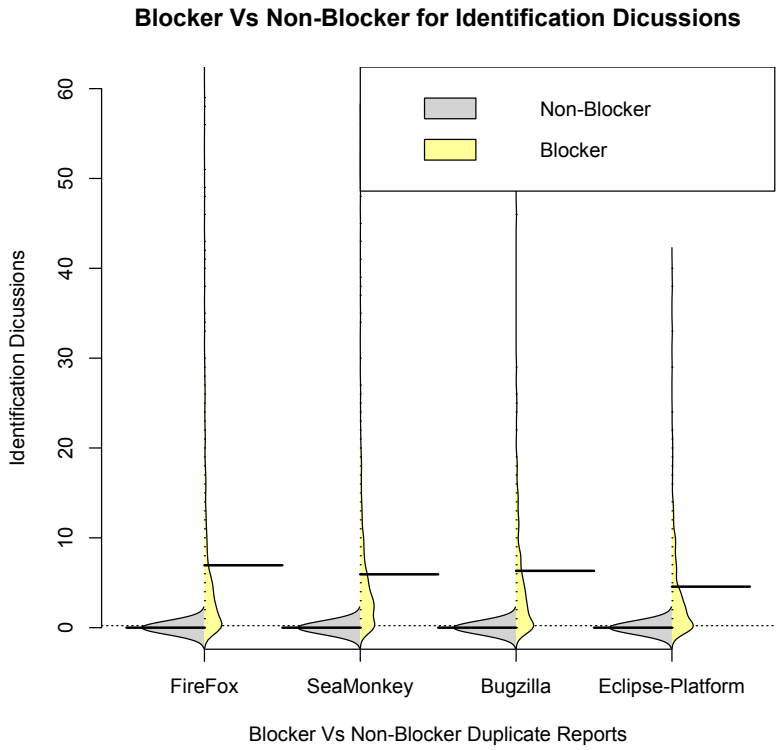
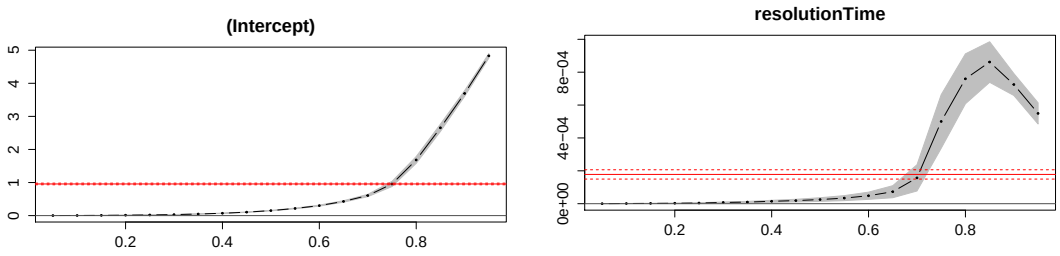


Figure 4.6: The Identification Discussions Beanplot of Blocker vs Non-Blocker duplicate reports.

the complexity of an issue (i.e., overall resolution time of an issue), I built a Quantile regression model (Angrist and Pischke, 2008). The model examines the relation between a group of predictor variables and specific percentiles of the response variable. Quantile regression allows us to understand how some percentiles of the response variable are affected differently by different percentiles of the predictor variables. Such regression is appropriate given my desire to understand the impact of the full range of my metrics. In my case, the response variable is the *Identification Delay* of a report, and my predictor variable is the overall *Resolution Time* of the issue. Figure 4.7 shows the Quantile regression results for the Firefox project. The red horizontal lines



(a) Intercept (b) Coefficient Change

Figure 4.7: Quantile regression Intercept and Coefficient Change for the Resolution Time when modeling the Identification Delay in Firefox.

are the Linear Regression model (Neter et al., 1996) coefficients with a red dotted confidence interval. The black curves are the Quantile regression coefficients with a gray confidence interval. Figure 4.7 shows that the resolution time has higher coefficients at higher quantiles, which shows that the complexity of an issue plays a role in the Identification Delay of duplicate reports. The same pattern holds for the other studied projects.

4.6.3 On the Generality of my Observations

To explore the generality of my observations, I built two global models instead of building project specific models as done in Section 4. I built one model for Identification Delay and another model for Identification Discussion. To build a global model, I put data from all the studied projects into one dataset.

The goal of the global model is to uncover the generalization of my observations. However, due to the varying sizes of the studied projects, I took random samples of equal size from each project. The size of the random sample from each project is equal to the size of the smallest dataset (3,099 issues for Bugzilla). Such an approach to create my dataset for the global model ensures that no data from one specific project can over

Table 4.9: Evaluation results for the global models

Global Model	Class	Precision	Recall	F-Score	ROC area
Identification Discussion	Not-Discussed	0.67	0.80	0.73	0.69
	Discussed	0.60	0.43	0.50	
Identification Delay	Fast	0.73	0.85	0.79	0.75
	Slow	0.67	0.49	0.57	

take the global trends across the studied projects. The accuracies of the two global models are close to the project-specific models (*see* Table 4.9).

Table 4.10 presents the results of the Scott-Knott test for the studied factors of the global models. Similar to the project-specific models, the *Total Duplicates Identified* and *Description Similarity* factors rank as the most impacting factors. However, the *Priority* and *Reporting Recency of Peers* factors that are important in the individual models of Bugzilla and Eclipse-Platform (see Section 4) do not rank as one of the top important factors in the global model. If I only built a global model, I would not observe the importance of the *Priority* and *Reporting Recency of Peers*. Such results highlight the value of building project specific models in order to uncover the different ways for handling duplicate issue reports among the studied projects. My global model highlights the general trends across the projects.

4.7 Threats to Validity

In this section, I discuss the threats to the validity of my observations.

External validity. The study in this chapter is conducted on four large open source projects. While three of the projects (i.e., Firefox, SeaMonkey, Bugzilla) are incubated by the Mozilla Foundation, these projects share only 9% of the top developers (the ones

Table 4.10: Scott-Knott test results when comparing the importance per factor for the global models. Factors are divided into groups that have a statistically significant difference in the importance mean ($p < 0.05$).

Identification Discussion Model			**Identification Delay Model**		
Group	**Factors**	**Mean**	**Group**	**Factors**	**Mean**
1	Total Duplicates Identified	0.0142	1	Reporting Recency of Peers	0.0301
2	Description Similarity	0.0134	2	Recency of Peer Identification	0.0267
	Priority		3	Total Duplicates Identified	0.0253
	Reporting Recency of Peers		4	Priority	0.0118
3	Recency of Peer Identification	0.0092		Peers CC List	
	Title Similarity		5	Description Similarity	0.0112
4	Peers CC List	0.0066		Waiting Load	
	isDependable		6	Title Similarity	0.0061
5	Severity	0.0048		Fixed Issues per Identifier	
	isBlocking		7	isDependable	0.0053
	Waiting Load			isBlocking	
	Fixed Issues per Identifier		8	Duplicate Peers Identified	0.0040
6	Description Size	0.0031		Description Size	
	Duplicate Peers Identified		9	Severity	0.0023
	Reproduce Steps		10	Reproduce Steps	0.0016
7	Title Size	0.0003		Title Size	
	isWeekend		11	isWeekend	0.0006

who retrieve over 80% of duplicate reports in total). My earlier findings (*see* Section 4) highlight as well that each one of these projects follows a different process for managing their issue reports. As usual, additional case studies are always desirable. However, for the purpose of this chapter, I do believe that my raised observations even in a single project are sufficient to raise concerns about future research for duplicate retrieval.

In particular, the purpose of my study is not to establish a general truth about all duplicate reports across all projects of the world. Instead, the goal of my study is to raise awareness that the amount of the needed effort spent on identifying duplicates varies considerably and that (at least for my studied projects) the duplicate identification for a large proportion of reports appears to be a trivial task.

In this study, I examined projects that used the Bugzilla issue tracking system. There

exist several other issue tracking systems. Each issue tracking system may introduce additional confounding factors that would impact my observations. To avoid the bias from such confounding factors, I choose to use the same issue tracking system. Replicating my study using projects that use other issue tracking system may complement my results. Future studies might wish to explore whether my observations would generalize to other open source and commercial software projects. Nevertheless, my observations highlight the need for future duplicate retrieval research to factor in the identification effort when proposing new approaches and when designing evaluation measures, otherwise evaluations are likely not to reflect the true value of newly proposed approaches and the practical impact of such approaches will remain unexplored.

Internal validity. The internal threat considers drawing conclusions from the used factors. My study of modeling the needed effort for identifying duplicate issue reports cannot claim causal relations, as I am investigating correlations, rather than conducting impact studies. The important factors for the needed effort for identifying duplicate issue reports do not indicate that the factors lead to low/high efforts. Instead, my study indicates the possibility of a relation that should be studied in depth through user studies.

Another possible way to measure discussion effort is to consider the amount of time that is associated with each comment instead of the number of comments. However, such estimation is not a straightforward one. Hence, I opted for a simple measure of effort. Future studies should explore other measures of effort instead of my current basic measure of comment counting.

I choose one day as a threshold to determine rapidly Identified reports, since I want

to ensure globally distributed developers all have a chance to see the issue report. Using more granular time periods (e.g., minutes) to measure the identification delay may introduce more noise into the data because there might be cases that a developer is too busy with other activities to immediately look into a newly filed issue report.

I used the time to mark a duplicate report as the identification delay. However, this time is just an upper bound approximation for the time that the developer spent on each duplicate report. Hence, the identification delay might be trivial for an even larger number of issues than what I report, or maybe the short identification delay is just because the developer sense about the importance of some issue reports. To achieve a better estimation of the needed time for identifying duplicate reports, I consider the time of issue triaging as the start time for looking at each duplicate report. The different processes of triaging may impact my measurement of identification effort. However, I address this threat by selecting the latest triage date just before identifying a duplicate (i.e., some issues are triaged multiple times). I measure the identification effort by counting the number of discussion comments on an issue report. However, some discussions may be from non-developers. Unfortunately, in my study I do not have the data to determine whether a commenter is a developer or not.

Construct validity. I measure effort using three metrics: *Identification Delay*, *Identification Discussions* and *Involved People*. However, these metrics may not fully measure the effort that is needed for identifying duplicate reports. In addition, there might be other ways to measure the needed efforts for identifying duplicate report. I plan to explore more metrics to measure such effort in future case studies.

4.8 Chapter Summary

A large portion of the issue reports are duplicates. Various automated approaches are proposed for retrieving such duplicate reports in order to save developers' efforts. However, there exists no research that studies the effort that is actually needed for identifying duplicate issue reports. In this chapter, I examine such effort using issue reports from four open source projects, i.e., Firefox, SeaMonkey, Bugzilla, and Eclipse-platform. I find that:

1. More than half of the duplicate reports are identified within one day, with no discussion and without the involvement of any people (other than the reporter); nevertheless there exists a small number of reports that are identified as duplicates after a long time, with many discussions and with the involvement of as many as 80 developers.

2. The developer experience and the knowledge about duplicate peers of an issue report, such as description similarity, play an important role in the effort that is needed for identifying duplicates.

This chapter shows that in many cases, developers require minimal effort to retrieve that an issue report is a duplicate of an existing one; while there exists some cases where such identification is a difficult and effort-consuming task. However, nowadays automated approaches for retrieving duplicate reports treat all issues the same.

Moreover, the results of this chapter show that the strong reliance on textual similarity by state of the art duplicate retrieval approaches will likely lead to the retrieval of duplicates that developers are already able to retrieve with minimal effort. While, the duplicate reports that need minimal effort are still worthy to be handled by current

automated retrieval approaches; my results highlight the need for duplicate retrieval approaches to put additional emphasis on the issue reports that are more difficult to identify as a duplicate.

CHAPTER 5

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

A large portion of prior research approaches were evaluated using data that spans a relatively short period of time (i.e., the classical evaluation). However, in this chapter, I show that the classical evaluation tends to overestimate the performance of automated approaches for retrieving duplicate issue reports. Instead, I propose a realistic evaluation using all the reports that are available in the ITS of a software project. I conduct experiments in which I evaluate the automated retrieval of duplicate reports using the classical and realistic evaluations. I find that the realistic evaluation shows that previously proposed approaches perform considerably lower than previously reported using the classical evaluation. As a result, I conclude that the reported performance of approaches for retrieving duplicate issue reports is significantly overestimated in literature. In order to improve the performance of the automated retrieval of duplicate issue reports, I propose to leverage the resolution field of issue reports. My experiments show that a relative improvement in the performance of a median of 7-21.5% and a maximum of 19-60% can be achieved by leveraging the resolution field of issue reports for the automated retrieval of duplicates.

An earlier version of this chapter is published in the IEEE Transactions on Software Engineering Journal (TSE) (Rakha et al., 2017).

5.1 Introduction

Many of the prior research approaches for retrieving duplicates are evaluated on a small subset of the reports that are available in an ITS (Sun et al., 2010, 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Lazar et al., 2014; Hindle et al., 2015; Zou et al., 2016). For example, Sun et al. (2010) select reports from a time frame and indicate that they unlabel an issue report as duplicate, if the report that it duplicates was not reported in the same time frame. In this type of evaluation, which I call the *classical evaluation* throughout this chapter, the assumption is that duplicately-reported issues are reported shortly after the original report. However, as I show in this chapter, this assumption is incorrect in many cases.

Instead, I propose a *realistic evaluation* for the automated retrieval of duplicate issue reports that uses all issue reports that are available in the ITS of a project. In this chapter, I conduct an experiment using data from the ITSs of the Mozilla Foundation, the Eclipse Foundation and OpenOffice in which I show that the classical evaluation relatively overestimates the performance of BM25F (Robertson et al., 2004) and REP (Sun et al., 2011), two popular approaches for retrieving duplicate issue reports, by a median of 17-42%. My results show that the proposed realistic evaluation should be used in future studies to report the performance of automated approaches for retrieving duplicate issue reports.

In the second part of this chapter, I evaluate two approaches for improving the performance of the BM25F and REP approaches (see Chapter 2). First, I evaluate the impact of limiting the set of issue reports that is searched by the approaches using a time-based threshold (e.g., search only in the last 6 months). However, such a limitation does not lead to a significant improvement in performance.

Second, I propose an approach that extends BM25F and REP with the use of the resolution field of issue reports. By using this field, I can filter issue reports in the results of BM25F and REP that are more likely to be duplicate candidates considering their resolution field value. For example, an issue report that was fixed a long time ago, is unlikely to be a duplicate of an issue report that has been reported recently. My results show that using the resolution field during the retrieval of duplicate issue reports relatively improves the performance of BM25F and REP by a median of 7-21.5% and a maximum of 19-60%. I use the Recall rate and Mean Average Precision (MAP) as performance metrics (Sun et al., 2010; Nguyen et al., 2012; Hindle et al., 2015).

In particular, in this chapter I address the following research questions:

RQ1: How much do the realistic evaluation and classical evaluation differ?

> The classical evaluation assumes that duplicate reports are reported shortly after the original report (i.e., short-term duplicates). However, there are long-term duplicates that also need to be retrieved, which is taken into account by the realistic evaluation. In this RQ, I study the differences between the realistic and classical evaluations over the lifetime of each studied ITS. I find that the classical evaluation significantly overestimates the performance of BM25F and REP for the three studied ITSs. The median observed

relative difference in the performance metrics is 17-42%.

RQ2: How does limiting the issue reports that are searched by time impact the performance of automated approach according to the realistic evaluation?

In RQ1, I observe that it is not possible to simply ignore all long-term duplicates without overestimating the performance of automated approaches for the retrieval of duplicate issue reports, as is done by the classical evaluation. In RQ2, I study whether it is possible to use a time-based threshold to ignore some of the long-term duplicates without impacting the performance. I find that it is not possible to limit the issue reports that are searched by the latest n-months without significantly impacting the performance of automated approaches for retrieving duplicate issue reports. However, I show that it is possible to limit the issue reports that are searched using a time-based threshold without a noticeable difference in performance in practice.

RQ3: How does leveraging the resolution field of an issue impact the performance according to the realistic evaluation?

In RQ3, I build upon my findings in RQ1 and RQ2 by proposing an enhancement to existing approaches that filters candidate issue reports based on their resolution value (e.g., FIXED or WONTFIX), the time since their resolution and their rank in the returned list of duplicate candidates. My results show that I can leverage the resolution field of issue reports to improve the

automated retrieval of duplicates. The proposed enhancement in this chapter improves the performance of the automated retrieval of duplicates by a median of 7-21.5% and a maximum of 19-60%.

5.2 The Main Contributions of this Chapter

I this chapter, I:

- Propose a realistic evaluation for the automated retrieval of the duplicate reports in contrast to prior research.
- Study the impact of limiting the search previously reported issues using a simple time window.
- Propose a genetic algorithm-based approach that filters the previously reported issues based on their resolution value and time.

5.3 Background

5.3.1 Performance Evaluation of Automated Retrieval of Duplicate Issue Reports

Prior research (see Table 5.1) has used an evaluation process in which the data is limited to the issues that are reported in a chosen evaluated time period. During the evaluation, all issues that are reported outside the evaluated time period are ignored. I refer

Table 5.1: An overview of prior work that used data sets that were limited to a one year period to evaluate the performance of an automated approach for the retrievel of duplicate issue reports. The data sets that were limited are highlighted in bold.

Study	Used data set(s)
Aggarwal et al. (2015)	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Hindle et al. (2015)	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010, Android 2007-2012
Jalbert and Weimer (2008)	**Mozilla Feb-Oct 2005**
Lazar et al. (2014)	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Nguyen et al. (2012)	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Sun et al. (2010)	Mozilla before June 2007, **Eclipse 2008**, **OpenOffice 2008**
Sun et al. (2011)	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010, Eclipse 2001-2007
Zhou and Zhang (2012a)	**Eclipse 2008**
Zou et al. (2016)	**Eclipse 2008**

to this evaluation process that is widely used by prior research as the *classical evaluation*. Figure 5.2a illustrates how the classical evaluation process works. For each evaluated time period, the following steps are applied:

1. An evaluated time period is selected. The issue reports are divided into two parts: (1) tuning data (200 duplicates) and (2) testing data (i.e., the remaining data after removing the 200 tuning duplicates). Note that the testing data contains both duplicate and unique issue reports.

2. 200 duplicate issue reports are used to tune the automated approach (i.e., BM25F or REP).

3. The automated approach retrieves the duplicate candidates for the issue reports

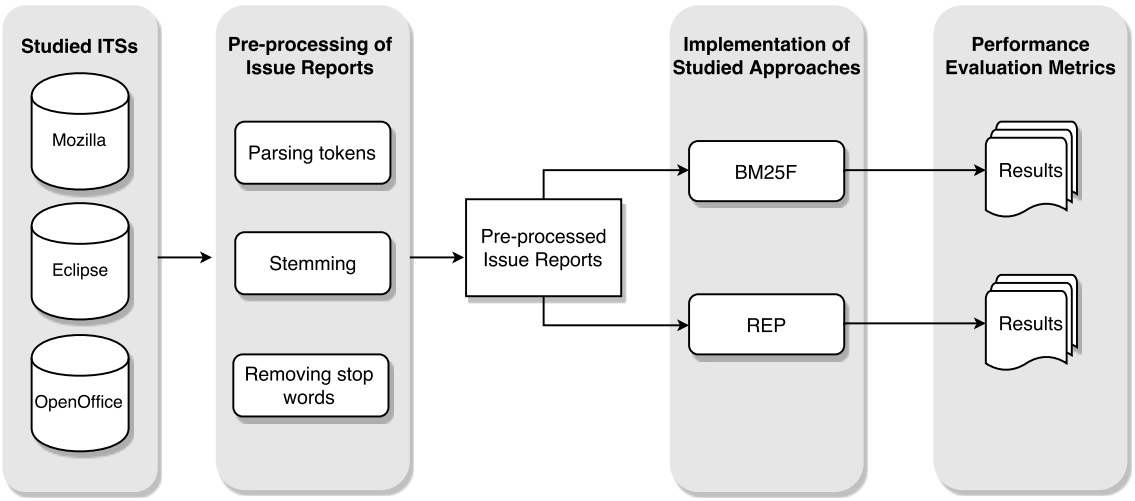


Figure 5.1: Overview of the experimental setup.

in the testing data.

4. The performance metrics of the retrieval are calculated.

The main drawback of using the classical evaluation is that the previously-reported issues that do not reside in the chosen evaluation period are ignored. The classical evaluation is valid only when duplicate issues are reported shortly after the master report. In such case, the master report of a duplicate issue report will then be within the testing period. Hence, the approaches were evaluated on their capabilities of retrieving short-term duplicate issue reports only. However, as I will show in this chapter, this assumption does not hold, leading to a relative overestimation of performance. But first, I discuss related work.

5.3.2 Related Work

Research on automatically retrieving duplicate issue reports can be roughly divided into two subdomains (see Chapter 2). The first subdomain focuses on determining

whether a newly-reported issue was already reported before. Hence, work in this subdomain focuses on retrieving whether a new issue report has a duplicate report in the ITS, or whether it describes a new issue.

The second subdomain focuses on finding (*retrieving*) existing issue reports that are the most similar to a new issue report. Hence, work in this subdomain does not give a decisive answer to whether a newly-reported issue was already reported before. Instead, a list of n candidate reports is returned containing the reports that are the most likely to be a duplicate of the newly-reported issue. The task of deciding whether the newly-reported issue is a duplicate is left to the reporter or triager (Rakha et al., 2015).

In this chapter, I focus on revisiting the performance evaluation that was used in prior work in the second subdomain (see Chapter 2).

Retrieving Duplicate Issue Reports

In this chapter, I use two approaches from the first two aforementioned groups in my experiments (see Chapter 2). I apply the BM25F (Robertson et al., 2004) approach as an example of the textual similarity-based approaches, and the REP (Sun et al., 2011) approach as an example of the textual and categorical similarity-based approaches. I use these approaches because of their popularity and reproducibility due to their available implementations, which allows us to compare my results with prior work. I do not use a topic-based approach since there exists no readily available implementation for this type of approach. Nevertheless, the used approaches in this chapter form the foundation of the topic-based approaches which increases the applicability of the findings in this chapter to all types of duplicate retrieval approaches in literature nowadays.

Table 5.2: The number of analyzed issue reports in each studied ITS for the evaluation periods used in prior studies.

ITS	# of issues	# of duplicates	Period
Mozilla	60,797	10,043	2010
Eclipse	46,000	3,815	2008
OpenOffice	31,345	4,228	2008-2010

5.4 Experimental Setup

In this section, I present the studied ITSs, the process of preparing the data that is used for my experiments and the used performance evaluation metrics. Figure 5.1 presents an overview of the experimental setup.

5.4.1 Studied ITSs

In this chapter, I choose the studied ITSs based on two criteria:

- **Size:** There should be a considerable number of issue reports (i.e., thousands of issue reports) in the ITS.
- **Usage in prior work:** The ITSs should be used in prior research on the automated retrieval of duplicates.

Therefore, I choose to study the ITSs from two popular software foundations and one popular software system, i.e., the Mozilla Foundation (hereafter referred to as Mozilla), the Eclipse Foundation (hereafter referred to as Eclipse) and OpenOffice. The Mozilla foundation hosts a variety of open source software projects such as Firefox, SeaMonkey, and Bugzilla. The Eclipse foundation is an open source community that includes a large set of toolkits which support a wide range of software development technologies.

OpenOffice is a popular Apache Software Foundation project that supports office work such as text editing and spreadsheet calculations.

In this chapter, I collect the XML file and historical information of each issue report in the three studied ITSs. I collect the XML for each report using a simple crawler that increments the id parameter of the XML URL.[1]

I focus on the same periods (see Table 6.2) as used by prior research (Sun et al., 2011; Nguyen et al., 2012; Lazar et al., 2014; Aggarwal et al., 2015; Hindle et al., 2015). Using the same data periods makes it possible to compare my results with prior research. Table 6.2 presents the number of issue reports and duplicate reports for each studied ITS.[2]

To determine the ground truth of whether an issue report is a duplicate, I use the labels that were provided by the developers that use the ITSs. Hence, all duplicate issue reports were manually labelled as being a *DUPLICATE* in the resolution field of the report. Because I recently crawled (i.e., in December 2015) relatively old data (i.e., reports that were submitted before 2011), I can safely assume that the labels are correct and stable.

5.4.2 Pre-processing of Issue Reports

Before running the automated approaches for retrieving duplicates, a set of pre-processing steps must be applied to the textual contents of every issue report (Sun et al., 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Hindle et al., 2015). These steps are:

[1]Example issue report XML: `https://bugzilla.mozilla.org/show_bug.cgi?ctype=xml&id=10000`

[2]Note that these numbers are higher than reported in prior work. In the remainder of this chapter, I will show that the numbers that are reported in prior work are incorrect. Hence, I present the correct numbers in Table 6.2.

- *Parsing tokens:* parsing the words in the textual fields of the issue reports (i.e., summary and description) that are separated by a certain text delimiter, such as a space.
- *Stemming:* finding the stem of words that have different forms such as verbs. For example, the words "argue" and "arguing" are reduced to the stem "argu".
- *Removing stop words:* removing words that do not add a significant meaning, such as "does", "be", and "the".

In this chapter, I use the following fields of an issue report: issueID, description, summary, component, priority, type, version and report date. I follow a bucket structure (Sun et al., 2011) where all the duplicate reports are placed inside the same bucket in which the earliest reported issue report is called the *master* report. I use the same text preprocessing implementation used by Sun et al. (2011) and Nguyen et al. (2012).

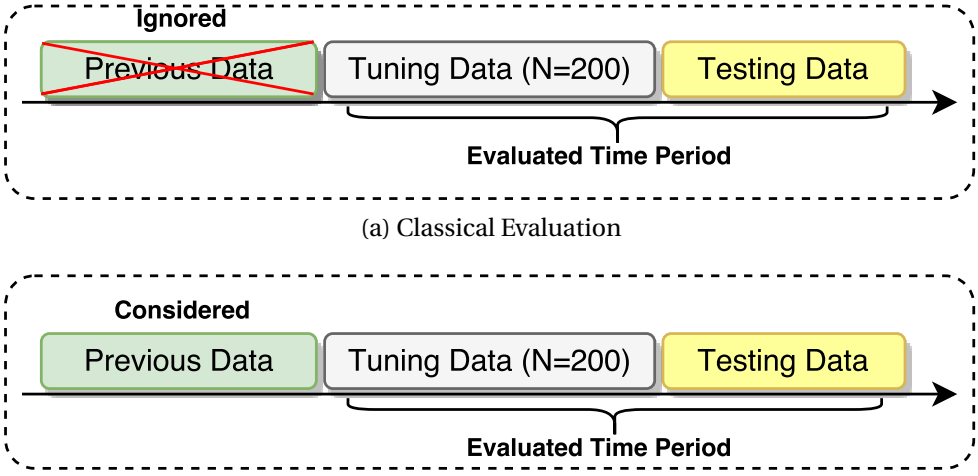
5.4.3 Implementation of the Experiments

In this chapter, I use the BM25F and REP implementations that are provided by Sun et al[3]. In addition, I made the implementation and results of my experiments available as a replication package[4].

5.5 Exploratory Study

As explained in Section 5.3.1, the classical evaluation ignores a significant portion of the issue reports in an ITS when retrieving duplicate candidates. In this section, I first

[3] http://www.comp.nus.edu.sg/~specmine/suncn/ase11/index.html
[4] http://sailhome.cs.queensu.ca/replication/realistic_vs_classical/



(a) Classical Evaluation

(b) Realistic Evaluation

Figure 5.2: Classical evaluation versus realistic evaluation.

Table 5.3: The combinations of a duplicate report and its master report that are considered by the classical and realistic evaluation. Note that all other combinations of D and M do not occur as D is not in the testing data for those combinations.

Previous data	**Tuning data**	**Testing data**	**Considered in classical evaluation?**	**Considered in realistic evaluation?**
M(aster)		D(uplicate)	No	Yes
	M	D	Yes	Yes
		M, D	Yes	Yes
		D, M	Never (D cannot be reported before M)	

propose a more realistic evaluation, which does not ignore issue reports. Second, I describe an exploratory study that is conducted to get an indication of how the classical evaluation of the automated approaches for retrieving duplicate issue reports overestimates the performance. I compare the evaluation that is conducted in prior work (i.e., the classical evaluation) with a realistic evaluation using the same issue reports.

5.5.1 Realistic Evaluation

I propose to use an evaluation in which no data is ignored. I refer to this type of evaluation as the *realistic evaluation*. I chose this name because this type of evaluation closely resembles the way in which an ITS works in practice. Figure 5.2b illustrates how the realistic evaluation works compared to the classical evaluation. The only difference between the realistic evaluation and the classical evaluation is that the classical evaluation ignores the issue reports that are reported before the evaluated time period, while the realistic evaluation does not. Hence, the following steps are taken during the realistic evaluation:

1. Use exactly the same tuning data as used during the classical evaluation.
2. Consider both tuning and previous data when searching for master candidates for the reports in the testing data.
3. Calculate the performance metrics for the automated retrieval on the testing data.

Table 5.3 shows for each combination of duplicate D in the testing data and its master report M whether the combination is considered by the classical and realistic evaluation.

In the remainder of this section, I present my exploratory study on the differences between the classical and realistic evaluation.

5.5.2 Classical vs. Realistic Evaluation Processes

I compare the classical and realistic evaluation processes for the time periods that are evaluated by prior work (Sun et al., 2010, 2011; Nguyen et al., 2012; Aggarwal et al., 2015;

Table 5.4: The number of considered duplicate issue reports for both types of evaluation for the studied ITSs in the exploratory study.

	# of duplicates (classical)	# of duplicates (realistic)	Ignored by classical (%)
Mozilla (2010)	7,551	10,043	-23.5%
Eclipse (2008)	2,918	3,815	-23.5%
OpenOffice (2008-2010)	3,299	4,228	-22.0%

Lazar et al., 2014; Hindle et al., 2015; Zou et al., 2016). I compare the number of duplicate reports that are considered by both evaluations and the difference in performance that is yielded by the two types of evaluation.

Number of considered duplicates: The number of considered duplicates is the total number of newly-reported duplicates within the testing data that have a master report in the previously-reported issues. Because the classical evaluation limits the issue reports that are searched, newly-reported issues may fail to be recognized as a duplicate because their master reports are in the ignored issue reports (Sun et al., 2010). Hence, the number of considered duplicates is likely to be lower in the classical evaluation than in the realistic evaluation, because the classical evaluation only focuses on short-term duplicates. Table 5.4 shows the difference in the number of considered duplicates for both types of evaluation. I observe that up to 23.5% of the duplicate reports are ignored when I apply the classical evaluation as compared to the realistic evaluation. These observations indicate that applying the classical evaluation ignores a considerable percentage of duplicates that the triager has to manually retrieve without benefiting from the proposed automated approaches.

Table 5.5: Performance comparison of the classical evaluation and the realistic evaluation for the studied ITSs.

	Classical	Realistic	Overestimation	
			Absolute	Relative
Mozilla (2010)				
BM25F				
Recall$_{top5}$	0.52	0.28	+0.24	+85.7%
Recall$_{top10}$	0.59	0.34	+0.25	+74.2%
MAP	0.47	0.26	+0.21	+74.5%
REP				
Recall$_{top5}$	0.58	0.43	+0.15	+39.8%
Recall$_{top10}$	0.66	0.51	+0.15	+32.9%
MAP	0.49	0.38	+0.11	+32.1%
Eclipse (2008)				
BM25F				
Recall$_{top5}$	0.60	0.49	+0.11	+22.4%
Recall$_{top10}$	0.67	0.55	+0.12	+21.8%
MAP	0.54	0.46	+0.08	+17.4%
REP				
Recall$_{top5}$	0.67	0.55	+0.12	+21.8%
Recall$_{top10}$	0.73	0.62	+0.11	+18.1%
MAP	0.57	0.48	+0.09	+17.4%
OpenOffice (2008-2010)				
BM25F				
Recall$_{top5}$	0.53	0.41	+0.12	+29.2%
Recall$_{top10}$	0.61	0.48	+0.13	+27.1%
MAP	0.45	0.36	+0.09	+22.2%
REP				
Recall$_{top5}$	0.58	0.45	+0.13	+29.4%
Recall$_{top10}$	0.66	0.53	+0.13	+26.0%
MAP	0.49	0.38	+0.11	+28.9%

The classical evaluation completely ignores up to 23.5% of the duplicates from the performance calculation for the same evaluated time periods.

Performance overestimation: To determine the difference in performance between the classical and realistic evaluation, I conduct the following steps:

1. Use the first 200 duplicate issue reports in the evaluated time period to tune the automated approaches for retrieving duplicate issue reports.

2. Use the other issue reports in the evaluated time period as testing data and run the automated approaches on each issue report in the testing data.

3. Calculate the $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP of the automated approaches using the classical and realistic evaluation.

Table 5.5 presents the absolute and relative difference in performance metrics for both types of evaluation processes for the BM25F and REP approaches. I define the relative overestimation of the classical evaluation process of a performance metric as:

$$\text{Overestimation}_{\text{relative}} = (\frac{metric_{classical}}{metric_{realistic}} - 1) * 100\% \tag{5.1}$$

where $metric_{classical}$ is the metric as yielded by the classical evaluation process, while $metric_{realistic}$ is the metric as yielded by the realistic evaluation process.

Table 5.5 shows that for the three studied ITSs, the classical evaluation relatively overestimates all observed performance metrics. For the BM25F approach, the relative overestimation is up to 85.7%, 74.2% and 74.5% for $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP, respectively. For the REP approach, the relative overestimation is up to 39.8%, 32.9% and 32.1% for $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP, respectively.

> For the BM25F approach, the classical evaluation relatively overestimates the performance metrics with up to 85.7%, 74.2% and 74.5% for the $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP. For the REP approach, the classical evaluation relatively overestimates the performance metrics with up to 39.8%, 32.9% and 32.1% for the $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP.

My two observations for the differences between the classical evaluation and realistic evaluation motivate the study that is presented in the rest of this chapter.

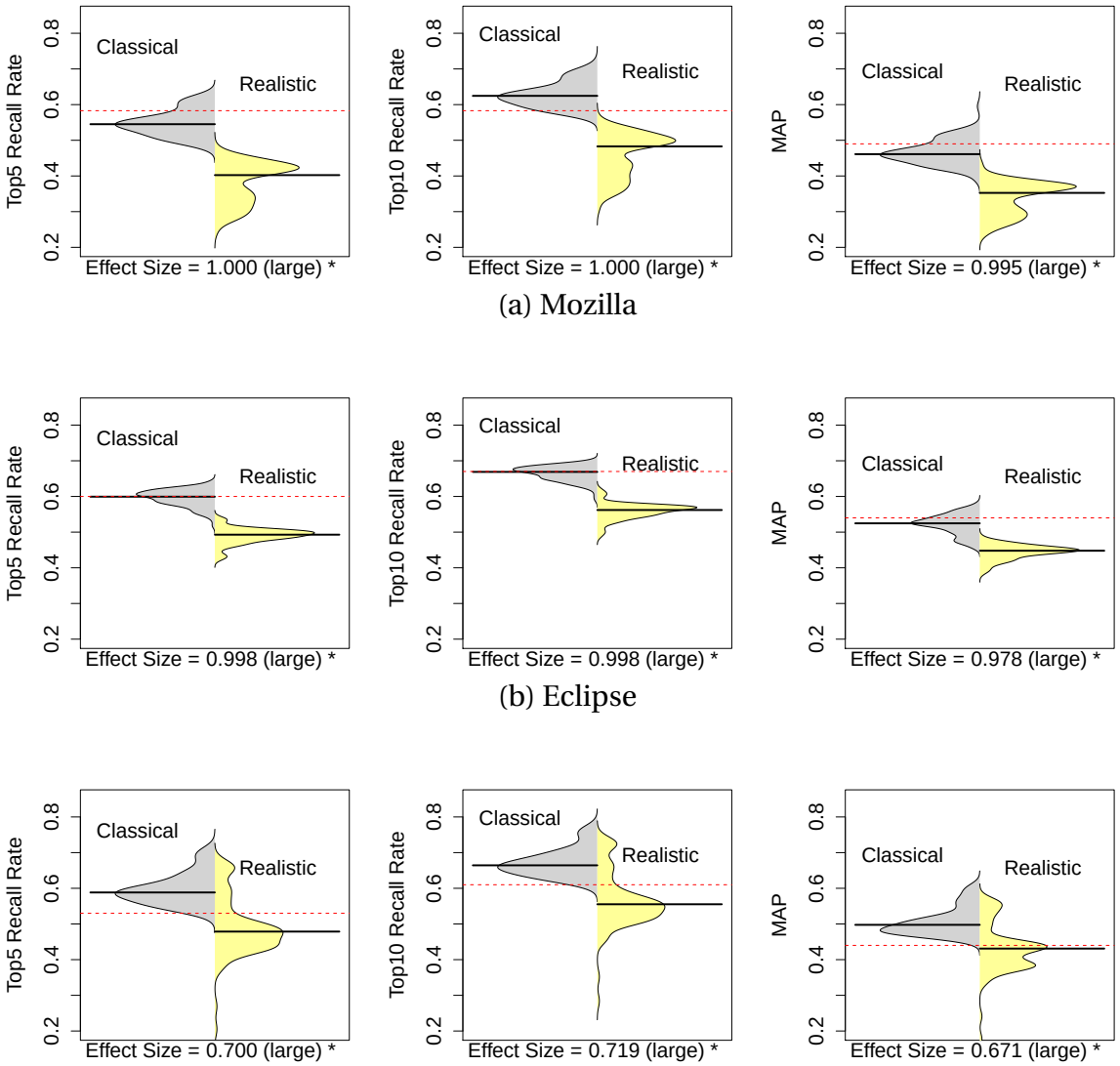
5.6 Experimental Results

In this section, I present the results of my experiments. For every research question, I discuss the motivation, approach and results.

RQ1: How much do the realistic evaluation and classical evaluation differ?

Motivation. In Section 5.5, I showed that there is a difference between the classical and realistic evaluation for the evaluated time periods that are used in prior work (Sun et al., 2010, 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Lazar et al., 2014; Hindle et al., 2015; Zou et al., 2016). However, the observations for the evaluated time periods may not generalize to data from other periods. Therefore, I study whether the observations in the exploratory study generalize to all the available periods in the studied ITSs.

Approach. For each studied ITS, I randomly select 100 chunks of data (i.e., each chunk represents the issue reports over a one year period). Each chunk of data follows the



(a) Mozilla

(b) Eclipse

(c) OpenOffice

** $p < 0.001$*

Figure 5.3: Classical evaluation vs. realistic evaluation for the BM25F approach for 100 evaluated years of data that were randomly selected over the lifetime of the studied ITSs. The dotted red line represents the performance value that is observed in the exploratory study for the classical evaluation in Section 5.5.1.

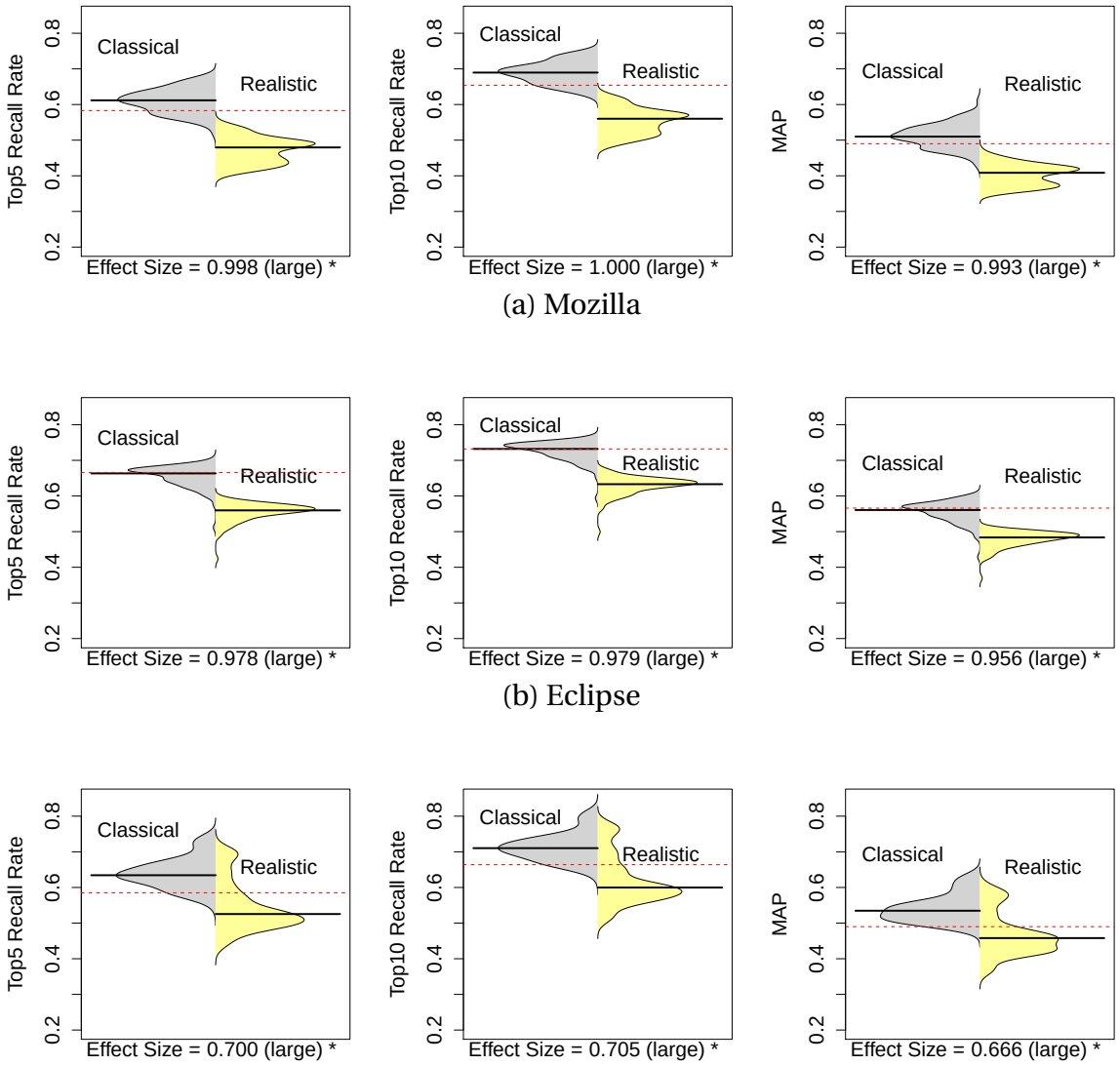
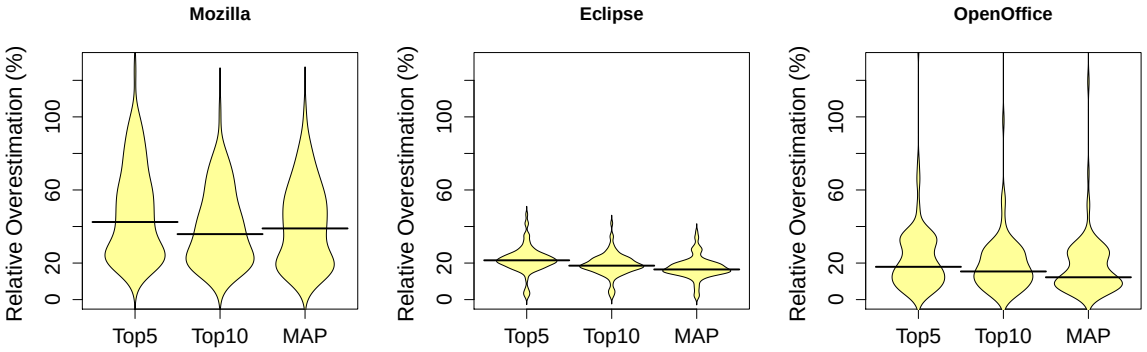
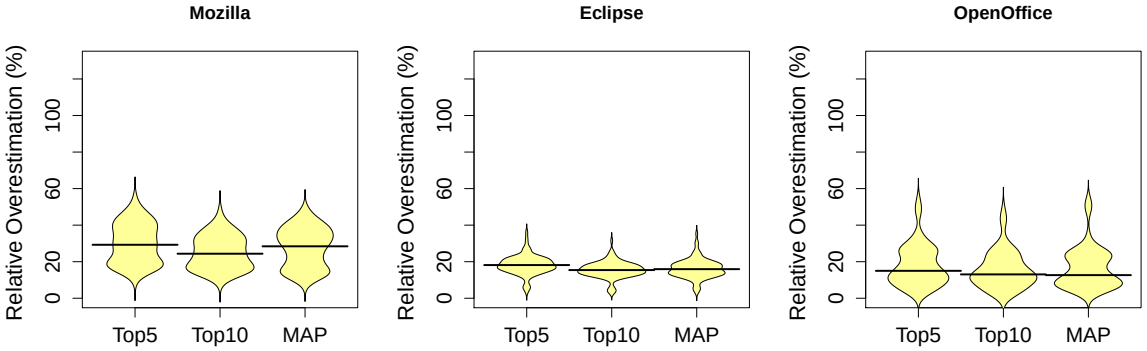


Figure 5.4: Classical evaluation vs. realistic evaluation for the REP approach for 100 evaluated years of data that were randomly selected over the lifetime of the studied ITSs. The dotted red line represents the performance value that is observed in the exploratory study for the classical evaluation in Section 5.5.1.



(a) BM25F



(b) REP

Figure 5.5: Performance relative overestimation by the classical evaluation.

same structure as presented previously in Figure 5.2. Hence, the first 200 duplicate reports of a chunk of data are used as tuning data, while all other reports in the chunks are used as testing data. I calculate the $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP performance metrics using the classical and realistic evaluation. The testing periods that are specified in Table 6.2 are only used in my exploratory study. In RQ1, the randomly-selected chunks cover the lifetime of each studied ITS. Note that the chunks of one year can start at any day of the year. Hence, the chunks January 1, 2010 to January 1, 2011 and January 2, 2010 to January 2, 2011 are considered different chunks. By randomly selecting 100 of such chunks for each ITS, I ensure that the evaluated chunks cover the whole lifetime

an ITS. The lifetimes of the studied ITSs are 2001-2010, 2002-2008, and 1998-2010 for OpenOffice, Eclipse and Mozilla, respectively. I compare the distributions of each performance metric yielded by the classical and realistic evaluation for each ITS using a paired *Mann-Whitney U* statistical test (Gehan, 1965). I use the following hypotheses:

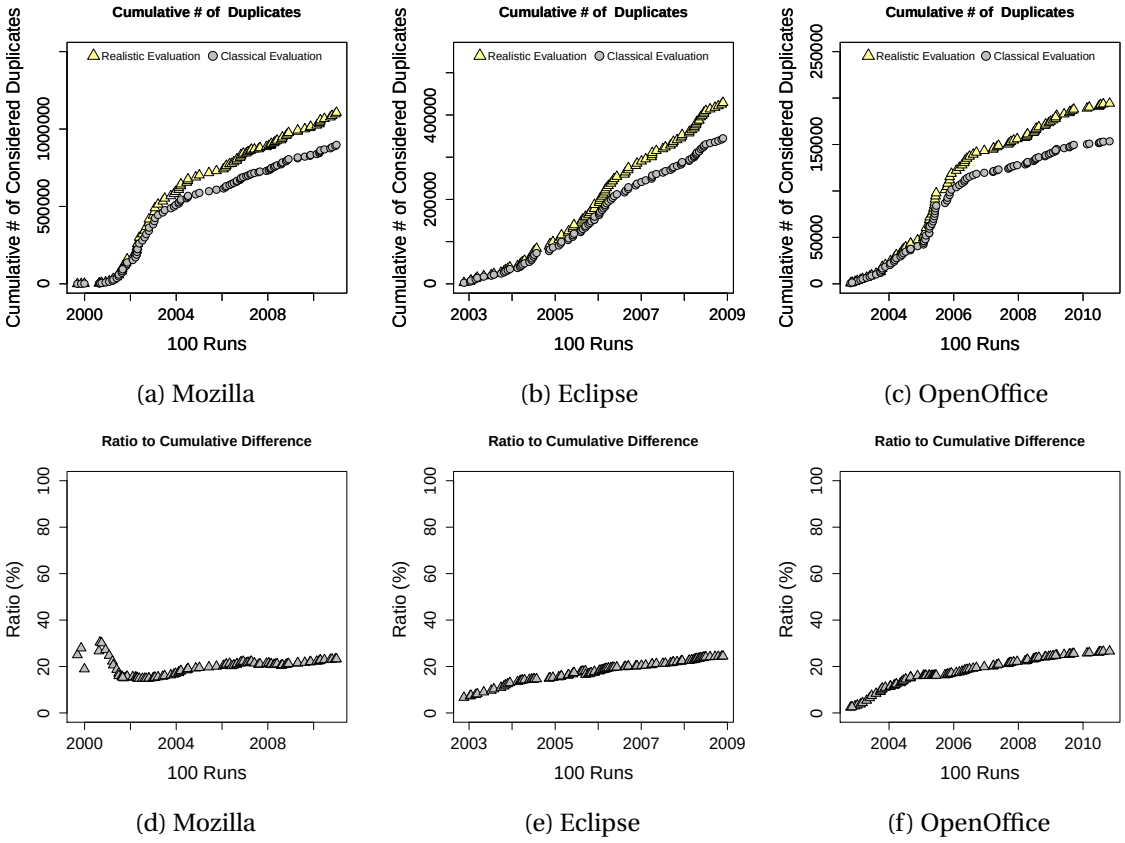
H_0: The classical and realistic evaluations yield the same performance.

H_1: The classical and realistic evaluations yield different performance.

I reject H_0 and accept H_1, when $p < 0.01$. In addition, I calculate the *effect size*, as the effect size quantifies the difference between the two distributions. I use *Cliff's Delta* as it does not require the normality assumption of a distribution to quantify the effect size (Long et al., 2003). The following thresholds are used for the *Cliff's Delta (d)* (Romano et al., 2006):

$$\begin{cases} \text{trivial} & \text{for } |d| \leq 0.147 \\ \text{small} & \text{for } 0.147 < |d| \leq 0.33 \\ \text{medium} & \text{for } 0.33 < |d| \leq 0.474 \\ \text{large} & \text{for } 0.474 < |d| \leq 1 \end{cases} \tag{5.2}$$

In addition, I calculate the relative overestimation (see Section 5.5) and kurtosis (Joanes and Gill, 1998) of the distributions. Kurtosis explains the peakedness of a distribution. The Gaussian distribution has a kurtosis of 3. A kurtosis higher than 3 means that the distribution has a higher peak than the Gaussian distribution, while a kurtosis lower than 3 means that the distribution has a lower peak. A high kurtosis means that the values in the distribution share a relatively strong agreement on the average of the distribution, while a low kurtosis means that there is no clear agreement on the average.

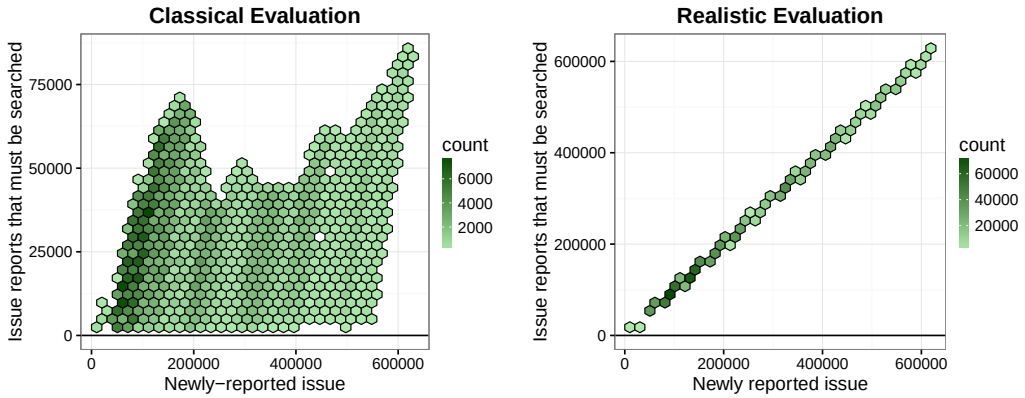


(a) Mozilla (b) Eclipse (c) OpenOffice

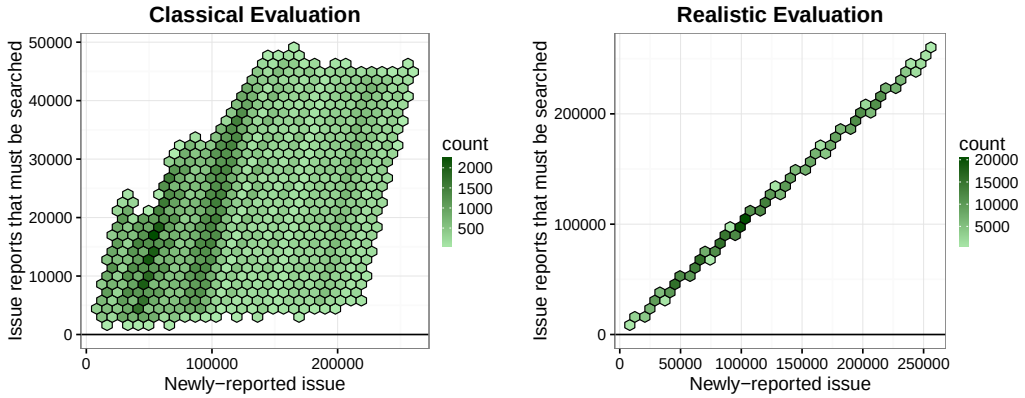
(d) Mozilla (e) Eclipse (f) OpenOffice

Figure 5.6: The number of considered duplicates issues over the evaluated time periods (100 runs).

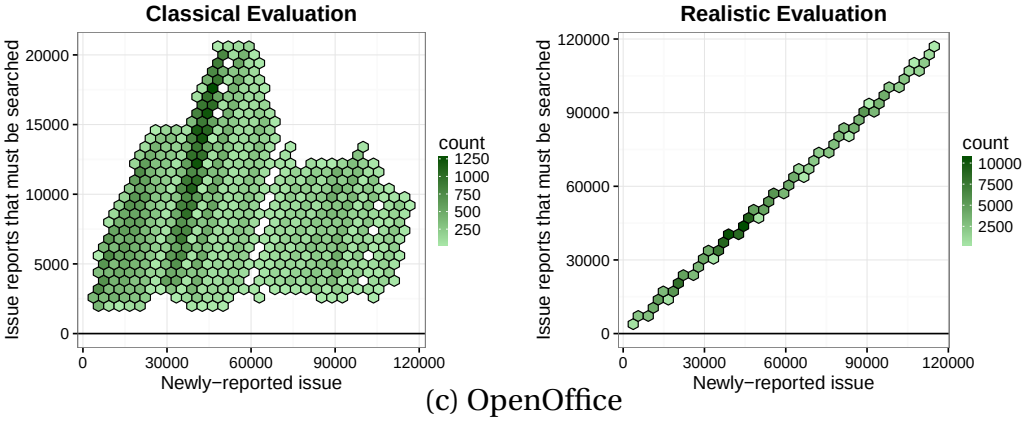
Results. **The classical evaluation relatively overestimates the performance of automated approaches for the retrieval of duplicate issue reports with a median of 17-42%.** Figures 5.3 and 5.4 show the distributions of the performance metrics that are yielded by the classical and realistic evaluation for the 100 randomly-selected chunks of data for each studied ITS. The results show that the realistic evaluation yields a significantly-lower performance than the classical evaluation. For all studied ITSs and metrics, the observed effect size is large. Figure 5.5 shows the distribution of the relative overestimation of the classical evaluation for the performance metrics for all the



(a) Mozilla

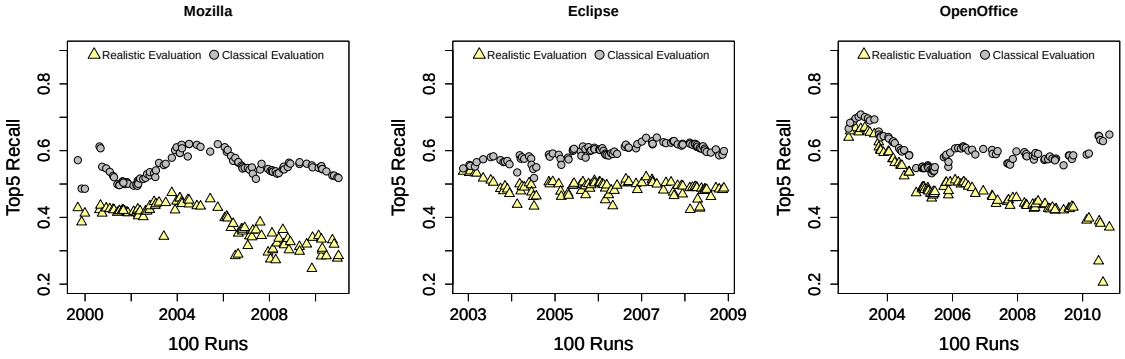


(b) Eclipse

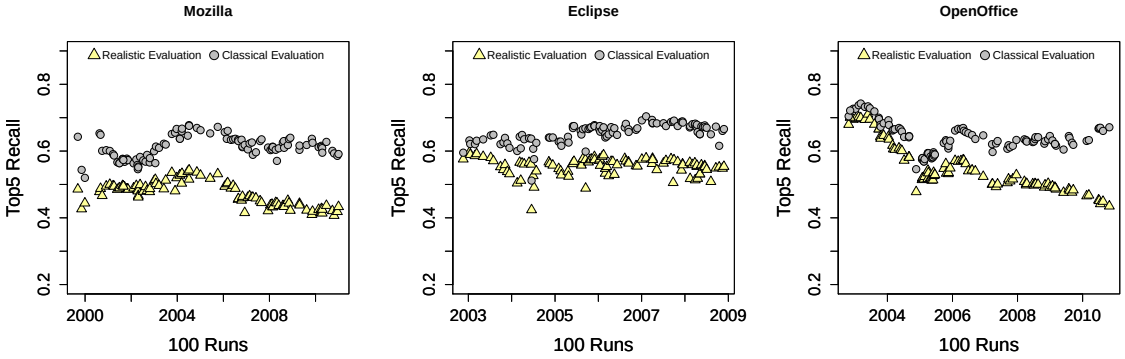


(c) OpenOffice

Figure 5.7: The number of issue reports that must be searched for each newly-reported issue.



(a) BM25F



(a) REP

Figure 5.8: The $\text{Recall}_{\text{top5}}$ yielded by the classical and realistic evaluation over time.

studied ITSs. The classical evaluation relatively overestimates the performance of automated approaches for the retrieval of duplicates by a median of 17-42%. These results support my suggestion that the classical evaluation should not be used to evaluate the performance for the automated retrieval of duplicate issue reports but instead, researchers should use my proposed realistic evaluation.

The yielded performance varies greatly for both evaluation types. Figures 5.3 and 5.4 show that the yielded performance metrics for both evaluation types show a large variation for all the studies ITSs. Table 5.6 shows the kurtosis for the observed

Table 5.6: The kurtosis comparison of the classical and realistic evaluation for the studied ITSs.

	Mozilla		**Eclipse**		**OpenOffice**	
	Classical	**Realistic**	**Classical**	**Realistic**	**Classical**	**Realistic**
BM25F						
Kurtosis ($Recall_{top5}$)	**2.40**	**1.92**	3.20	4.10	3.14	3.98
Kurtosis ($Recall_{top10}$)	**2.56**	**2.03**	3.52	4.28	3.25	4.18
Kurtosis (MAP)	3.79	**2.03**	**2.47**	3.42	3.18	4.36
REP						
Kurtosis ($Recall_{top5}$)	**2.45**	**2.01**	7.20	8.45	**2.87**	**2.84**
Kurtosis ($Recall_{top10}$)	**2.23**	**2.15**	7.64	8.10	3.10	**2.74**
Kurtosis (MAP)	**2.95**	**1.93**	5.35	7.18	**2.83**	**2.65**

The values that are highlighted in **bold** indicate a distribution that is more varied than the normal distribution (i.e., kurtosis < 3).

performance metrics. For all metrics and studied ITSs, the kurtosis is close to 3, which indicates that there is a large variation in the performance of the approach for the evaluated chunk (i.e., year). The exception is the REP approach for Eclipse. However, Figure 5.4 (b) shows that the variations is still considerably large.

Figure 5.3 and 5.4 show a dotted red line for the classical evaluation performance for the tested year periods by the prior work (see Section 5.5).

As clearly demonstrated by the figures and Table 5.6, a single value is not sufficient to accurately report about the performance of an approach for automatically retrieving duplicate issue reports. These results indicate that future studies on the automated retrieval of duplicates should report their performance as ranges of performance metric values (i.e., with confidence intervals) to give more accurate results.

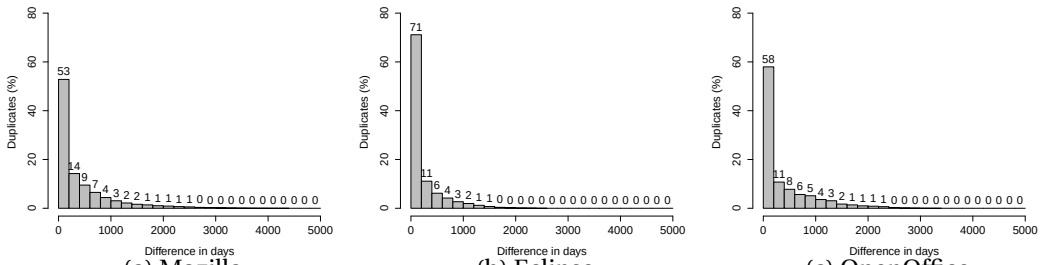
The realistic evaluation considers more duplicate issues for the same evaluated time periods. Figure 5.6 shows the number of duplicates that are considered by the classical and realistic evaluations. In the early stages of an ITS, there is not much difference between the number of duplicate issues considered by both types of evaluation.

However, as a project evolves (i.e., its ITS grows), the classical evaluation ignores up to 30%, 26%, and 27% for Mozilla, Eclipse, and OpenOffice, respectively. These results emphasize the importance of using the realistic evaluation, especially for an ITS that has been around for some time (i.e., long-lived software projects).

The performance of automated retrieval of duplicate issue reports is negatively impacted by the number of issue reports that must be searched. Figure 5.7 shows a hexagon bin plot[5] for the number of issue reports that must be searched for each newly-reported issue. A hexagon bin plot is a special type of scatter plot that avoids over-plotting of large datasets by combining many data points into hexagon-shaped bins. These bins indicate the approximate number of issue reports that must be searched to retrieve the duplicate of a specific newly-submitted issue report. The approximation here refers only to the visual clustering of the data into different levels of colors. For example, the hexagon figure for the realistic evaluation of Mozilla has four levels of colors from dark green (i.e., 60,000 to 80,000 newly reported issues) to bright green (i.e., 10,000 to 20,000 newly reported issues).

I observe that the realistic evaluation searches each previously-reported issue in the ITS to retrieve a duplicate. However, the number of issue reports that must be searched has a negative impact on the performance. Figure 5.8 shows the $\text{Recall}_{\text{top5}}$ over time for the BM25F and REP approaches. In case of the realistic evaluation, the maximum performance is achieved at the earliest years when the number of issue reports that must be searched is the smallest. For example, the $\text{Recall}_{\text{top5}}$ for the BM25F approach applied to OpenOffice drops from 0.63 to around 0.3. The drop in performance is not as clear for Eclipse, but the maximum performance is still achieved in the early year runs for the Eclipse ITS. Similar observations hold for $\text{Recall}_{\text{top10}}$ and MAP, hence, I omit

[5]Hexbin package: https://cran.r-project.org/web/packages/ggplot2/index.html

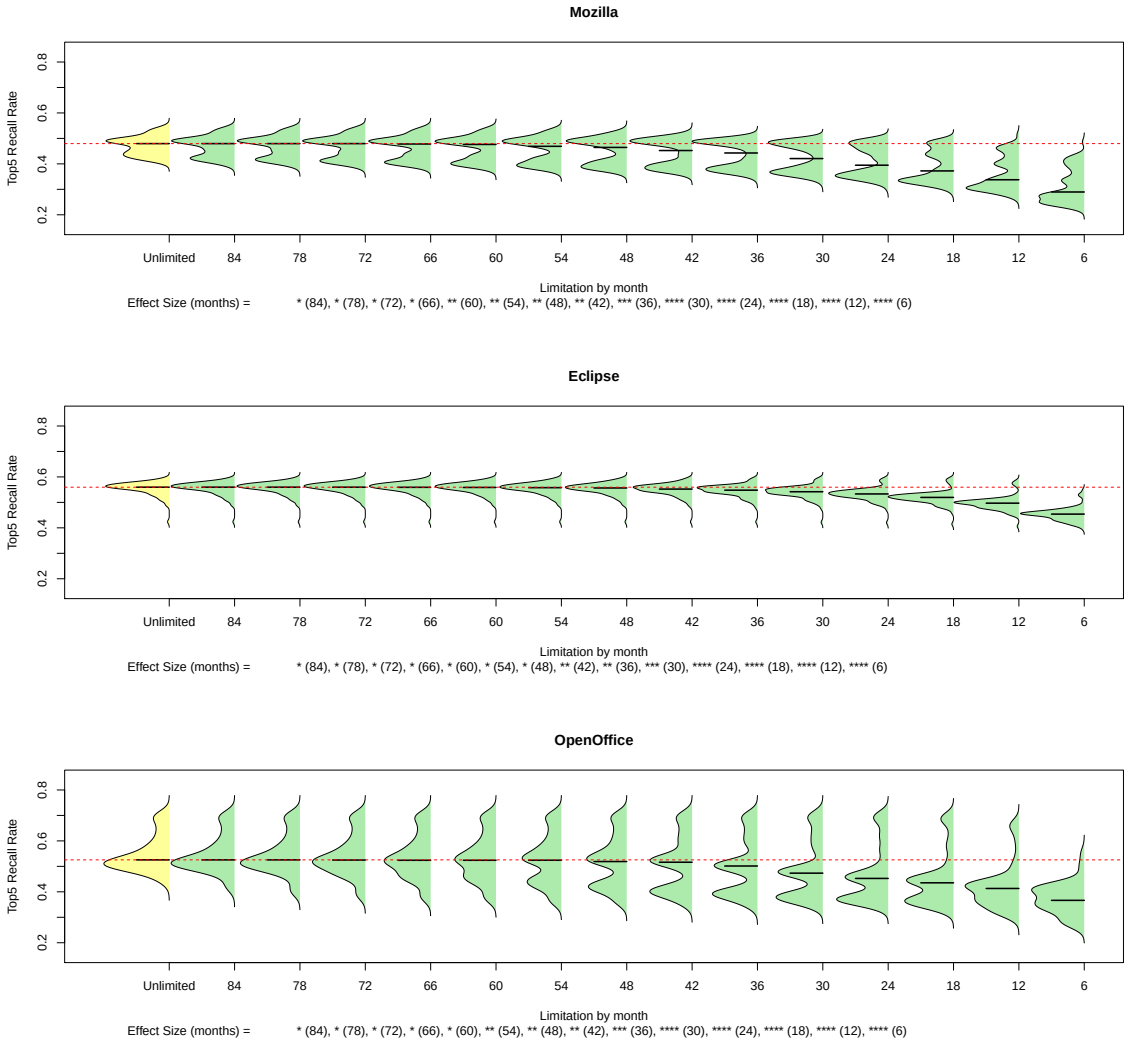


(a) Mozilla (b) Eclipse (c) OpenOffice

Figure 5.9: The REP approach: the distribution of the difference in days between the newly-reported duplicate issues and their masters in the top 5 candidates list.

those figures from this chapter. These results highlight the impact of selecting a certain time period for measuring the performance using realistic evaluation. Choosing a time period at the early stages of an ITS may lead to an unexpected high performance in contrast to later time periods. Future studies should use a wide range of time periods to evaluate an automated approach for retrieving duplicate issue reports in order to achieve more confidence in the performance metrics.

> The classical evaluation yields inaccurate results. Reporting a range of values of a performance metric as yielded by the realistic evaluation gives a more adequate representation of the performance of an automated approach for the retrieval of duplicate issue reports.

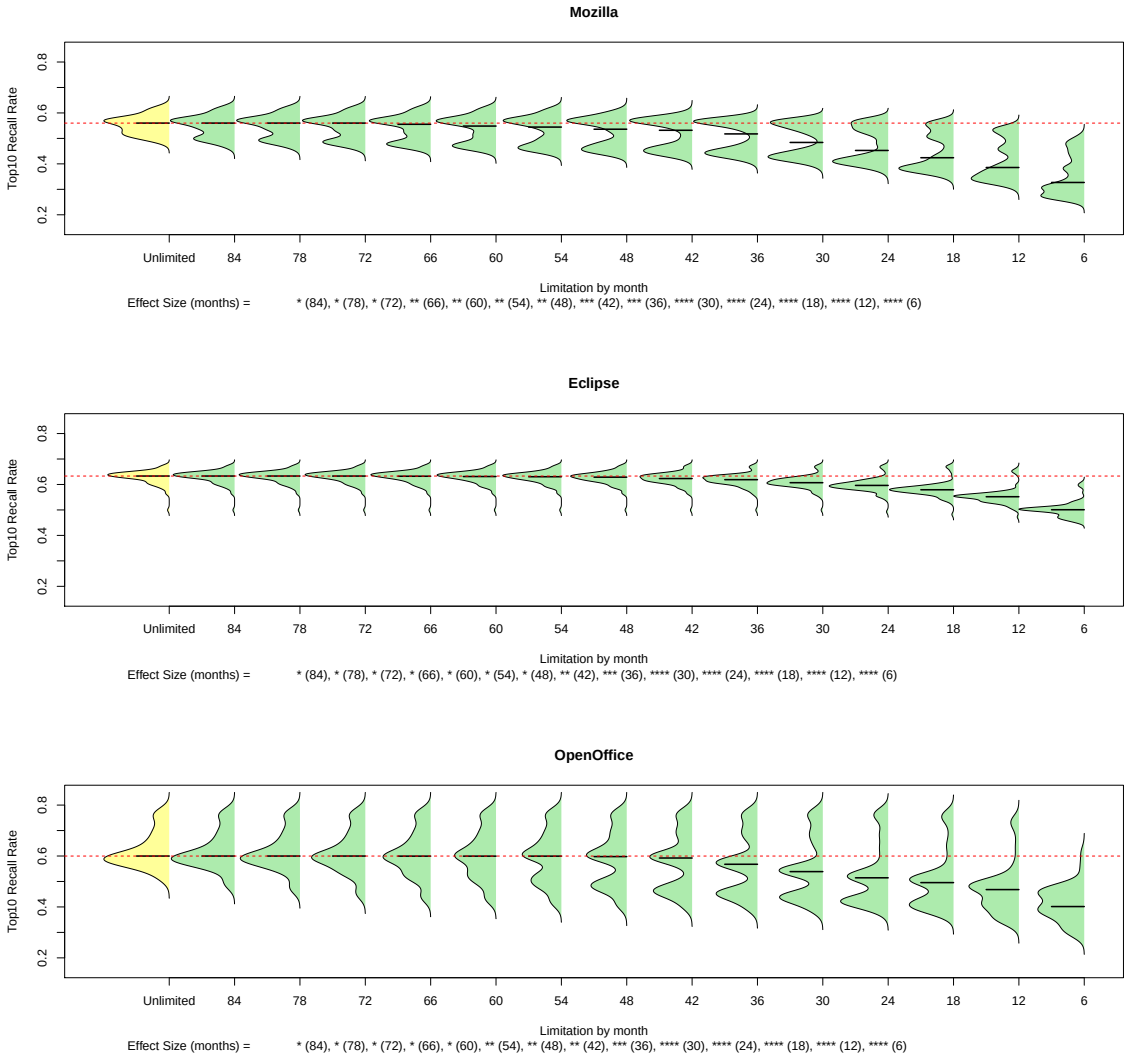


*Effect Size: * trivial, ** small, *** medium, **** large*

Figure 5.10: The $\text{Recall}_{\text{top5}}$ yielded after limiting the issue reports that are searched by *n*-months for REP. The red dotted line is the median Recall before applying any limitation.

RQ2: How does limiting the issue reports that are searched by time impact the performance of automated approach according to the realistic evaluation?

Motivation. In my exploratory study and RQ1, I found that the classical evaluation overestimates the performance of approaches for the automated retrieval of duplicate



*Effect Size: * trivial, ** small, *** medium, **** large*

Figure 5.11: The $\text{Recall}_{\text{top10}}$ yielded after limiting the issue reports that are searched by n-months for REP. The red dotted line is the median Recall before applying any limitation.

issue reports. The intuitive explanation for my finding is that the performance with the realistic evaluation drops as the ITS ages (Banerjee et al., 2016), as more long-term duplicates come into the ITS, which are ignored by the classical evaluation. As the results of RQ1 show, simply ignoring these long-term duplicates results in an overestimation

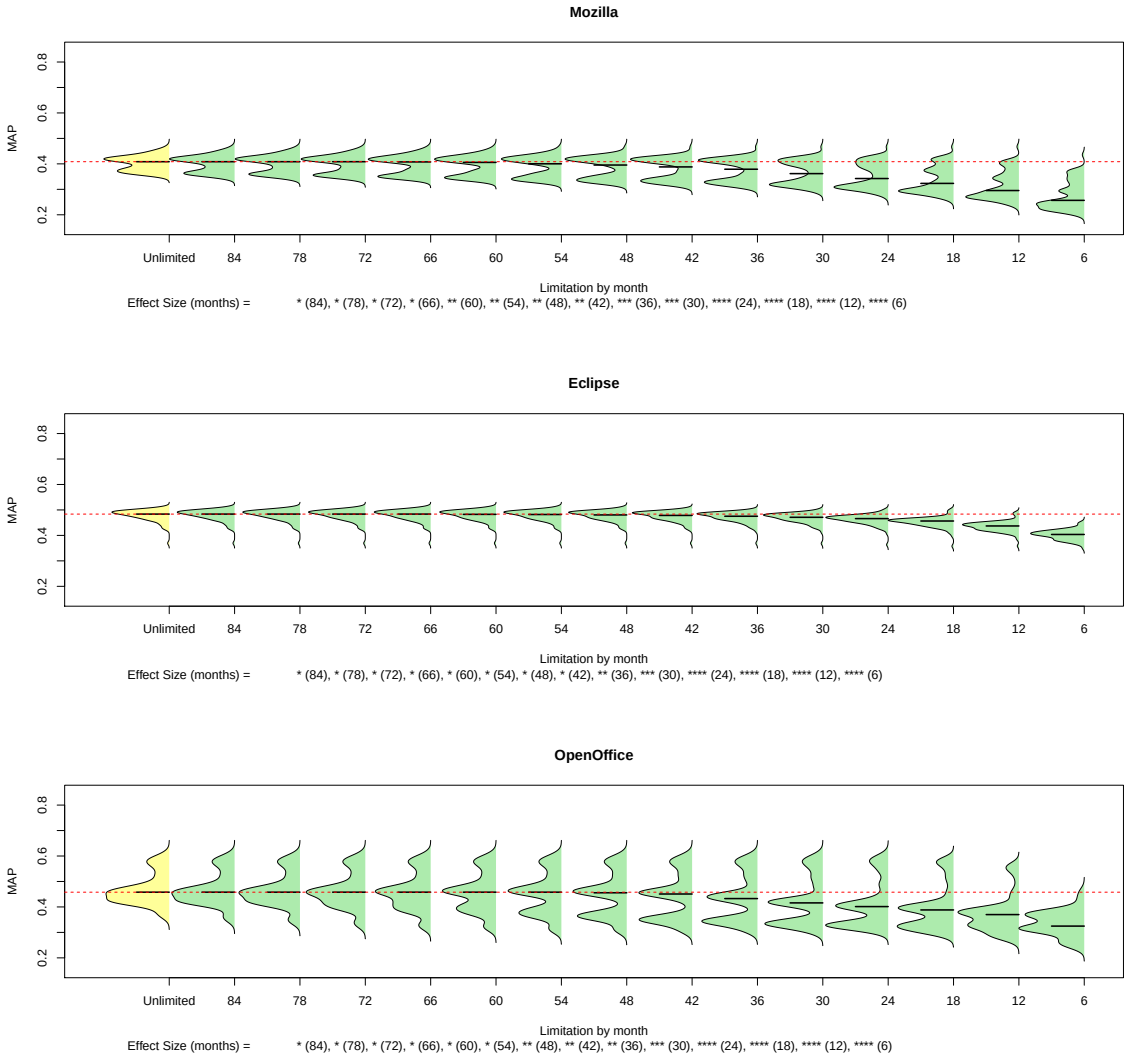
of the performance.

Figure 5.9 shows the distribution of the number of days between the reporting of a duplicate issue and its master report. As Figure 5.9 shows, the chances of a newly-reported issue being a duplicate of an old issue report are negligible. In this RQ, I investigate whether it is possible to ignore some of the long-term duplicates in the search without impacting the performance of automated approaches for the retrieval of duplicate issue reports. Ignoring issue reports in the search can be beneficial in terms of a reduced search time.

Approach. In this RQ, I reuse the same realistic evaluation runs of RQ1 (i.e., 100 randomly-selected runs over the lifetime of each studied ITS). However, I ignore some of the searched issue reports using an age-based threshold. For each newly-reported duplicate issue, I limit the searched issue reports to the ones that have a maximum age of n months. I evaluate values for n ranging from 6 months to 84 months (in steps of 6 months). Similar to RQ1, I use the Mann-Whitney U test and Cliff's Delta effect size to study the impact of limiting the issue reports that are searched.

Results. **It is not possible to limit the issue reports that are searched without decreasing the performance of automated approaches for retrieving duplicate issue reports.** Figures 5.10, 5.11 and 5.12 show the influence of limiting the issue reports that are searched on the performance of the automated retrieval of duplicates by the REP approach. These figures show the performance in the case of the realistic evaluation without limitation along with the performance distributions after each limitation by n months.

The limitation by relatively small thresholds for n (i.e., 6 to 30 months) shows a large



*Effect Size: * trivial, ** small, *** medium, **** large*

Figure 5.12: The MAP yielded after limiting the issue reports that are searched by n-months for REP. The red dotted line is the median Recall before applying any limitation.

decrease in the performance of the automated retrieval of duplicates. Larger thresholds, such as n=48 or larger, show a significant change compared to no limitation (i.e., n=∞) but with trivial effect-size. The decrease in performance shows that all long-term duplicates are important in terms of impact on the performance when it comes

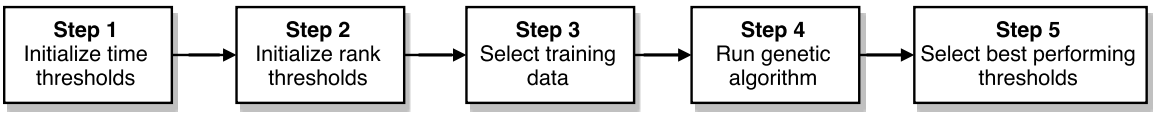


Figure 5.13: Overview of the threshold optimization approach.

to evaluating the performance of automated approaches for the retrieval of duplicate issue reports. Hence, it is not possible to limit the issue reports that are searched without significantly overestimating the performance of these approaches. However, in practice, limiting the searched issue reports to those with a maximum age of 48 months would not noticeably affect the performance (i.e., the effect size of the difference is trivial). Similar results are observed for the BM25F approach, hence I omit these figures from this chapter. These results indicate that limiting the searched issue reports by n-months could be a viable enhancement when trying to improve the performance of automated approaches for the retrieval of duplicates for long-lived software projects. However, there will always be a statistically significant effect on the observed performance. A tradeoff needs to be made between getting the most realistic view of the performance of an automated approach for retrieving duplicate issue reports, and the decreased cost of having to search a smaller set of issue reports to retrieve a list of master candidates.

> The limitation of the issue reports that are searched by n months significantly decreases the performance of the automated retrieval of duplicate issue reports while applying the realistic evaluation. However, for larger values of n, this decrease has a trivial effect size.

RQ3: How does leveraging the resolution field of an issue impact the performance according to the realistic evaluation?

Motivation. In the results of RQ2, I found that limiting the issue reports that are searched based on the time since their reporting date negatively affects the performance of approaches for the retrieval of duplicate issue reports. Instead, I propose to use a filtering mechanism based on the resolution field and the elapsed time since the resolution of an issue report when searching for duplicates. The intuition behind using the resolution field is that the resolution, combined with the time since the resolution, is related to the chances of a newly-submitted report being a duplicate. For example, if an issue report was resolved as *FIXED* several years ago, the chances of a newly-submitted report being a duplicate of that master are relatively small, because the occurrence of the fixed issue is unlikely. On the other hand, the chances of a newly-submitted report being a duplicate of an issue that will not be fixed (i.e., WONTFIX), are large regardless of the time since the resolution.

As Figure 5.9 shows, the majority of duplicates are short-term duplicates. However, long-term duplicates still need to be retrieved in some cases. My hypothesis is that by filtering issue reports that are unlikely to be the master of a new issue report, I can improve the performance of automated approaches for the retrieval of duplicate issue reports. Hence, the resolution field of an issue report may help with the retrieval of duplicate reports, as I can prioritize certain reports in the ranking. I am the first to use the resolution field of the issue reports to improve the performance of the automated retrieval of duplicates.

Approach. The approach that I propose is based on applying several filters to the ranked list of master candidates that is outputted by the automated approaches (i.e.,

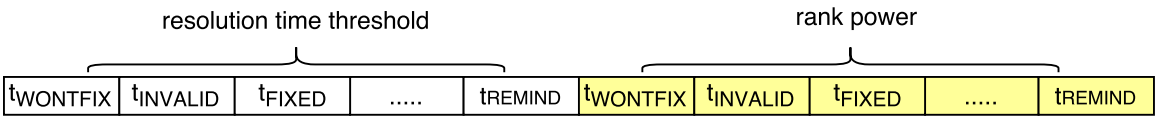


Figure 5.14: The vector of threshold variables that is optimized by the NSGA-II algorithm.

BM25F and REP). These filters are based on the resolution field (e.g., FIXED or INVALID) and the time since the resolution of the masters in the returned candidate list at the time of each newly-reported duplicate issue. The idea of my approach is that I have a time-based threshold for each resolution value and each rank in the returned list of candidates. Then, I filter each candidate from the returned list for which its resolution was much longer than the threshold for its resolution value and rank. Hence, if the threshold for FIXED reports at rank 2 is 10 days, I filter a report that was resolved as FIXED more than 10 days ago, when it is at rank 2 in the returned list.

To find the optimal thresholds for these filters, I use a genetic algorithm (Deb et al., 2002). I evaluate my proposed filtration on the same 100 chunks of data that were used in RQ1 while applying the realistic evaluation. I designed the following formula for the thresholds (in days):

$$\text{threshold}_{resolution,rank} = \frac{t_{resolution}}{(rank)^{r_{resolution}}} \tag{5.3}$$

where $t_{resolution}$ is the optimized resolution time variable and $r_{resolution}$ is the optimized rank variable for a resolution value. The *rank* is the actual location of the master report in the candidate list. The idea of using $r_{resolution}$ as the power of rank is to

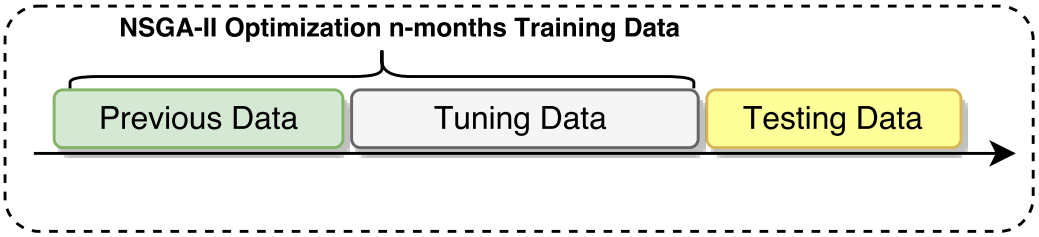


Figure 5.15: Training periods selection for thresholds optimization for each chunk of the 100.

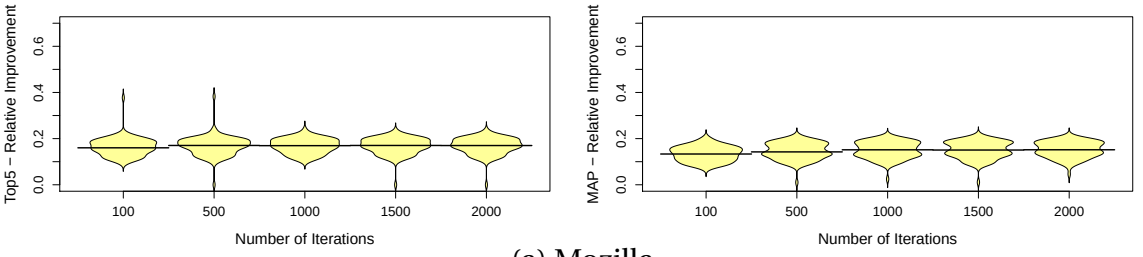
Table 5.7: An example of the optimized threshold variables for OpenOffice with $\text{Recall}_{\text{top5}}$ as objective.

	Optimized variables		**Resulting thresholds for the top 3 ranks (in days)**					
Resolution	$t_{resolution}$	$r_{resolution}$	Threshold for rank 1		Threshold for rank 2		Threshold for rank 3	
WONTFIX	3108.73	4.54	$(\frac{3108.73}{1^{4.54}}) =$	3108.73	$(\frac{3108.73}{2^{4.54}}) =$	133.63	$(\frac{3108.73}{3^{4.54}}) =$	21.21
INVALID	217.72	3.86	$(\frac{217.72}{1^{3.86}}) =$	217.72	$(\frac{217.72}{2^{3.86}}) =$	14.99	$(\frac{217.72}{3^{3.86}}) =$	3.13
WORKSFORME	51.99	4.26	$(\frac{51.99}{1^{4.26}}) =$	51.99	$(\frac{51.99}{2^{4.26}}) =$	2.71	$(\frac{51.99}{3^{4.26}}) =$	0.48
FIXED	286.19	1.79	$(\frac{286.19}{1^{1.79}}) =$	286.19	$(\frac{286.19}{2^{1.79}}) =$	82.76	$(\frac{286.19}{3^{1.79}}) =$	40.05
LATER	1675.40	3.46	$(\frac{1675.40}{1^{3.46}}) =$	1675.40	$(\frac{1675.40}{2^{3.46}}) =$	152.25	$\frac{1675.40}{3^{3.46}} =$	37.43
REMIND	1813.77	0.79	$(\frac{1813.77}{1^{0.79}}) =$	1813.77	$(\frac{1813.77}{2^{0.79}}) =$	1048.98	$\frac{1813.77}{3^{0.79}} =$	761.48
CUSTOM	2278.83	2.67	$(\frac{2278.83}{1^{2.67}}) =$	2278.83	$(\frac{2278.83}{2^{2.67}}) =$	358.07	$(\frac{2278.83}{3^{2.67}}) =$	121.30

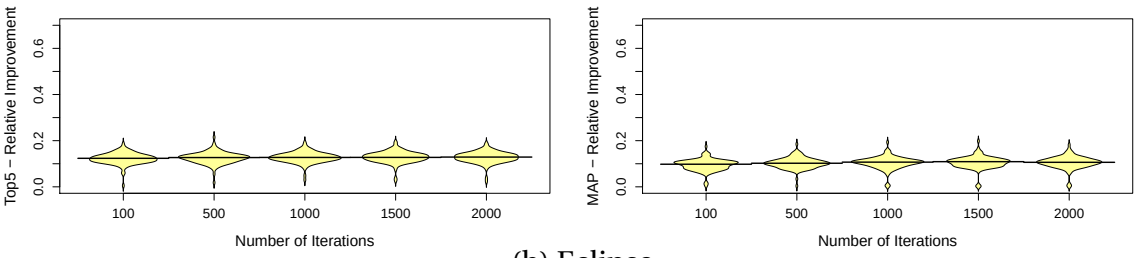
'punish' the master candidates that have a low ranking (i.e., low similarity). A master is filtered from the list of candidates when the time since its resolution ($time_{diff}$) changed is larger than the threshold for that resolution field value and rank.

Extracting the resolution field. To extract the resolution field at the time of reporting a new issue, I extract the resolution history of each issue in the ITS. I then extract the resolution field at the time of reporting the new duplicate issue. By repeating this extraction for all issues for each newly-reported duplicate issue, I get an accurate view of the resolution field values.

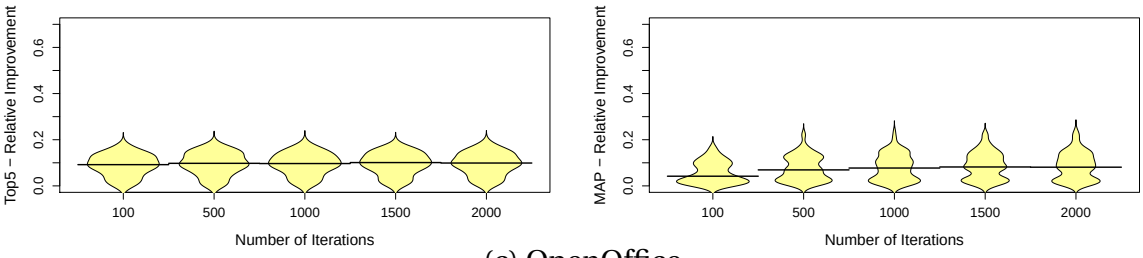
Optimizing the thresholds. In order to find the optimal thresholds that are used in



(a) Mozilla



(b) Eclipse



(c) OpenOffice

Figure 5.16: The relative improvement in performance after running the genetic algorithm for various numbers of iterations for the REP approach.

the filters, I use the NSGA-II (Deb et al., 2002) genetic algorithm to find the thresholds for each possible value for the resolution field.[6] I choose NSGA-II algorithm because it is known to often achieve a high accuracy in software engineering research (Singh et al., 2016; Karim et al., 2016). Figure 5.14 shows the vector that is optimized by the NSGA-II algorithm. The vector consists of 14 variables (i.e., a time variable and a rank variable for each of the 7 possible resolution field values).

[6]In particular, I use the MOEA framework (Hadka, 2011).

A genetic algorithm follows a process that closely resembles natural evolution to optimize a solution based on an objective. In my case, the objectives are to optimize the studied performance metrics: (1) $\text{Recall}_{\text{top5}}$, (2) $\text{Recall}_{\text{top10}}$ and (3) MAP. The idea of a genetic algorithm is to make small changes to an input vector (i.e., the vector in Figure 5.14), and evaluate how these changes affect the metrics that need to be optimized. The genetic algorithm then guides the evolution of the vector towards its optimized value considering the objective.

Figure 5.13 presents an overview of my threshold optimization process. Below I detail each step of my process.

1. Initialize the time variables (i.e., $t_{WONTFIX}$, $t_{INVALID}$, ..., t_{REMIND}) in the input vector to the largest possible value, which is the age of the ITS at the time of the evaluated period. Hence, if an ITS is 8 years old at the time of a certain chunk of data, all time variables are initialized to $8 * 365 = 2,920$ days.

2. Initialize the rank variables (i.e., $r_{WONTFIX}$, $r_{INVALID}$, ..., r_{REMIND}) to the highest rank that is considered for the metric that is being optimized. For example, if $\text{Recall}_{\text{top10}}$ is being optimized, all rank variables are initialized to 10.

3. To select training data for the genetic algorithm, I need a set of duplicate candidate lists. To collect such a set, divide the chunk of data into tuning and testing data using the same approach as in RQ1 (i.e., use the first 200 duplicate reports as tuning data and the other reports as testing data). Tune the automated approach using the tuning data, and run the tuned approach for the duplicate reports that were reported within n-months before the testing data (see Figure 5.15). Collect the list that is returned for each duplicate report, and use all collected lists as the training data for the genetic algorithm.

4. Run the genetic algorithm with the goal of optimizing the performance (i.e., Recall rate or MAP) of the training data set. In each iteration, the genetic algorithm slightly changes the input vector. The time and rank variables in the input vector are then used to adjust the thresholds, and the performance of the training data is calculated after filtering the candidate lists based on the new thresholds.

5. After 1,000 iterations, stop the genetic algorithm and select the best-performing vector of variables. Figure 5.16 shows the relative improvement in performance after running the genetic algorithm for 100, 500, 1,000, 1,500 and 2,000 iterations for the REP approach. I used 1,000 iterations in my experiments because the relative improvement in performance did not change considerably after 1,000 iterations.

I used the default settings (Hadka, 2011) of the MOEA framework for the parameters of the NSGA-II algorithm. Increasing the number of iterations to more than 1,000 did not further change the results that are presented in this RQ. Note that this process is repeated for each of the 100 chunks of data that I evaluate, as the chunks are randomly selected from the lifetime of the ITS. Hence, the thresholds need to be optimized for every run of the experiment. In addition, I run the algorithm for each studied ITS and automated approach.

An example of optimized thresholds. Table 5.7 shows an example of the optimized variables (i.e., $t_{resolution}$ and $r_{resolution}$) that are generated for one run of OpenOffice using the NSGA-II algorithm. In addition, I show the resulting thresholds for the top 3 ranks. I observe that the optimized thresholds for the WORKSFORME master reports are the most strict. Intuitively, the strict thresholds can be explained by the unlikeliness of an issue report that was resolved as WORKSFORME being reported again, as the

WORKSFORME resolution status indicates that the report probably did not describe a real issue. In addition, I observe that the thresholds for the WONTFIX reports are the highest. The explanation for the height of the thresholds is that the issue reports that are resolved as WONTFIX are likely to contain actual issues, which may occur again.

Results. **The proposed filtration approach improves the performance of the automated retrieval of duplicate issue reports by a median of 10-22%.** Figures 5.17 and 5.18 show the $\text{Recall}_{\text{top5}}$, $\text{Recall}_{\text{top10}}$ and MAP distributions pre and post-filtration for the 100 runs of the studied ITSs after optimizing the thresholds with 2 months of training data. The results show that the performance post-filtration is significantly better for all the studied ITSs. Figure 5.19 shows the distribution of the relative improvement of all studied performance metrics in all studied ITSs using BM25F or REP. The median relative improvement of the Recall rate ranges from 10-22%. Similarly, the MAP has a median relative improvement in the range of 7-18%. While these improvements may look small, common relative improvements of prior work in this research area are in the range of a few percent (Sun et al., 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Hindle et al., 2015; Zou et al., 2016). Hence, my relative improvement can be considered large. The maximum improvement is achieved when the ITS ages with a relative improvement of up to 60%. These results highlight that the resolution field of issue reports can help to improve the automated retrieval of duplicate issue reports. The main drawback of using the resolution field is that it does not help in the early stages of the lifetime of an approach since there are only a small number of resolved issues, making it difficult to find the optimized thresholds. However, in the early stages there are less issue reports to be searched and hence it is much easier to retrieve duplicate reports. Therefore, using the resolution field in an ITS with relatively few issue reports

is unnecessary.

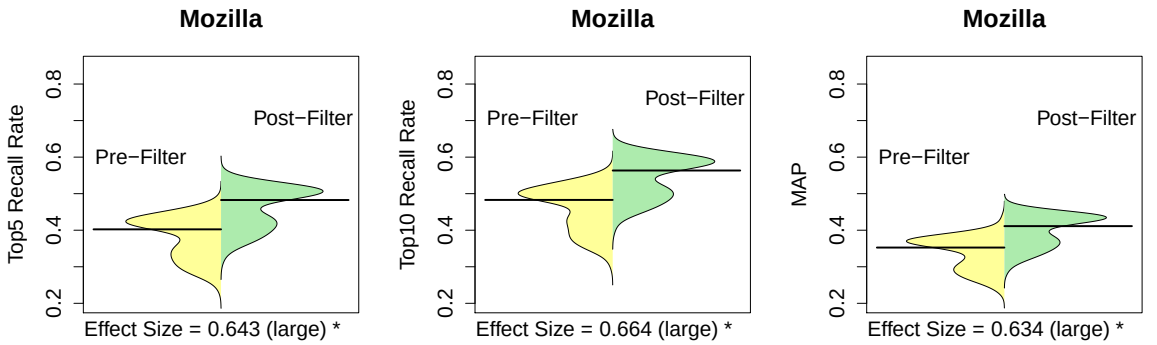
Longer training periods do not increase the performance. Figure 5.20 shows the performance of the REP approach after applying various values of n for the number of months used in the training data. There is no significant difference between the performance of the various values of n. I observed similar behaviour for the BM25F approach, hence I omit the figures for BM25F from the chapter.

> While applying the realistic evaluation, leveraging the resolution field value and the rank to filter the returned master candidates improves the overall performance of automated approaches for the retrieval of duplicate reports with a median of 10-21.5% for Recall and 7-18% for MAP. The thresholds that are used by the filters can be optimized automatically using a genetic algorithm.

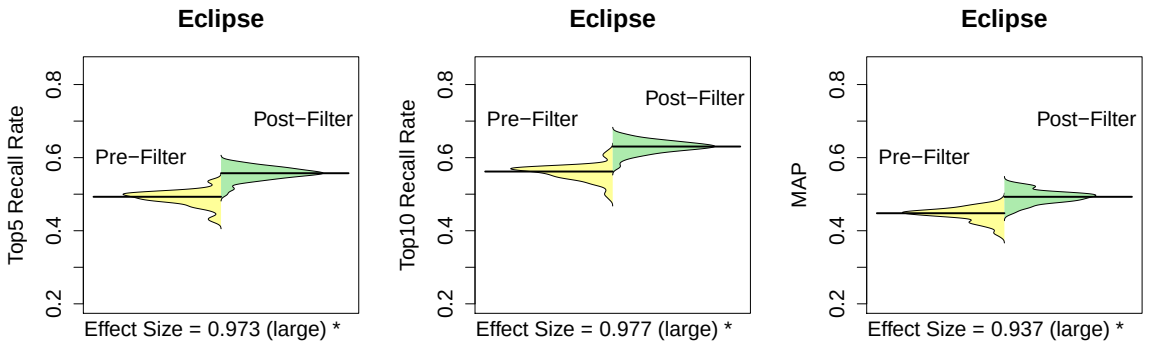
5.7 Threats to Validity

In this section, I discuss the threats to the validity of my conclusions.

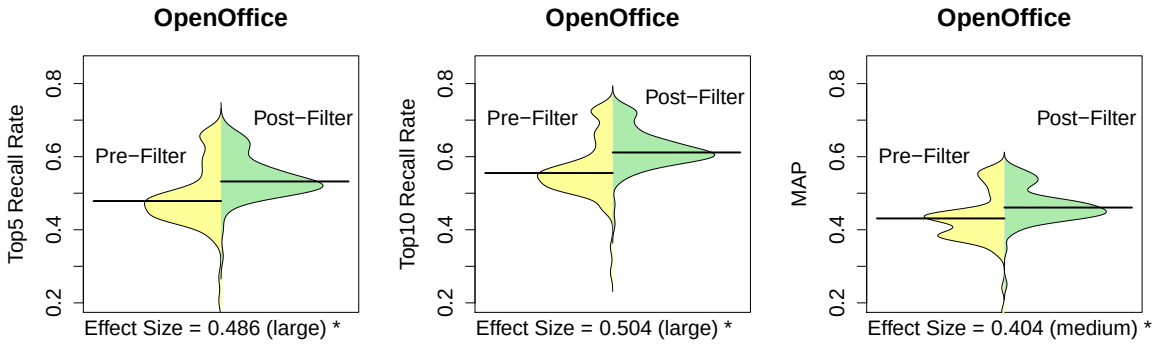
External Validity. One of the external threats to my results is generalization. In this chapter, I studied three ITSs of open source software systems of different sizes and from different domains. Developers in open-source software projects could have different behavior for retrieving duplicate issue reports compared to developers of commercial software. In addition, all my studied projects make use of the Bugzilla issue tracking system. My findings might not hold true for other software projects with other types of issue tracking systems. In order to address this threat, additional case studies on other projects (both open source and commercial), with other types of issue tracking systems (e.g., JIRA) are needed. This chapter shows that one should not simply ignore



(a) Mozilla



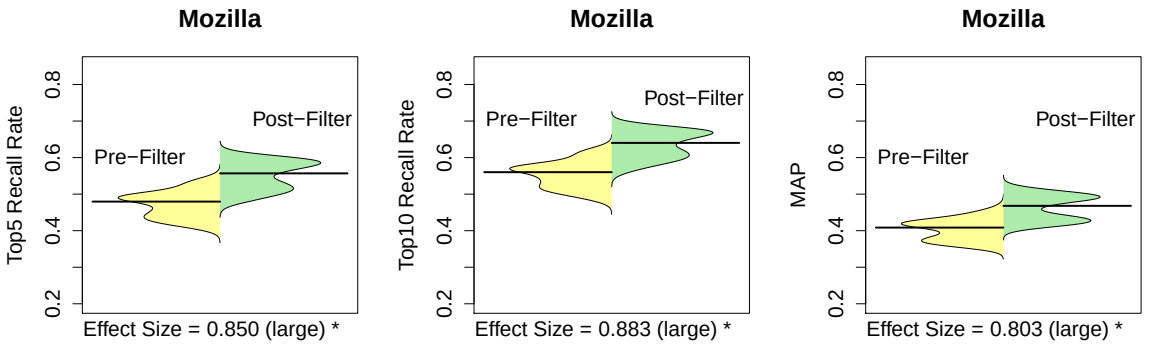
(b) Eclipse



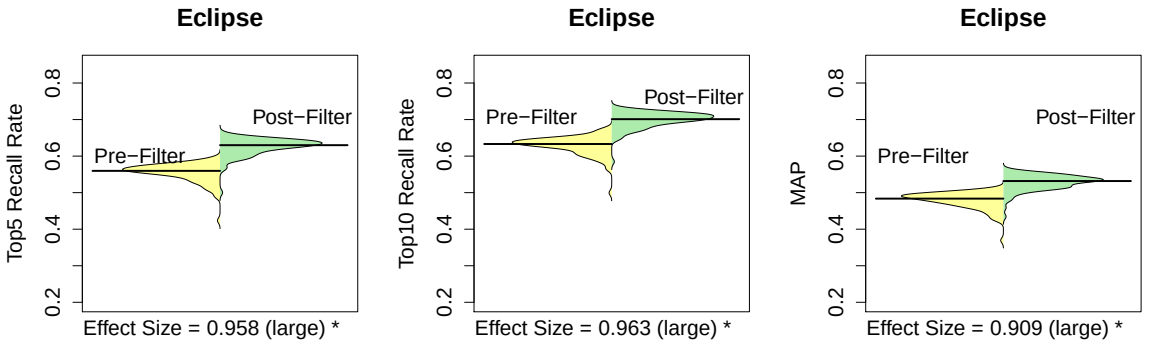
(c) OpenOffice

$^{*}\, p < 0.001$

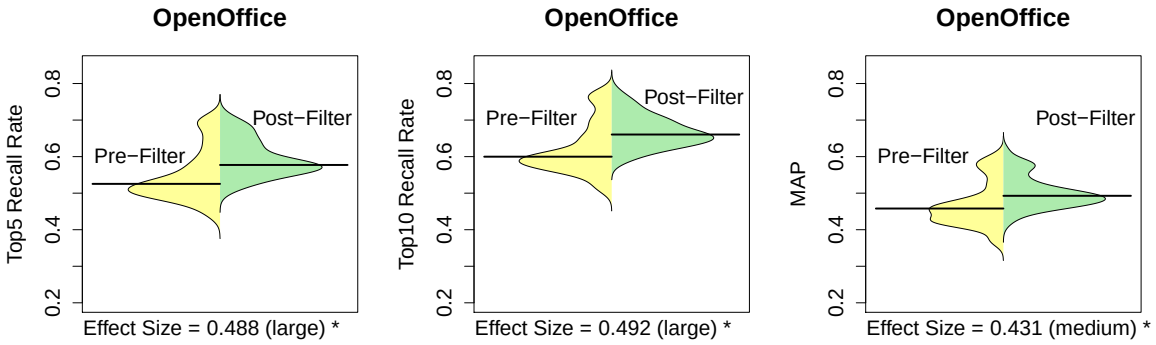
Figure 5.17: The impact of using the resolution field on the results of the BM25F approach (2 months training data).



(a) Mozilla



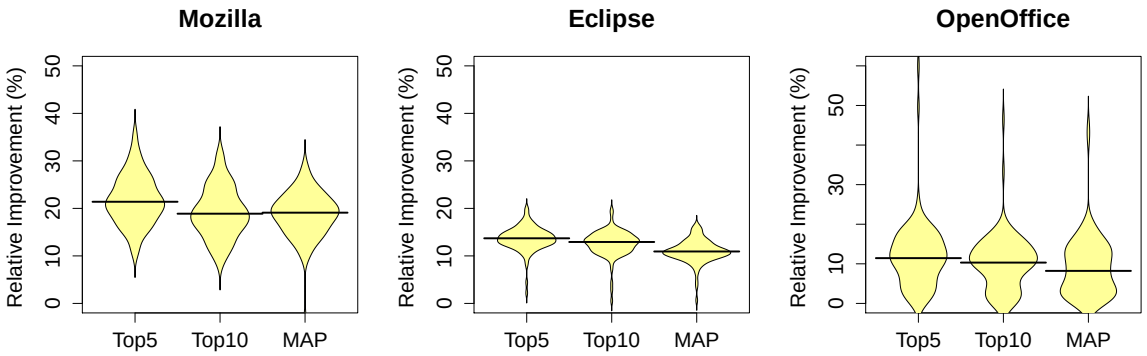
(b) Eclipse



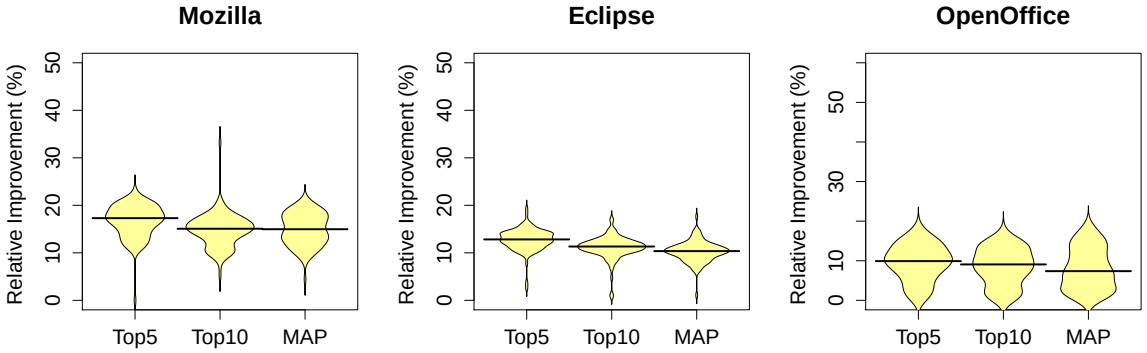
(c) OpenOffice

** $p < 0.001$*

Figure 5.18: The impact of using the resolution field on the results of the REP approach (2 months training data).



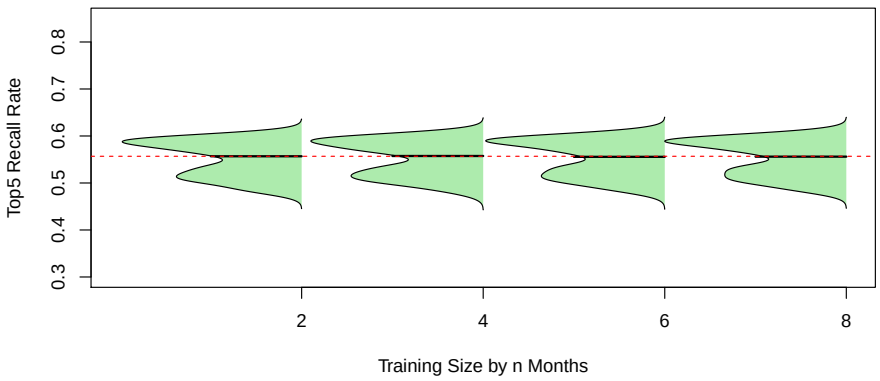
(a) BM25F



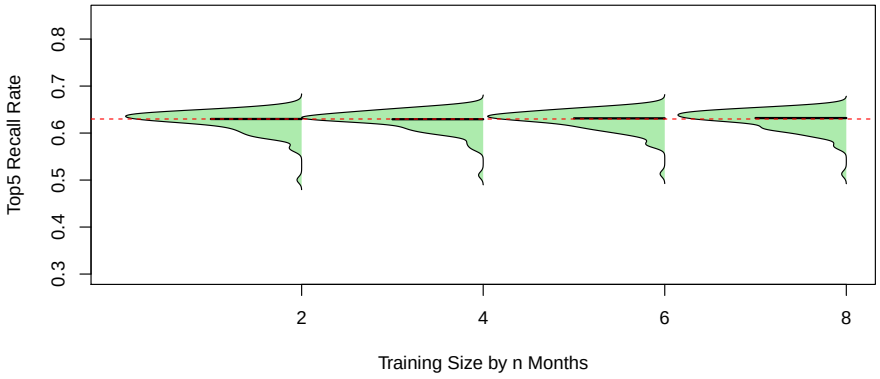
(b) REP

Figure 5.19: The relative improvement in performance after filtering the results using my approach.

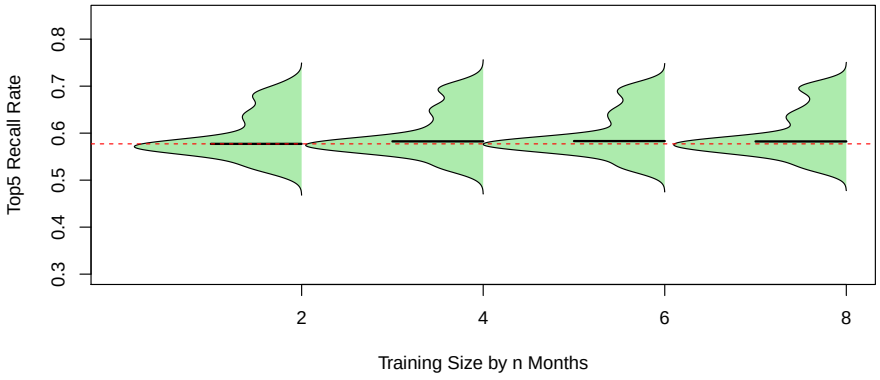
data during the evaluation of approaches for retrieving duplicate issue reports while expecting that there is no impact on the evaluated performance of these approaches. However, more studies are needed to understand whether such impact is significant on a larger number of projects.



(a) Mozilla



(b) Eclipse



(c) OpenOffice

Figure 5.20: The REP approach: applying the proposed approach using different n-training months. The red dotted line is the median Recall when applying 2 months for training as was done to yield the leftmost distribution. There are no significant differences in the performance.

This study is the first to explore the design of the performance evaluation of automated approaches for retrieving duplicate issue reports. In this chapter, I do not claim that the proposed evaluation is the most possible realistic evaluation. Therefore, I expect and hope that future work will dig even deeper into the concept of realistic evaluation (while defining it better).

Construct Validity. In my experiments, I tested the automated retrieval of duplicates using two approaches which can threaten the generality of my findings to other approaches. Almost all approaches for retrieving duplicate issue reports share the same base technique (TF-IDF). In this chapter, I apply REP and BM25F as these are by far the most used TF-IDF-based approaches for the retrieval of duplicate issue reports (Sun et al., 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Hindle et al., 2015; Zou et al., 2016). REP (Sun et al., 2011) and BM25F (Robertson et al., 2004) have always been treated as two separate approaches in prior research (Sun et al., 2011; Nguyen et al., 2012; Zou et al., 2016). The REP approach depends on a ranking function that combines the $BM25F_{ext}$ approach (which is an extended version of BM25F by Sun et al. (2011)) along with categorical fields of issue reports. Nowadays, even the most recent approaches make only small increments to the REP or BM25F approach (Nguyen et al., 2012; Aggarwal et al., 2015; Hindle et al., 2015; Zou et al., 2016). Therefore, studying the REP and BM25F approaches covers the majority of the spectrum of the approaches for retrieving duplicate issue reports. Hence, the observations in this chapter are highly probable to hold for similar approaches in literature which increases the applicability and impact of my findings. My findings are general in nature, e.g., it is intuitive that excluding issue reports will affect the performance evaluation of other approaches for the retrieval of duplicate issue reports as well. Nevertheless, future studies are necessary

to study whether my results are indeed generalizable to the evaluation of approaches that are not based on BM25F or REP.

In the MAP metric calculation, I used 1,000 as the maximum list size. The list size of 1,000 includes more than 90% of all duplicates which yields accurate results for calculating the MAP metric for all available duplicates. I did not use 100% MAP because this is extremely computationally intensive (Nguyen et al., 2012). In addition, my choice of list size does not significantly affect the results as the practical applicability of the candidates that are ranked outside the top 1,000 is negligible.

For the experiments that are applied on each studied ITS, I 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). However, running the approaches for retrieving duplicate issue reports is costly in terms of time, hence I limit the number of chunks that I evaluate to 100 randomly-selected chunks. Evaluating 100 chunks of one-year length should give a high enough confidence as the evaluated chunks randomly cover the full lifetime of the studied ITSs.

I evaluated only how leveraging the resolution field impacts the performance in RQ3. The main motivation behind the selection of the resolution field is that intuitively, the resolution is related to the chances of a newly-submitted report being a duplicate. For example, if an issue report was resolved as fixed several years ago, the chances of a newly-submitted report being a duplicate of that report are relatively small. Hence, I use the resolution field in combination with thresholds for the time and rank to give a lower priority. At this moment, I have no such intuition for the other available fields. I encourage and expect future work to first explore theoretical or intuitive underpinnings and then to search for fields that can further improve the performance of the

automated retrieval of duplicate reports.

5.7.1 The Sensitivity of the Choice of the Tuning Data

As explained in Section 5.3, both REP and BM25F have parameters that are tuned to find their optimal values (Robertson et al., 2004; Sun et al., 2011). The BM25F approach has 10 parameters while the REP approach has 19 parameters. In prior work on retrieving duplicate issue reports (Sun et al., 2010, 2011; Nguyen et al., 2012; Aggarwal et al., 2015; Lazar et al., 2014; Hindle et al., 2015), the number of duplicate issue reports in the tuning data set was always fixed to $N = 200$. In this section, I do a preliminary investigation of whether the choice of N impacts the performance of the automated approaches for retrieving duplicate issue reports. I conduct the realistic evaluation after tuning BM25F and REP with a tuning data set that contains 10, 50, 100, 200, 1000, 25%, 50% and 100% of the available duplicate issue reports in each ITS. In addition, I investigate whether the contents of the tuning set impact the performance, by conducting the realistic evaluation after tuning with a tuning data set of (1) 200 long-term duplicates, (2) 200 short-term duplicates and (3) 100 long-term and short-term duplicates. I selected short-term duplicates as duplicates that were reported within six months, and I selected long-term duplicates as duplicates that were reported at least two years apart. I take the following steps to conduct my investigation:

1. Randomly select N duplicate issue reports from the ITS for the aforementioned values of N.

2. Tune BM25F and REP for each ITS.

3. Calculate the performance metrics for each ITS and automated approach for the

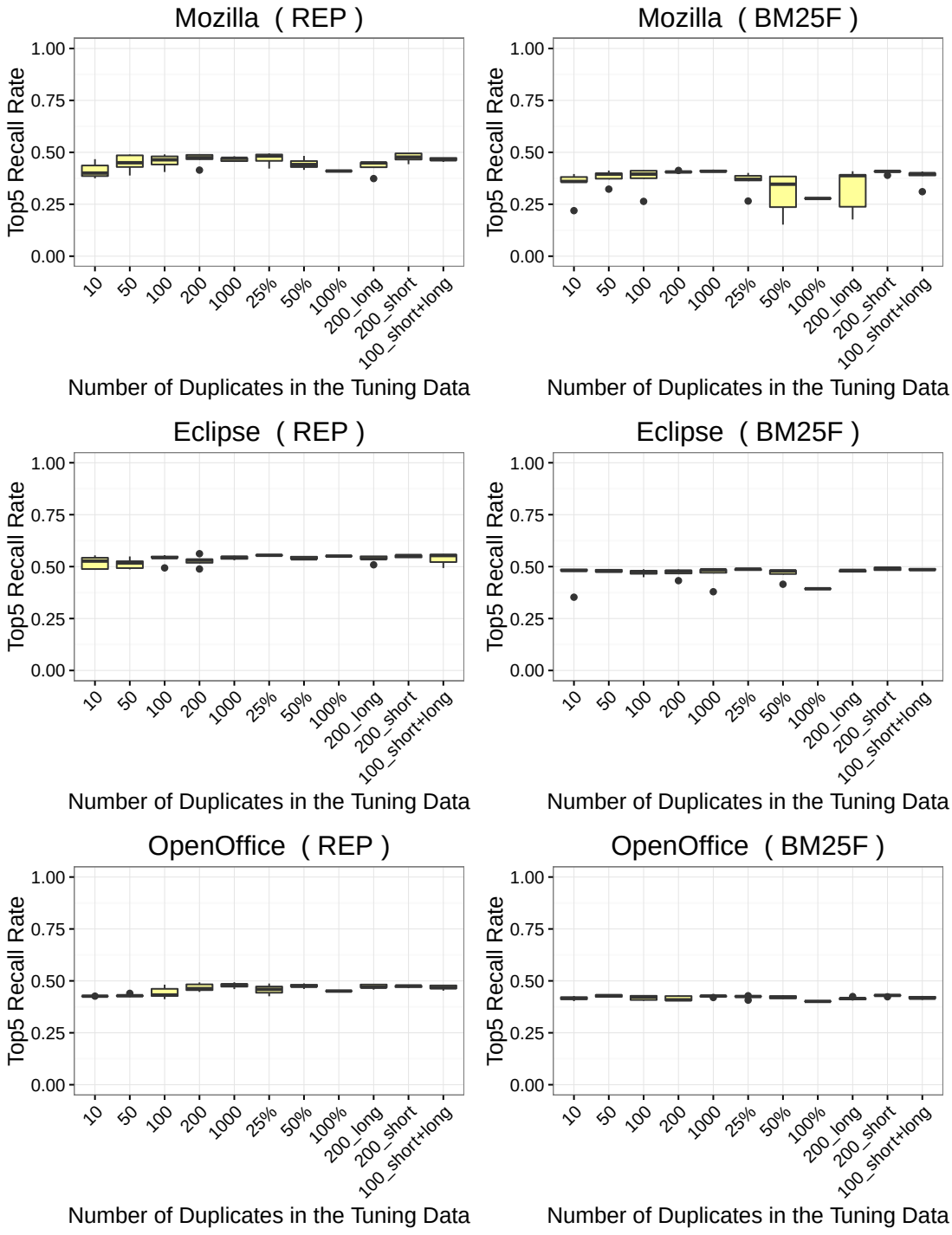


Figure 5.21: The $\text{Recall}_{\text{top5}}$ of the BM25F and REP approaches (as obtained by the realistic evaluation) after tuning with different tuning data sizes for 5 runs per tuning data size.

time periods that are shown by Table 6.2.

I repeat the steps above five times for each value of N (except for $N = 100\%$, as no random selection is possible) and the three settings with long and short-term duplicates. Figure 5.21 shows the results of my investigation.

I observe that for Eclipse and OpenOffice, the choice of tuning data does not affect the performance of the approach greatly. For Mozilla, the differences are larger. In particular, there are some choices of tuning data which negatively affect the performance of BM25F. Throughout this chapter, I used the exact same tuning data for my runs of the realistic and classical evaluation. Hence, my results are not affected by the choice of tuning data. However, my observations are yet another indication of how a simplistic evaluation (e.g., using a single run) of the performance of automated approaches for the retrieval of duplicate issue reports can lead to an overestimation of the reported performance. Future studies should address in more depth the impact of tuning in the retrieval of duplicate issue reports using my proposed realistic evaluation.

5.8 Chapter Summary

In an issue tracking system (ITS), users of a project can submit issue reports that describe a bug, a feature request or a change request. As a result, some issues are reported multiple times by different users. In order to minimize the effort spent on such duplicate issue reports, automated approaches have been proposed for retrieving these duplicate reports.

In general, these approaches are evaluated using a small subset of issue reports available in an ITS. In the first part of this chapter, I show that this type of evaluation

relatively overestimates the performance of these approaches by a median of 17-42% for the ITSs of the Mozilla foundation, the Eclipse foundation and OpenOffice using BM25F and REP, two popular approaches for retrieving duplicate issue reports. Instead, I propose to use a type of evaluation that uses all issue reports available in the ITS, yielding a more realistic performance of the approach.

In the second part of this chapter, I show that using the resolution field value of an issue report during the retrieval of duplicate reports can yield a relative performance improvement of a median of 10-22% for Recall and 7-18% for MAP.

The main takeaways of this chapter are as follows:

1. In future studies of automated retrieval of duplicate issue reports:
   (a) Instead of a single value, a range of values of a performance metric should be reported, and
   (b) The realistic evaluation as proposed in this chapter should be used instead of the classical evaluation as traditionally used.
2. Using the resolution field value of an issue report can significantly improve the performance of the retrieval of duplicate issue reports.

I acknowledge that the proposed realistic evaluation in chapter requires considerably more computational power than the classical evaluation. However, the wide availability of computation clusters makes the realistic evaluation feasible for most researchers. In addition, the results presented in this chapter show that the relative overestimation reported by the classical evaluation is a serious issue. Hence, researchers should no longer relegate the issue by estimating performance using the classical evaluation because of the high computational costs of the realistic evaluation.

CHAPTER 6

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

This chapter investigates the impact of the "just-in-time duplicate retrieval feature" on the duplicate reports that end up in the ITS. In particular, I study the differences between duplicate reports before and after the activation of this new feature. I 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. In addition, I compare the performance of the state of the art automated retrieval of duplicate reports using two popular approaches. I 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 identified 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. I observe that the performance gap between BM25F and REP becomes even larger after the activation of the just-in-time duplicate retrieval feature. I 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.

An earlier version of this chapter is under review at the Empirical Software Engineering Journal (EMSE).

6.1 Introduction

To evaluate the performance of prior research 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 (BuzillaRelease, 2017) and Jira 6.0 (Jira, 2017)), 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.

This chapter 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. I study duplicate issue reports from three large projects (Mozilla-Firefox, Mozilla-Core and Eclipse-Platform). I 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 retrieval discussion. In addition, I evaluate the performance of two popular approaches, BM25F (Robertson et al., 2004) and REP (Sun et al., 2011), for the automated retrieval of duplicate reports on before and after-JIT

duplicates. In particular, I 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?

> 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. I show that the performance gap between BM25F and REP is even larger for after-JIT duplicates. The results in this chapter 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.

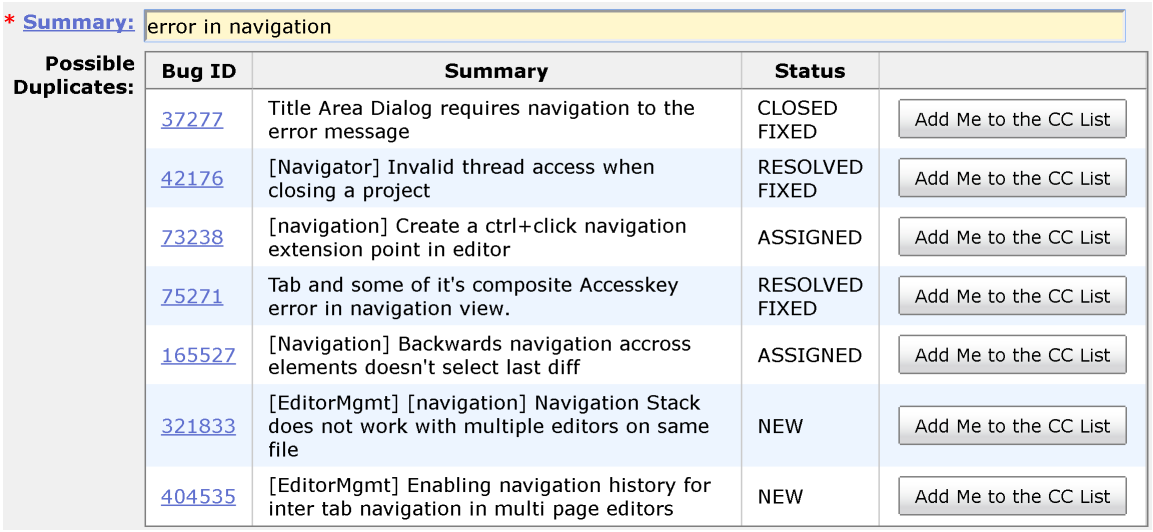


Figure 6.1: An example of the JIT duplicate retrieval feature in Bugzilla.

6.2 The Main Contributions of this Chapter

This chapter of the thesis adds the following contributions:

- Investigates the impact of the JIT duplicate retrieval feature on the characteristics of duplicate reports.
- Studies the impact of the JIT feature on the performance evaluation of automated approaches for the retrieval of duplicate issue reports.

6.3 Background

In this section, I present an overview of the JIT duplicate retrieval feature and discuss related work.

6.3.1 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[1] at filing time to assist in the retrieval of possible duplicate reports. Figure 6.1 shows an example of 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[2]. Such a feature was frequently requested by Bugzilla users (see #22353[3] in the Bugzilla project). The JIT duplicate retrieval feature in Bugzilla depends only on the contents of the summary field. In this study, I 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*.

6.3.2 Related Work

There are several prior studies on duplicate issue reports and the automated retrieval of such reports (see Chapter 3). In this section, I survey work that is related to my study.

[1] http://dev.mysql.com/doc/internals/en/full-text-search.html
[2] https://github.com/bugzilla/bugzilla/blob/master/Bugzilla/Bug.pm#L599
[3] http://bugzilla.mozilla.org/show_bug.cgi?id=22353

Study	Before-JIT Data	After-JIT Data
Runeson et al. (2007)	✓	
Jalbert and Weimer (2008)	✓	
Wang et al. (2008)	✓	
Sun et al. (2010)	✓	
Sureka and Jalote (2010)	✓	
Sun et al. (2011)	✓	
Nguyen et al. (2012)	✓	
Cavalcanti et al. (2013)	✓	
Rakha et al. (2015) [Chapter 4]	✓	
Aggarwal et al. (2015)	✓	
Banerjee et al. (2016)*	✓*	✓*
Rakha et al. (2017) [Chapter 5]	✓	

* Banerjee et al. (2016) mixed Before-JIT and After-JIT in the same evaluation

Table 6.1: A summary of prior research based on their usage of before-JIT and after-JIT duplicate reports in their evaluation (ordered by publication date).

Data Selection for Performance Evaluation

Prior work uses different issue reports from the ITSs for their performance evaluation. Table 6.1 shows an overview of prior research based on the data that was used. I observe that almost all studies were applied on issues that were reported before the activation of the JIT duplicate retrieval feature. However, in this chapter I 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, I focus on highlighting the differences between before and after-JIT duplicates. In addition, I study how prior work is affected by these differences.

6.4 Experimental Setup

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

6.4.1 Studied Projects:

In this chapter, I selected the studied projects based on the following criteria:

1. **Project uses the JIT duplicate retrieval feature:** the project uses an ITS that had 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).

For the experiments in this chapter, I 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 [4] and Eclipse[5] foundation started using Bugzilla 4.0, including the JIT duplicate retrieval feature, in 2011. I 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 (Bettenburg et al., 2008b; Rakha et al., 2015). 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 (Anvik et al., 2005; Bettenburg et al., 2008a; Sun et al., 2011; Alipour et al., 2013; Rakha et al., 2015).

I crawled the issue reports' XML files for the studied projects up to 31-Dec-2015. Then, I parsed the XML files for the further analysis that is presented in this chapter.

[4] https://www.bugzilla.org/news/
[5] https://bugs.eclipse.org/bugs/show_bug.cgi?id=359299

Table 6.2: 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%)

Table 6.2 shows the number of duplicate reports for each studied software project. I 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.

6.4.2 Pre-processing of Issue Reports

For this chapter, I executed the same text pre-processing explained in Chapter 5.

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

I examine the characteristics of duplicates using the following metrics:

Ratio of duplicates: This ratio is the proportion of issue reports that are duplicates within a given month (Anvik et al., 2005; Bettenburg et al., 2008b). I divide the number of duplicate issues that are reported within one particular month by the total number of reported issues in that month. I 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. I 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.

Identification delay: As described in Chapter 4, the identification delay is the time in days from triaging to resolving an issue report as a duplicate. I study this time as it is an indication of how long it takes developers to identify a duplicate. I expect that duplicate reports after the activation of the JIT retrieval feature need more time to be identified, because more time may be spent on less similar duplicates (Rakha et al., 2015).

Identification discussions: As described in Chapter 4, 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 #****). I 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. I 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.

Textual similarity: State of the art automated approaches for retrieval of duplicates, such as BM25F and REP, rely heavily on textual similarity. Generally, the duplicates that are most similar 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 retrieved (Rakha et al., 2015). As suggested in Chapter 4, I reuse the trigram

textual similarity (Sureka and Jalote, 2010; Lazar et al., 2014) to highlight the differences in the duplicate reports' similarity to their master reports. 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, I removed all special characters, double spaces and converted all the text to lowercase. Then, I applied the trigram algorithm to calculate the similarity between texts the description fields of the duplicate issue reports and their masters. I 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.

6.5 Experimental Results

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

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 (Anvik et al., 2005; Bettenburg et al., 2007, 2008a; Rakha et al., 2015). Therefore, many studies have proposed automated approaches to assist in the retrieval of duplicate reports (Runeson et al., 2007; Wang et al., 2008; Sun et al., 2011; Nguyen et al., 2012; Alipour et al., 2013; Aggarwal et al., 2015; Jalbert and Weimer, 2008; Hindle et al., 2015). In addition, recent versions of ITSs provide a JIT duplicate retrieval (see Section 6.3.1), which shows

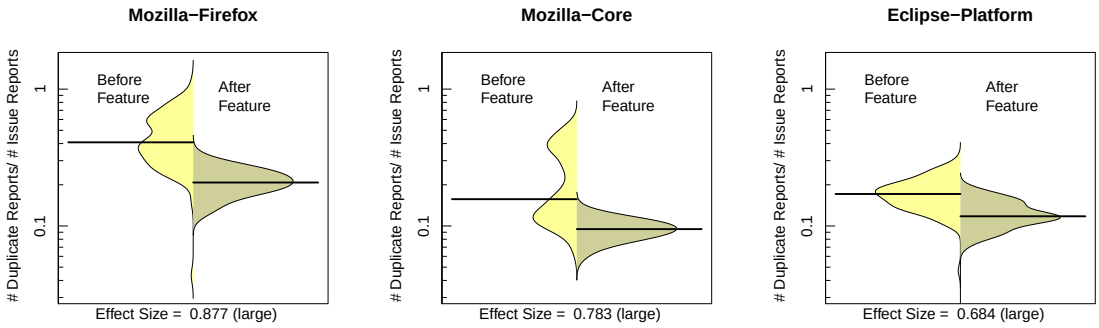
a list of candidate duplicates for each report at filing time. In the evaluation of the proposed approaches, prior work never considered the possible impact of the JIT duplicate retrieval feature on the duplicate reports that end up in the ITS. In this chapter, I study this impact. First, I compare the characteristics of before-JIT and after-JIT duplicates in this RQ. Examining the differences between before-JIT 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, I compared the characteristics of the before-JIT and after-JIT duplicates. I 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 6.4.1, I 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, I have many years of issue reports that were reported after the activation of the JIT duplicate retrieval feature. I 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, Identification discussions and Textual similarity) that were presented in Section 6.4.3.

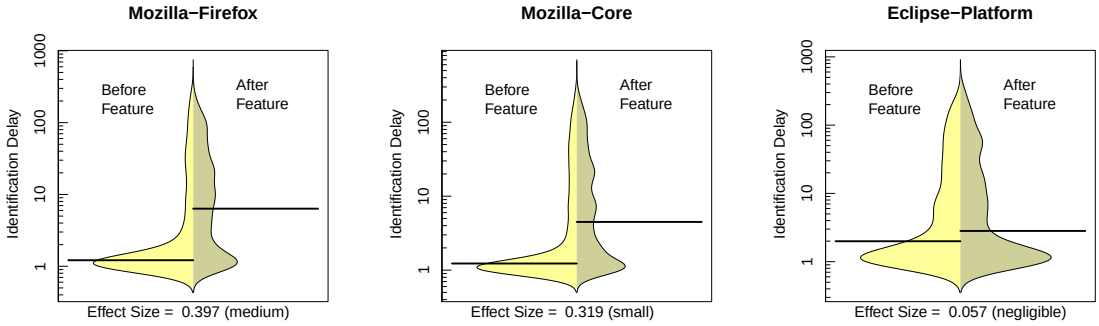
For each metric, I compared the two groups using the *Mann − Whitney U* statistical test Gehan (1965). I use the following hypotheses:

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

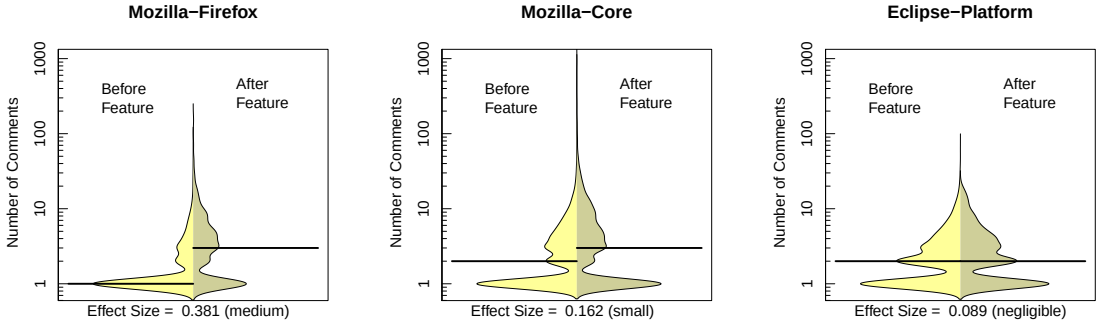
H_1 = The two groups of duplicate reports have different metric values.



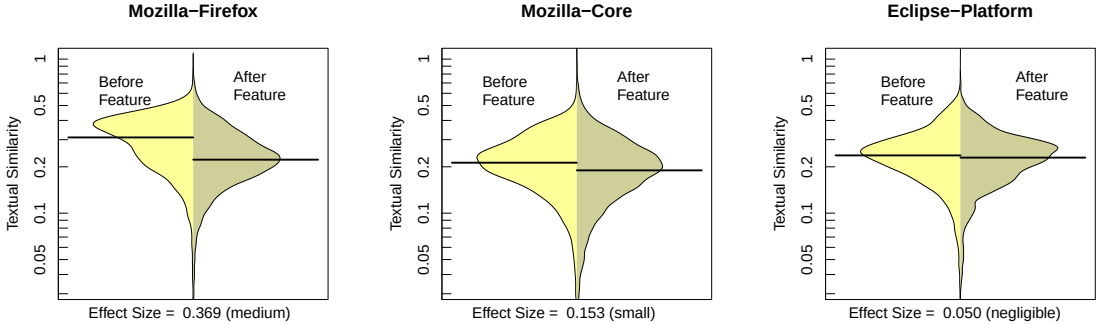
(a) Ratio of Duplicates



(b) Identification Delay



(c) Identification Discussions



(d) Textual Similarity

Figure 6.2: 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.

I reject H_0 and accept H_1, when $p < 0.01$. In addition, I calculated the *effect size*, as the effect size quantifies the difference between the two group distributions. I used *Cliff's Delta* as it does not require the distributions normality assumption to quantify the effect size (Long et al., 2003). I use the same thresholds that are mentioned in Chapter 5.

Results. **Significantly fewer duplicate reports end up in an ITS after the activation of the JIT duplicate retrieval feature.** Figure 6.2a 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. I 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 6.2b and 6.2c show the distributions of the identification delay and identification discussions for the studied projects. I 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 my manual investigation, one possible explanation for my 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[6].

[6]Issue#393235: https://bugs.eclipse.org/bugs/show_bug.cgi?id=393235. I manually

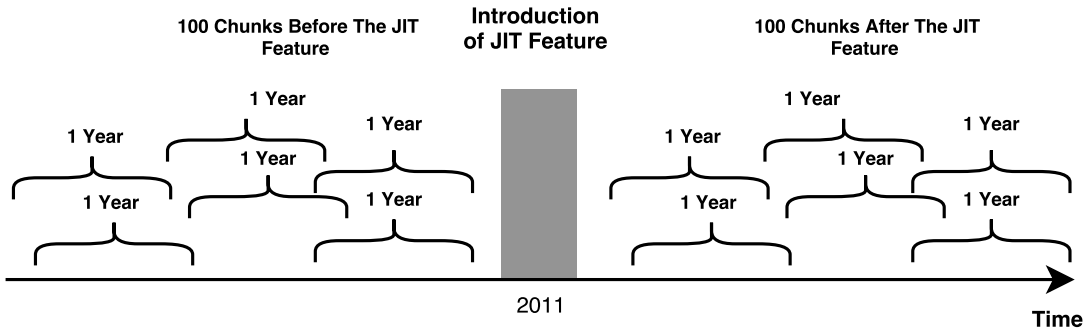
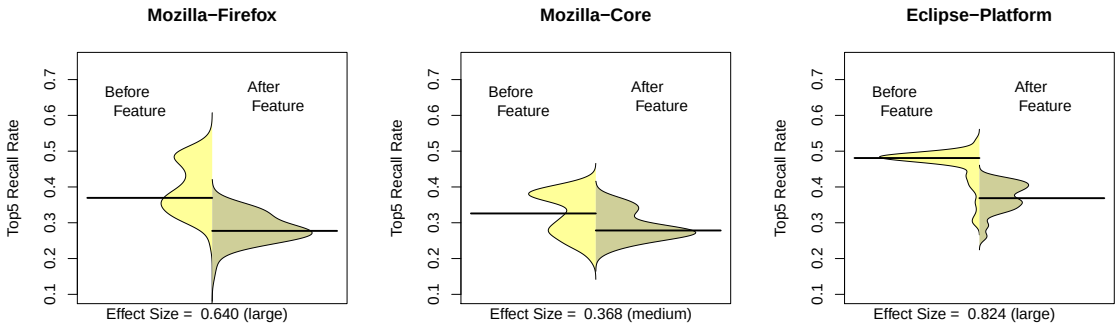


Figure 6.3: The selection of data chunks before and after the activation of the JIT duplicate retrieval feature.

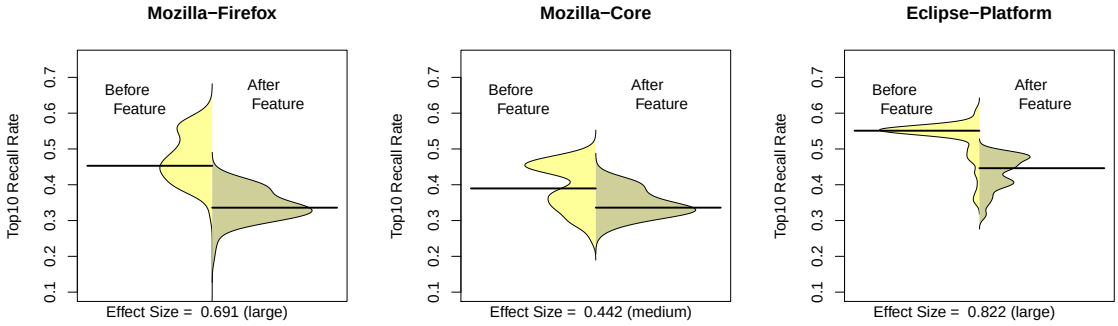
After-JIT duplicates share a significantly smaller textual similarity with their master reports. Figure 6.2d shows the distributions of the textual similarity of the duplicate reports of the studied projects. I observe that after-JIT duplicates share a smaller textual similarity with their masters than 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, I believe that future ITSs need to employ additional techniques to retrieve duplicates. In the following RQ, I delve deeper into my observations about current automated approaches for the duplicate retrieval.

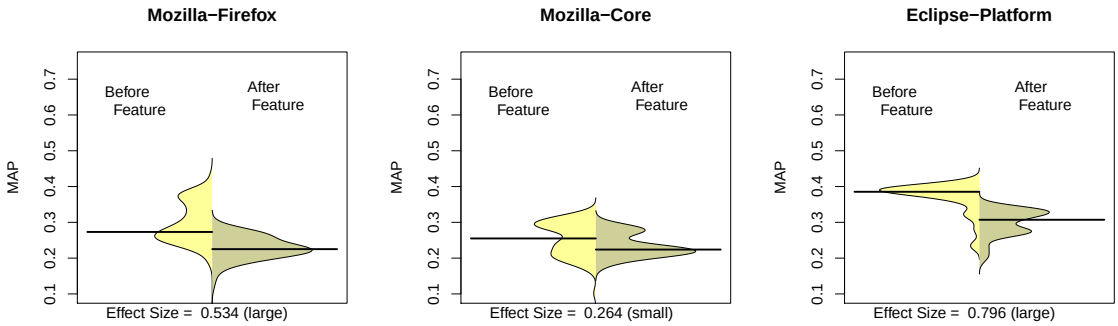
verified that this issue still persists, as of (September- 2017).



(a) Top-5 Recall rate

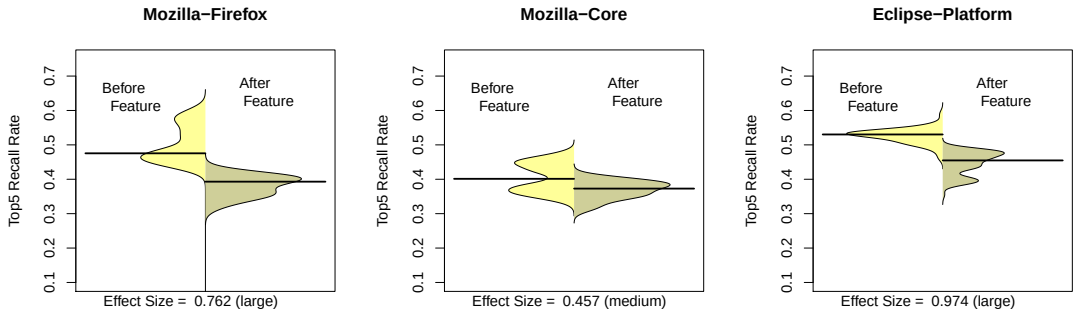


(b) Top-10 Recall rate

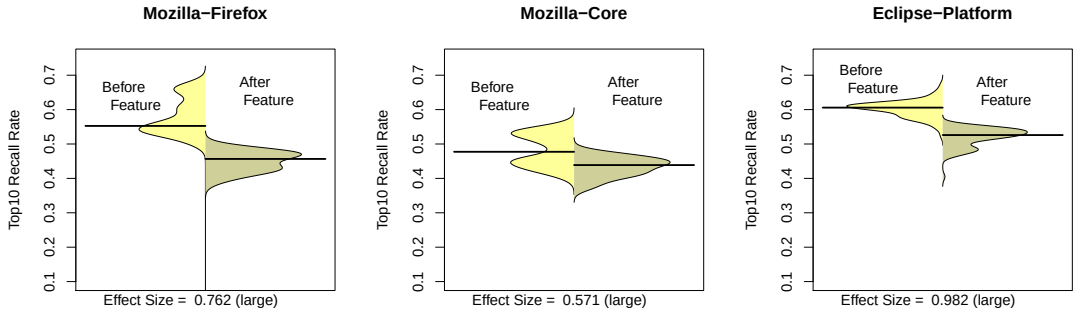


(c) MAP

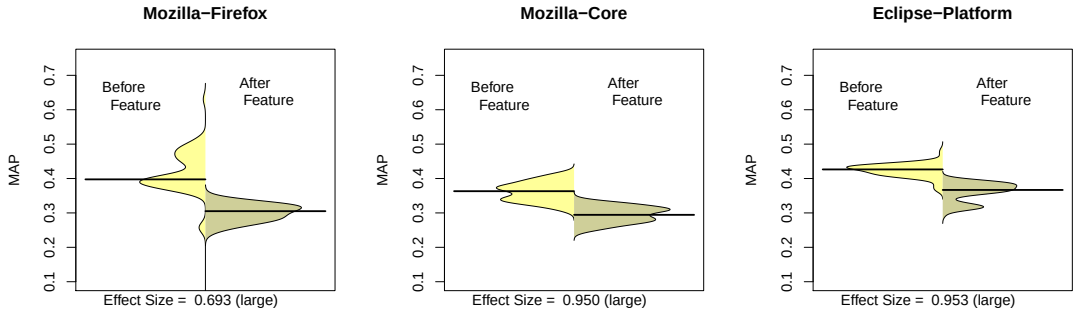
Figure 6.4: A comparison of the performance of BM25F before and after the activation of the JIT duplicate retrieval feature for the studied projects.



(a) Top-5 Recall rate

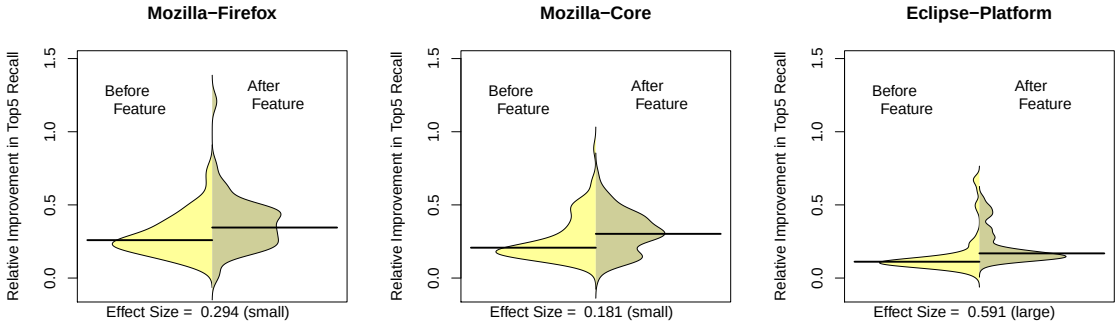


(b) Top-10 Recall rate

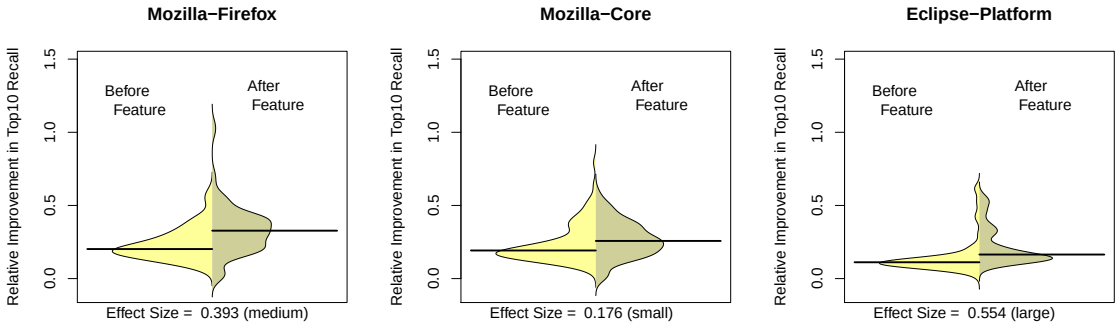


(c) MAP

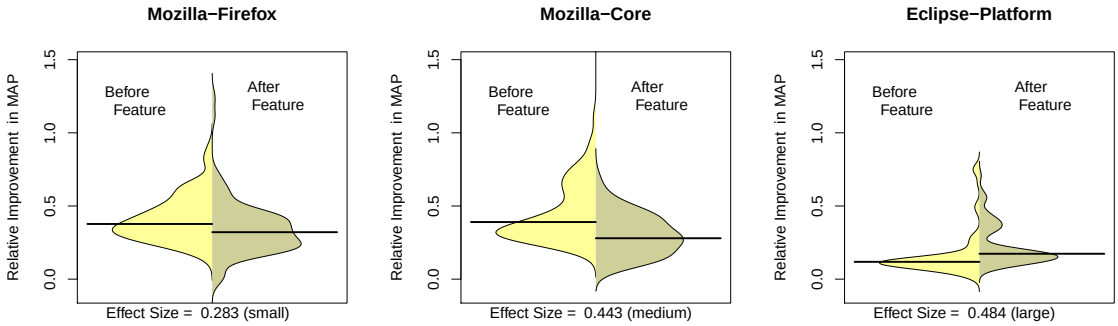
Figure 6.5: A comparison of the performance of REP before and after the activation of the JIT duplicate retrieval feature for the studied projects.



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



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



(c) Relative improvement in the MAP

Figure 6.6: 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).

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, I 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, I compare the performance of the BM25F and REP approaches when evaluated on before-JIT and after-JIT duplicates. First, I randomly picked 100 chunks of before-JIT duplicates and 100 chunks of after-JIT duplicates from the studied ITSs (similar to Chapter 5). Each chunk of data is of one year-length (see Figure 6.3) and consists of all issue reports that are reported in one year. For each studied project, I could 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, I used the first 200 duplicates of that chunk to tune the approach (Sun et al., 2011; Nguyen et al., 2012; Lazar et al., 2014; Aggarwal et al., 2015; Hindle et al., 2015), then I calculated the performance of the BM25F and REP approaches on the remaining duplicates in that chunk. Then, I compared the performance before and after the activation of the JIT duplicate retrieval feature from two perspectives:

Performance per Approach: for both BM25F and REP, I 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.

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 (Sun et al., 2010, 2011; Nguyen et al., 2012; Lazar et al., 2014; Aggarwal et al., 2015; Hindle et al., 2015; Zou et al., 2016; Zhou and Zhang, 2012b). Prior research has already shown that REP outperforms BM25F (Sun et al., 2011; Nguyen et al., 2012; Zhou and Zhang, 2012b). However, this result followed from an evaluation using before-JIT duplicates. I study whether such prior result still holds for after-JIT duplicates. Similar to prior research (Sun et al., 2010, 2011; Nguyen et al., 2012; Lazar et al., 2014; Aggarwal et al., 2015; Hindle et al., 2015; Zou et al., 2016; Zhou and Zhang, 2012b), I 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:

$$\text{Relative Improvement} = \frac{measure_{REP} - measure_{BM25F}}{measure_{BM25F}} \tag{6.1}$$

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 my 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, I compared the distributions of performance measures using the Mann–Whitney U test and Cliff's Delta effect size.

Results. **The performance of both BM25F and REP is lower for after-JIT duplicates.** Figures 6.4 and 6.5 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 identification 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 6.6 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.

> My results show that duplicate retrieval based on basic textual similarity is no longer useful, as the JIT duplicate retrieval feature finds 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.

6.6 Implications

In this section, I discuss the implications of my work for researchers and ITS developers that are interested in the automated identification of duplicate reports.

The JIT duplicate retrieval feature appears to lower the number of duplicates that end up in the ITS. My 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 (Runeson et al., 2007; Wang et al., 2008; Sun et al., 2011; Nguyen et al., 2012; Alipour et al., 2013; Aggarwal et al., 2015; Jalbert and Weimer, 2008; Hindle et al., 2015), 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 identification 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 identification using after-JIT duplicates. In RQ1, I showed how after-JIT duplicates differ from before-JIT duplicates. In addition, I 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.

6.7 Threats to Validity

In this section, I discuss the threats to the validity of my study.

6.7.1 External Validity

These threats concern the ability of this chapter to generalize my findings to other software projects. In this study, I 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. 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[7]. Future studies should investigate the impact of requiring a manual action to trigger the JIT feature.

I 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 of my study. However, the large majority of the studies on the automated retrieval of duplicate reports focus on Bugzilla (Runeson et al., 2007; Nguyen et al., 2012; Alipour et al., 2013; Hindle et al., 2015; Aggarwal et al., 2015; Sun et al., 2011; Zou et al., 2016; Zhou and Zhang, 2012b). Therefore, my 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 chapter, I focused on REP and BM25F as these are by far the most used TF-IDF-based approaches for retrieving duplicate issue reports. REP (Sun et al., 2011) and BM25F (Robertson et al., 2004) have always been treated as two separate approaches in prior research (Sun et al., 2011; Nguyen et al., 2012; Zou et al., 2016). The REP approach depends on a ranking function that combines the BM25Fext approach, (which is an extended version of BM25F by Sun et al. Sun et al. (2011)), along with categorical fields of issue reports. Nowadays,

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

even the most recent approaches make only small increments to the REP or BM25F approach (Nguyen et al., 2012; Aggarwal et al., 2015; Hindle et al., 2015). Therefore, studying the REP and BM25F approach covers the majority of the spectrum of the approaches for retrieving duplicate issue reports.

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

6.7.2 Internal Validity

In this study, I assumed that the only impactful change in ITSs in 2011 was the activation of the JIT duplicate retrieval feature. In addition, I assumed that the JIT duplicate retrieval feature was not deactivated after its activation. 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.

I 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, I limited the number of chunks that I evaluated to 100 randomly-selected chunks of before-JIT and after-JIT duplicates. Evaluating 200 chunks (i.e., each chunk is one-year length) in total should

give a strict enough confidence interval.

6.8 Chapter Summary

In this chapter, I explored the impact of the activation of the JIT duplicate retrieval feature on the automated retrieval of duplicate reports. First, I 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, I explored four metrics of characteristics covering the ratio of duplicates, identification delay, retrieval discussion, and textual similarity. Second, I 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 this chapter 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 identified 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, I showed that both BM25F and REP perform significantly lower on after-JIT duplicates.

This chapter findings highlight that future studies of the automated retrieval of duplicate issue reports have to focus on after-JIT duplicates, as these duplicates are more representative of new issue reports. In addition, my findings indicate that automated retrieval of duplicates based on basic textual similarity is no longer effective, as many textually similar duplicates are already identified by the JIT duplicate retrieval feature (hence are not even filed). 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.

CHAPTER 7

Conclusion and Future Work

This chapter condenses the research ideas that are presented in this thesis. In addition, I propose avenues for future research.

7.1 Thesis Contributions

The goal of this thesis is to highlight a set of experimental design choices that are important to be considered in the future research when evaluating an automated approach for duplicate report retrieval. Below, I summarize the main conclusions of this thesis:

1. **Duplicate reports can be classified into two groups based on the effort needed**

for identification. Duplicate reports often provide new information and valuable information to developers Bettenburg et al. (2008b). However, the manual task of identifying the duplicate reports is tedious and time-consuming. The manual task of identifying the duplicate reports is tedious and time-consuming. In this thesis, I empirically study the needed effort for manually retrieving duplicate issue reports. I found that duplicate reports can be classified into two groups based on the needed effort. Such a finding highlights the importance of including the effort in measuring the performance of the automated approaches for the retrieval of duplicate reports (see Chapter 4). Future research should explore such an aspect in more depth.

2. **Prior evaluations are not realistic and overestimate the performance for the automated approaches for duplicate reports retrieval.** I demonstrate the performance overestimation caused by prior research evaluation choices that includes ignoring all the previously reported issues when testing a certain time period and dropping duplicate reports from the performance measures (see Chapter 5).

3. **A genetic algorithm-based approach for the filtration of previously reported issues improves the performance for the automated approaches for duplicate reports retrieval.** I propose a genetic algorithm based approach to filter the previously reported issues (old issues), and compare my approach with the traditional time frame-based filter (see Chapter 5).

4. **The evaluation on testing periods after-JIT feature shows a significant drop in the performance of the automated approaches for duplicate reports retrieval.**

I investigate the impact of the JIT feature on the performance of automated approaches for the duplicate reports. I compare the performance of automated approach for duplicate reports retrieval before-JIT feature and after-JIT feature. I show that future evaluations should focus on the after-JIT feature duplicates (see Chapter 6).

The aim of this thesis was to empirically revisit the design choices for the automated retrieval of duplicate reports. As a result, I propose an initial framework for evaluating the automated approaches. Based on the suggested framework, I highlight that researchers should not rely only on the overall number of duplicates retrieved accurately as the only measure for their evaluation. Researchers should also consider quantifying the efforts needed by the developers when using the automated approaches. Additionally, researchers should not ignore previously reported issues from the search without evaluating the impact of that decision. Researchers should try to ensure that their evaluations are realistic as possible as they can. Finally, researchers should be aware of the new ITS features such as the addition of JIT duplicate retrievals which may impact the evaluation metrics.

7.2 Opportunities for Future Research

The experimental choices that are revisited in this thesis show the importance of such kind of studies in highlighting hidden and unexplored areas in well-studied problems such as the automated retrieval of duplicate issue reports. In addition to the choices revisited in this thesis, there are other experimental choices and challenges that I think should be addressed by future research.

7.2.1 Integration with Binary Classification

In this thesis, I focused on the automated retrieval of duplicate issue reports. However, there is another automation that focuses on deciding whether a newly-reported issue was already reported before (i.e., without retrieving the actual previously filed report (see Section 3.2.2)). The automation can be called binary classification of duplicates or non-duplicates. Integrating the automated retrieval of duplicate reports with the binary classification Aggarwal et al. (2015); Hindle et al. (2015) should highlight the value of merging the two types of automation.

7.2.2 Tuning by Leveraging Learning to Rank Methods

The large majority of the recent studies on the automated retrieval of duplicate issue reports improve the performance by adding extra features or using more complex ranking functions (i.e., REP). However, a large domain in information retrieval works on improving the performance of the ranking functions by implementing different tuning algorithms which are called *learning to rank* Trotman (2005). The learning to rank topic has recently been touched by (Zhou and Zhang, 2012a) but never really got much investigation, especially where there exists a large variety of learning to rank algorithms.

Bibliography

Aggarwal, K., Rutgers, T., Timbers, F., Hindle, A., Greiner, R., and Stroulia, E. (2015). Detecting duplicate bug reports with software engineering domain knowledge. In *SANER 2015: International Conference on Software Analysis, Evolution and Reengineering*, pages 211–220. IEEE.

Alipour, A., Hindle, A., and Stroulia, E. (2013). A contextual approach towards more accurate duplicate bug report detection. In *MSR 2013: Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 183–192.

Angrist, J. D. and Pischke, J.-S. (2008). *Mostly harmless econometrics: An empiricist's companion*. Princeton university press.

Anvik, J., Hiew, L., and Murphy, G. C. (2005). Coping with an open bug repository. In *Eclipse 2005: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39. ACM.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *ICSE 2006: Proceedings of the 28th International Conference on Software Engineering*, pages 361–370. ACM.

Baeza-Yates, R. and Frakes, W. B. (1992). *Information retrieval: data structures & algorithms*. Prentice Hall.

Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., and Cukic, B. (2016). Automated triaging of very large bug repositories. *Information and Software Technology*.

Berry, M. W. and Castellanos, M. (2004). Survey of text mining. *Computing Reviews*, 45(9):548.

Bertram, D., Voida, A., Greenberg, S., and Walker, R. (2010). Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In *CSCW 2010: Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 291–300. ACM.

Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., and Zimmermann, T. (2007). Quality of bug reports in eclipse. In *Eclipse 2007: Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, pages 21–25, New York, NY, USA. ACM.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2008a). What makes a good bug report? In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 308–318, New York, NY, USA. ACM.

Bettenburg, N., Premraj, R., Zimmermann, T., and Kim, S. (2008b). Duplicate bug reports considered harmful really? In *ICSM 2008: Proceedings of the IEEE International Conference on Software Maintenance*, pages 337–345. IEEE.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

BugzillaFields (2010). Bugzilla fields. https://www.mediawiki.org/wiki/Bugzilla/Fields. Last visited on 16/8/2017.

BuzillaRelease (2017). Bugzilla Release notes for Bugzilla 4.0. https://www.bugzilla.org/releases/4.0/release-notes.html. Last visited on 15/5/2017.

Cavalcanti, Y. C., Da Mota Silveira Neto, P. A., de Almeida, E. S., Lucrédio, D., da Cunha, C. E. A., and de Lemos Meira, S. R. (2010). One step more to understand the bug report duplication problem. In *SBES 2010: Proceedings of the 24th Brazilian Symposium on Software Engineering*, pages 148–157. IEEE.

Cavalcanti, Y. C., Neto, P. A. d. M. S., Lucrédio, D., Vale, T., de Almeida, E. S., and de Lemos Meira, S. R. (2013). The bug report duplication problem: an exploratory study. *Software Quality Journal*, 21(1):39–66.

Chowdhury, G. (2010). *Introduction to modern information retrieval*. Facet publishing.

Costa, D. A. D., Surafel Lemma Abebe, S. M., Kulesza, U., and E.Hassan, A. (2014). An empirical study of delays in the integration of addressed issues? In *Proceedings*

of 30th International Conference on Software Maintenance and Evolution (ICSME). IEEE.

Čubranić, D. (2004). Automatic bug triage using text categorization. In *SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer.

Davidson, J. L., Mohan, N., and Jensen, C. (2011). Coping with duplicate bug reports in free/open source software projects. In *VL/HCC 2011: IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 101–108. IEEE.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Feng, L., Song, L., Sha, C., and Gong, X. (2013). Practical duplicate bug reports detection in a large web-based development community. In *Asia-Pacific Web Conference*, pages 709–720. Springer.

Gehan, E. A. (1965). A generalized Wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52(1-2):203–223.

Ghotra, B., McIntosh, S., and Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*.

Goeminne, M. and Mens, T. (2011). Evidence for the pareto principle in open source software activity. In *the Joint Porceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, pages 74–82.

Hadka, D. (2011). Beginner's Guide to the MOEA Framework: Appendix A. https://github.com/MOEAFramework/MOEAFramework/releases/download/v2.12/MOEAFramework-2.12-BeginnersGuidePreview.pdf.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36.

Hassan, A. E. (2006). Mining software repositories to assist developers and support managers. In *ICSM 2016: Proceedings of the 32nd IEEE International Conference on Software Maintenance*, pages 339–342.

Hassan, A. E. (2008). The road ahead for mining software repositories. In *FoSM 2008: Proceedings of the 2008 Frontiers of Software Maintenance*, pages 48–57. IEEE.

Hiew, L. (2006). Assisted detection of duplicate bug reports. Retrospective Theses and Dissertations, 1919-2007.

Hindle, A., Alipour, A., and Stroulia, E. (2015). A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering*, 21:1–43.

IBM (2016). Orthogonal Defect Classification. http://researcher.watson.ibm.com/researcher/view_group.php?id=480. Last visited on 5/6/2016.

Jalbert, N. and Weimer, W. (2008). Automated duplicate detection for bug tracking systems. In *DSN 2008: Proceedings of the 38th IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 52–61. IEEE.

Jeong, G., Kim, S., and Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *ESEC/FSE 2009: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 111–120. ACM.

Jiang, Y., Cukic, B., and Menzies, T. (2008). Can data transformation help in the detection of fault-prone modules? In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 16–20. ACM.

Jira (2017). Jira Duplicate Detection. https://marketplace.atlassian.com/plugins/com.deniz.jira.similarissues/server/overview. Last visited on 11/5/2017.

Joanes, D. and Gill, C. (1998). Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1):183–189.

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.-i., Adams, B., and Hassan, A. E. (2010). Revisiting common bug prediction findings using effort-aware models. In *ICSM 2010: IEEE International Conference on Software Maintenance*.

Kampstra, P. et al. (2008). Beanplot: A boxplot alternative for visual comparison of distributions.

Kanaris, I., Kanaris, K., Houvardas, I., and Stamatatos, E. (2007). Words versus character n-grams for anti-spam filtering. *International Journal on Artificial Intelligence Tools*, 16(06):1047–1067.

Kanerva, P., Kristofersson, J., and Holst, A. (2000). Random indexing of text samples for latent semantic analysis. In *CogSci 2000: Proceedings of the 22th annual conference of the cognitive science society*, volume 1036. Citeseer.

Karim, M. R., Ruhe, G., Rahman, M., Garousi, V., Zimmermann, T., et al. (2016). An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *Journal of Software: Evolution and Process*, 28(12):1025–1060.

Kaushik, N. and Tahvildari, L. (2012). A comparative study of the performance of IR models on duplicate bug detection. In *CSMR 2012: Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, pages 159–168. IEEE Computer Society.

Koponen, T. (2006). Life cycle of defects in open source software projects. In *Open Source Systems*, pages 195–200. Springer.

Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B. (2010). Predicting the severity of a reported bug. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 1–10. IEEE.

Latecki, L. J., Sobel, M., and Lakaemper, R. (2006). New em derived from kullback-leibler divergence. In *KDD 2006: Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 267–276.

Lazar, A., Ritchey, S., and Sharif, B. (2014). Improving the accuracy of duplicate bug report detection using textual similarity measures. In *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 308–311. ACM.

Lerch, J. and Mezini, M. (2013). Finding duplicates of your yet unwritten bug report. In *CSMR 2013: Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pages 69–78. IEEE.

Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, 34(4):485–496.

Liaw, A. and Wiener, M. (2016). Random Forest R package. http://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Linares-Vásquez, M., Holtzhauer, A., Bernal-Cárdenas, C., and Poshyvanyk, D. (2014). Revisiting android reuse studies in the context of code obfuscation and library usages. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 242–251. ACM.

Long, J. D., Feng, D., and Cliff, N. (2003). Ordinal analysis of behavioral data. *Handbook of psychology*.

Marie Chavent, V. K. and Benoit Liquet, J. S. (2016). Variable Clustering. https://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf. Last visited on 1/3/2017.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2015). An empirical study of

the impact of modern code review practices on software quality. *Empirical Software Engineering*, pages 1–44.

Mende, T. and Koschke, R. (2010). Effort-aware defect prediction models. In *CSMR 2010: Proceedings of the 14th European Conference on Software Maintenance and Re-engineering*, pages 107–116. IEEE.

Mitchell, M. W. (2011). Bias of the random forest out-of-bag (oob) error for certain input parameters. *Open Journal of Statistics*, 1(03):205.

Mockus, A. and Weiss, D. M. (2000). Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180.

Nagwani, N. K. and Singh, P. (2009). Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs. In *ICAC 2009: Proceedings of the International Conference on Advances in Computing, Communication and Control*, pages 202–207. ACM.

Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. (1996). *Applied linear statistical models*, volume 4. Irwin Chicago.

Nguyen, A. T., Nguyen, T. T., Nguyen, T. N., Lo, D., and Sun, C. (2012). Duplicate bug report detection with a combination of information retrieval and topic modeling. In *ASE 2012: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 70–79. ACM.

Niwattanakul, S., Singthongchai, J., Naenudorn, E., and Wanapu, S. (2013). Using of Jaccard coefficient for keywords similarity. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, page 6.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Prifti, T., Banerjee, S., and Cukic, B. (2011). Detecting bug duplicate reports through local references. In *PROMISE 2011: Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pages 8:1–8:9. ACM.

Rakha, M. S., Bezemer, C.-P., and Hassan, A. E. (2017). Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *IEEE Transactions on Software Engineering (TSE)*, PP(99):1–27.

Rakha, M. S., Shang, W., and Hassan, A. E. (2015). Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*.

Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *CIKM 2004: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, pages 42–49. ACM.

Robnik-Šikonja, M. (2004). Improving random forests. In *Machine Learning: European Conference on Machine Learning 2004*, pages 359–370. Springer.

Rocha, H., Oliveira, G. d., Marques-Neto, H., and Valente, M. T. (2015). Nextbug: a bugzilla extension for recommending similar bugs. *Journal of Software Engineering Research and Development*, 3(1):1–14.

Romano, J., Kromrey, J. D., Coraggio, J., Skowronek, J., and Devine, L. (2006). 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*.

Runeson, P., Alexandersson, M., and Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *ICSE 2007: Proceedings of the 29th International Conference on Software Engineering*, pages 499–510. IEEE Computer Society.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Šarić, F., Glavaš, G., Karan, M., Šnajder, J., and Bašić, B. D. (2012). Takelab: Systems for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 441–448.

Scott, A. and Knott, M. (1974). A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, pages 507–512.

Shah, C. and Croft, W. B. (2004). Evaluating high accuracy retrieval techniques. In *SIGIR 2004: Proceedings of the 27th annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2–9. ACM.

Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., and Matsumoto, K.-i. (2013). Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042.

Singh, R., Bezemer, C.-P., Shang, W., and Hassan, A. E. (2016). Optimizing the performance-related configurations of object-relational mapping frameworks using a multi-objective genetic algorithm. In *In SPEC: Proceedings of the 7th ACM International Conference on Performance Engineering*, pages 309–320. ACM.

Somasundaram, K. and Murphy, G. C. (2012). Automatic categorization of bug reports using Latent Dirichlet Allocation. In *Proceedings of the 5th India Software Engineering Conference (ISEC)*, pages 125–130. ACM.

Steinwart, I. and Christmann, A. (2008). *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition.

Sun, C., Lo, D., Khoo, S.-C., and Jiang, J. (2011). Towards more accurate retrieval of duplicate bug reports. In *ASE 2011: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262. IEEE.

Sun, C., Lo, D., Wang, X., Jiang, J., and Khoo, S.-C. (2010). A discriminative model approach for accurate duplicate bug report retrieval. In *ICSE 2010: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 45–54. ACM.

Sureka, A. and Jalote, P. (2010). Detecting duplicate bug report using character n-gram-based features. In *APSEC 2010: Proceedings of the 2010 Asia Pacific Software Engineering Conference*, pages 366–374. IEEE Computer Society.

Syer, M. D., Nagappan, M., Hassan, A. E., and Adams, B. (2013). Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular

open-source android apps. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, pages 283–297. IBM Corp.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., Matsumoto, K.-i., Ghotra, B., Kamei, Y., Adams, B., Morales, R., Khomh, F., et al. (2015). The impact of mislabelling on the performance and interpretation of defect prediction models. In *ICSE 2015: Proceedings of the 37th International Conference on Software Engineering*.

Taylor, M., Zaragoza, H., Craswell, N., Robertson, S., and Burges, C. (2006). Optimisation methods for ranking functions with multiple parameters. In *CIKM 2006: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 585–593. ACM.

Thongtanunam, P., McIntosh, S., Hassan, A. E., and Iida, H. (2016). Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 1039–1050. IEEE.

Thung, F., Kochhar, P. S., and Lo, D. (2014). Dupfinder: Integrated tool support for duplicate bug report detection. In *ASE 2014: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 871–874. ACM.

Thung, F., Lo, D., and Jiang, L. (2012). Automatic defect categorization. In *WCRE 2012: Proceedings of the 19th Working Conference on Reverse Engineering*, pages 205–214. IEEE.

Trotman, A. (2005). Learning to rank. *Information Retrieval*, 8(3):359–381.

Valdivia Garcia, H. and Shihab, E. (2014). Characterizing and predicting blocking bugs in open source projects. In *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 72–81. ACM.

Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE 2008: Proceedings of the 30th International Conference on Software Engineering*, pages 461–470. ACM.

Xavier Robin, N. T., Alexandre Hainard, N. T., FrÃl'dÃl'rique Lisacek, J. S., and MÃijller, M. (2016). pROC R package. http://cran.r-project.org/web/packages/pROC/pROC.pdf. Last visited on 21/3/2017.

Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E., and Ubayashi, N. (2015). Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution*, pages 46–55. ACM.

Zar, J. H. (1972). Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association*, 67(339):578–580.

Zhou, J. and Zhang, H. (2012a). Learning to rank duplicate bug reports. In *Proceedings of the 21st International Conference on Information and Knowledge Management (CIKM)*, pages 852–861, New York, NY, USA. ACM.

Zhou, J. and Zhang, H. (2012b). Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*, pages 852–861. ACM.

Zimmermann, T., Nagappan, N., Guo, P. J., and Murphy, B. (2012). Characterizing and predicting which bugs get reopened. In *ICSE 2012: Proceedings of the 2012 International Conference on Software Engineering*, pages 1074–1083. IEEE Press.

Zou, J., Xu, L., Yang, M., Zhang, X., Zeng, J., and Hirokawa, S. (2016). Automated duplicate bug report detection using multi-factor analysis. *IEICE Transactions on Information and Systems*, E99.D(7):1762–1775.