ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/319878728

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

Article *in* IEEE Transactions on Software Engineering · December 2017

DOI: 10.1109/TSE.2017.2755005

CITATIONS
17

READS
180

3 authors:

Mohamed Sami Rakha
Queen's University
15 PUBLICATIONS **88** CITATIONS

SEE PROFILE

Cor-Paul Bezemer
University of Alberta
68 PUBLICATIONS **864** CITATIONS

SEE PROFILE

Ahmed E. Hassan
Queen's University
406 PUBLICATIONS **11,672** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Bounties in Open Source Development on GitHub: A Case Study of Bountysource Bounties View project

Project Mining Online Gaming Stores View project

All content following this page was uploaded by Ahmed E. Hassan on 23 August 2018.

The user has requested enhancement of the downloaded file.

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

Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E. Hassan

Abstract— Issue tracking systems (ITSs), such as Bugzilla, are commonly used to track reported bugs, improvements and change requests for a software project. 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. Several automated approaches have been proposed for retrieving duplicate reports, i.e., identifying the duplicate of a new issue report in a list of n candidates. These approaches rely on leveraging the textual, categorical, and contextual information in previously-reported issues to decide whether a newly-reported issue has previously been reported. In general, these approaches are evaluated using data that spans a relatively short period of time (i.e., the classical evaluation). However, in this paper, we show that the classical evaluation tends to overestimate the performance of automated approaches for retrieving duplicate issue reports. Instead, we propose a realistic evaluation using all the reports that are available in the ITS of a software project. We conduct experiments in which we evaluate two popular approaches for retrieving duplicate issues (BM25F and REP) using the classical and realistic evaluations. We find that for the issue tracking data of the Mozilla foundation, the Eclipse foundation and OpenOffice, the realistic evaluation shows that previously proposed approaches perform considerably lower than previously reported using the classical evaluation. As a result, we 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, we propose to leverage the resolution field of issue reports. Our 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.

Index Terms—Text analysis, software engineering, performance evaluation

1 Introduction

An issue tracking system (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 [8]. For example, users can receive emails when issue reports are resolved or assigned. Furthermore, they can discuss the issues with team members directly. An issue can represent a bug, a new feature, or a change request. Thus, many software projects rely on issue tracking systems for managing maintenance activities [17].

The identification of duplicately-reported issues is necessary since all too often, the same software issue might be reported more than once using a different vocabulary or software use cases. In popular software projects, such as Mozilla Firefox and the Eclipse IDE, the number of duplicate issue reports can be as large as 20-30% [3] of all reported issues. Manual identification of such duplicates requires a significant investment in effort and time [3, 9, 17, 25]. However, early identification of duplicate reports is important as it prevents multiple developers from wastefully working on the same software issues.

Several prior studies have proposed approaches to automatically identify duplicate issue reports [1, 2, 16, 17, 22, 28, 30, 34] which help the triagers [4] to decide whether a newly reported issue is a duplicate of a previously-reported issue. These approaches use the textual, categorical and contextual features of each reported issue to find the most similar reports. Prior research on automatically identifying duplicate issue reports can be roughly divided into two subdomains. The first subdomain focuses on classifying whether a newly-reported issue was already previously-reported [6, 15]. The second subdomain focuses on retrieving existing issue reports that are the possible duplicates of a newly-reported issue. In this paper, we focus on the retrieval of duplicate issue reports, hence, the second subdomain.

In general, many of these approaches are evaluated on a small subset of the reports that are available in an ITS [1, 16, 20, 22, 30, 31, 36]. For example, Sun et al. [31] 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 we call the *classical evaluation* throughout this paper, the assumption is that duplicately-reported issues are reported shortly after the original report. However, as we show in this paper, this assumption is incorrect in many cases.

Instead, we 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 paper, we conduct an experiment using data from the ITSs of the Mozilla foundation, the Eclipse foundation and

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

OpenOffice in which we show that the classical evaluation relatively overestimates the performance of BM25F [26] and REP [30], two popular approaches for retrieving duplicate issue reports, by a median of 17-42%. Our results show that our 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 our paper, we evaluate two approaches for improving the performance of the BM25F and REP approaches. First, we 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, we propose an approach that extends BM25F and REP with the use of the resolution field of issue reports. By using this field, we 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. Our 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%. We use the recall rate and mean average precision (MAP) as performance metrics [16, 22, 31].

In particular, in this paper we 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, we study the differences between the realistic and classical evaluations over the lifetime of each studied ITS. We 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 approaches according to the realistic evaluation?
In RQ1, we 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, we study whether it is possible to use a time-based threshold to ignore some of the long-term duplicates without impacting the performance. We 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, we 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 report impact the performance according to the realistic evaluation?
In RQ3, we build upon our 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. Our results show that we can leverage the resolution field of issue reports to improve the automated retrieval of duplicates. Our proposed enhancement improves the performance of the automated retrieval of duplicates by a median of 7-21.5% and a maximum of 19-60%.

Our work is the first to explore the design of the performance evaluation of automated approaches for retrieving duplicate issue reports. We expect and hope that future work will dig even deeper into the concept of realistic evaluation (while defining it better).

The rest of the paper is organized as follows. Section 2 presents background information and discusses related prior work. Section 3 presents our experimental setup. Section 4 describes an exploratory study that motivates our work. Section 5 presents the results of our experiments. Section 6 discusses threats to the validity of our study and Section 7 concludes the paper.

2 Background

2.1 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 a discussion of developers. 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 the 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 [22, 28, 30, 31].

2.2 Automated Retrieval of Duplicate Issue Reports

To assist triagers with the tedious process of retrieving duplicate issue reports, automated approaches have been previously proposed [1, 2, 16, 17, 22, 28, 30, 34]). Most of these approaches use information retrieval techniques [5]. 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 (e.g., as in the vector space model (VSM) [29]). 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) [10] 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:

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

Where $tf(t,d)$ is the frequency of the term t in a document d, while $idf(t,D)$ is the total number of documents in the corpus divided by the number of documents that have the term t. The TF-IDF vector of a document is the vector containing the TF-IDF statistic of all terms in that document. Every document in the corpus has a TF-IDF vector, allowing documents to be compared using the distance between their vectors. Several approaches are used to measure the distance between two TF-IDF vectors, such as the cosine similarity [10], Dice similarity [10], BLEU similarity [24] and Jaccard similarity [23]. In this paper, we perform our experiments using two popular approaches for retrieving duplicate issue reports that are based on TF-IDF: BM25F and REP [22, 30, 36].

BM25F [26] is an advanced document similarity approach that is based on the TF-IDF vectors of documents [7, 26]. 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 we refer throughout this paper as the *tuning data* for BM25F.

REP [30] 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:

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

Where 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 [33], which uses a set of duplicate reports to find these weights. We refer throughout this paper to that set of duplicate reports as the *tuning data* for REP. The $features_i$ variable holds the feature value for each of the textual and categorical fields. For each newly-reported issue, the REP function is used to retrieve a list of possible master issue reports (*candidates*). Table 1 shows an example of a REP ranking for a newly-reported duplicate issue #90892 in OpenOffice. 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.

TABLE 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.

2.3 Performance Evaluation of Automated Retrieval of Duplicate Issue Reports

Prior research (see Table 2) 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. We refer to this evaluation process that is widely used by prior research as the *classical evaluation*. Figure 2a illustrates how the classical evaluation process works. For each evaluated time period, the following steps are applied:

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

TABLE 2: 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. [1]	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Hindle et al. [16]	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010, Android 2007-2012
Jalbert et al. [17]	**Mozilla Feb-Oct 2005**
Lazar et al. [20]	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Nguyen et al. [22]	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010
Sun et al. [31]	Mozilla before June 2007, **Eclipse 2008**, **OpenOffice 2008**
Sun et al. [30]	**Mozilla 2010**, **Eclipse 2008**, OpenOffice 2008-2010, Eclipse 2001-2007
Zhou et al. [35]	**Eclipse 2008**
Zou et al. [36]	**Eclipse 2008**

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 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 identifying short-term duplicate issue reports only. However, as we will show in this paper, this assumption does not hold, leading to a relative overestimation of performance. But first, we discuss related work.

2.4 Related Work

Research on automatically identifying duplicate issue reports can be roughly divided into two subdomains. The first subdomain focuses on determining whether a newly-reported issue was already reported before. Hence, work in this subdomain focuses on identifying 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 [25].

In this paper, we focus on revisiting the performance evaluation that was used in prior work in the second subdomain. First, we discuss prior work in both subdomains.

2.4.1 Determining Whether a Newly-Reported Issue was Already Reported Before

Hiew et al. [15] proposed an approach that labels a newly-reported issue as 'duplicate' or 'unique'. Their approach extracts a TF-IDF vector from the summary and description field of the new report, and groups similar reports together in so-called 'centroids'. Based on a threshold for the similarity of the vectors within a centroid, the new issue report is labeled as a duplicate or unique report.

Feng et al. [12] proposed an approach to predict whether two issue reports are duplicates of each other. Their approach uses a classifier (such as Naive Bayes, Decision Tree or SVM) to predict 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 intuition behind using this data is that (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 new issue report.

Banerjee et al. [6] 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.

2.4.2 Retrieving Duplicate Issue Reports

Prior research has proposed several approaches for the retrieval of duplicate issue reports. Below, we summarize the prior work based on the type of extracted data that is used in the retrieval.

Textual Similarity-Based Approaches

The textual fields in issue reports are the summary and description field [3]. Runeson et al. [28] propose a natural language processing approach to automatically rank duplicate candidates. The issue reports are considered as text documents. The textual contents of the issue reports are preprocessed (i.e., using tokenization, stemming and stop word removal) and then a vector of term (word) frequencies (TF) is calculated for each issue report. The distances between the frequency vector of each newly-reported issue to the frequency vectors of the previously-reported issues are calculated using the cosine, Dice and Jaccard similarity [23, 24]. Wang et al. [34] extend Runeson et al.'s work [28] by not only considering TF, but also IDF. In addition, they consider execution traces to retrieve duplicate issue reports.

Sureka et al. [32] use character-based features to measure the text similarity between the issue reports. Using character-based features has several advantages, such as natural-language independence.

Textual and Categorical Similarity-Based Approaches

In addition to textual fields, an issue report contains

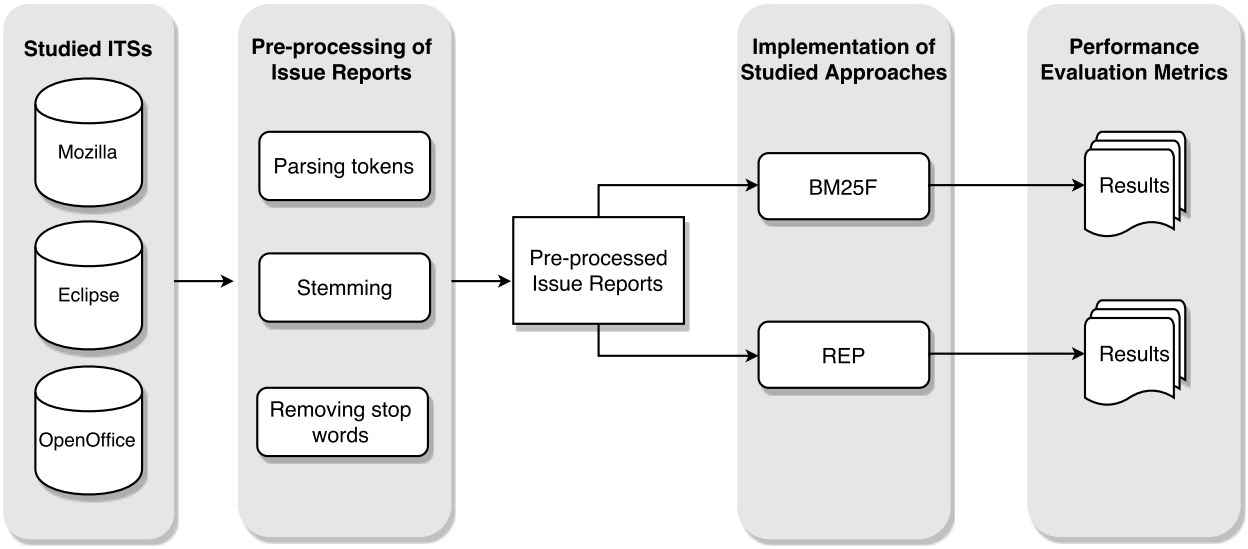


Fig. 1: Overview of the experimental setup.

categorical fields such as the component, version, and platform. Several approaches [17, 19, 30] have demonstrated that categorical fields can be useful for improving the performance of the automated retrieval of duplicate issue reports. The most important approach of this type is REP [30] which is explained in more detail in Section 2.2.

Topic-Based Approaches
Topic-based approaches extract topics from the textual fields of issue reports using an automated approach, such as LDA, and use these topics to calculate the similarity between reports. Nguyen et al. [22] propose a topic-based approach that extends BM25F using machine learning. Alipour et al. [2], Lazar et al. [20] and Aggarwal et al. [1] extend the REP approach with topic-based features.

In this paper, we use two approaches from the first two aforementioned groups in our experiments. We apply the BM25F [26] approach as an example for the textual similarity-based approaches, and the REP [30] approach as an example for the textual and categorical similarity-based approaches. We use these approaches because of their popularity and reproducibility due to their available implementations, which allows us to compare our results with prior work. We 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 paper form the foundation of the topic-based approaches which increases the applicability of the findings in this paper to all types of duplicate retrieval approaches in literature nowadays.

3 Experimental Setup

In this section, we present the studied ITSs, the process of preparing the data that is used for our experiments and the used performance evaluation metrics. Figure 1 presents an overview of the experimental setup.

TABLE 3: 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

3.1 Studied ITSs

In this paper, we 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, we 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 paper, we collect the XML file and historical information of each issue report in the three studied ITSs. We collect the XML for each report using a simple crawler that increments the id parameter of the XML URL.[2]

We focus on the same periods (see Table 3) as used by prior research [1, 16, 20, 22, 30]. Using the same data periods makes it possible to compare our results with prior research.

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

Table 3 presents the number of issue reports and duplicate reports for each studied ITS.[3]

To determine the ground truth of whether an issue report is a duplicate, we 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 we recently crawled (i.e., in December 2015) relatively old data (i.e., reports that were submitted before 2011), we can safely assume that the labels are correct and stable.

3.2 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 [1, 16, 22, 30]. These steps are:

- *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 paper, we use the following fields of an issue report: issueID, description, summary, component, priority, type, version and report date. We follow a bucket structure [30] where all the duplicate reports are placed inside the same bucket in which the earliest reported issue report is called the *master* report. We use the same text preprocessing implementation used by Sun et al. [30] and Nguyen et al. [22].

3.3 Implementation of the Experiments

In this paper, we use the BM25F and REP implementations that are provided by Sun et al[4]. In addition, we made the implementation and results of our experiments available as a replication package[5].

3.4 Performance Evaluation Metrics

Prior work on automatically retrieving duplicate issue reports [16, 22, 30, 31] uses the recall rate and the mean average precision (MAP) to evaluate the performance of the proposed approaches. These performance metrics are calculated for the newly-reported duplicate issues only, as there are no correct candidate reports for non-duplicates. Note that this is a limitation of all approaches for retrieving duplicate issue reports. The recall rate is defined as:

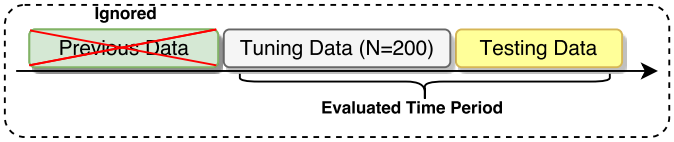
$$\text{Recall}_{\text{topN}}=\frac{retrieved_{topN}}{retrieved_{topN}+missed_{topN}}$$

$retrieved_{topN}$ is the number of duplicate reports that successfully had their master reports retrieved in the $topN$ ranked list of candidates. The $missed_{topN}$ is the number of duplicate reports that did not have their master reports retrieved in the $topN$ candidate list. The larger the list of master report candidates, the higher the recall rate. In this paper, we choose N=5 and N=10, where N represents the candidate list size. As suggested by Runeson et al. [28], the small size of the list (i.e., N=5 and N=10) results in an acceptable visualization space and prevents triagers of having to investigate a large number of duplicate candidates.

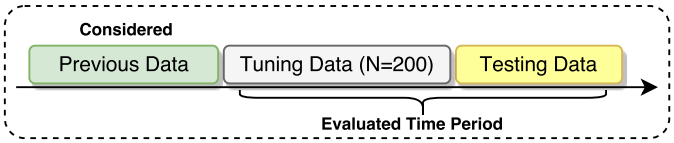
The MAP indicates in which rank the correctly retrieved duplicate candidate can be found in the returned list of candidates. The MAP is measured as follows:

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

where Q is the total number of accurately-found duplicate candidates, while *rank* is the position of the master report in the list. A MAP value close to 1 means that the accurate candidates are appearing on the first rank of the returned list all the time. In this paper, we limit the maximum size of the list to 1,000 for computational reasons. Similar to prior work, we use the recall rate and MAP metrics to evaluate the performance of the automated approaches for retrieving duplicates.



(a) Classical Evaluation



(b) Realistic Evaluation

Fig. 2: Classical evaluation versus realistic evaluation.

4 Exploratory Study

As explained in Section 2.3, the classical evaluation ignores a significant portion of the issue reports in an ITS when retrieving duplicate candidates. In this section, we first propose a more realistic evaluation, which does not ignore issue reports. Second, we 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. We compare the evaluation that is conducted in prior work (i.e., the classical evaluation) with a realistic evaluation using the same issue reports. Finally, we conduct an analysis of the sensitivity of the choice of the tuning data.

4.1 Realistic Evaluation

We propose to use an evaluation in which no data is ignored. We refer to this type of evaluation as the *realistic evaluation*. We chose this name because this type of evaluation closely resembles the way in which an ITS works in practice.

[3]Note that these numbers are higher than reported in prior work. In the remainder of this paper we will show that the numbers that are reported in prior work are incorrect. Hence, we present the correct numbers in Table 3.

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

[5]http://sailhome.cs.queensu.ca/replication/realistic_vs_classical/

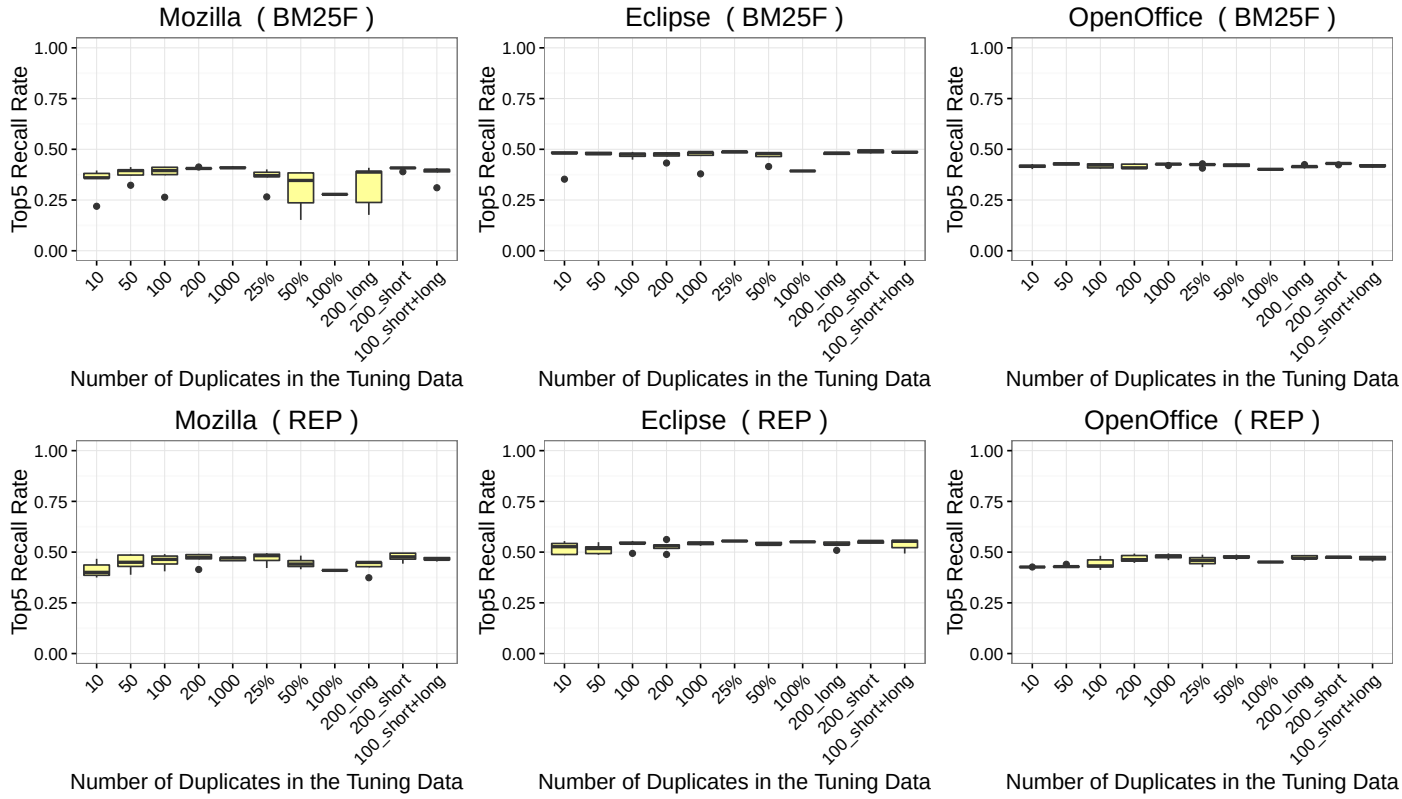


Fig. 3: The $\text{Recall}_{\text{top5}}$ of the BM25F and REP approaches (as obtained by the realistic evaluation) after tuning with different tuning data sizes for 5 runs per tuning data size.

TABLE 4: 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	Does not occur (D cannot be reported before M)	

Figure 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 4 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, we present our exploratory study on the differences between the classical and realistic evaluation.

TABLE 5: The number of considered duplicate issue reports for both types of evaluation for the studied ITSs in the exploratory study.

	# of duplicates (classical)	(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%

4.2 Classical vs. Realistic Evaluation Processes

We compare the classical and realistic evaluation processes for the time periods that are evaluated by prior work [1, 16, 20, 22, 30, 31, 36]. We 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

TABLE 6: 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%

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 [31]. 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 shows the difference in the number of considered duplicates for both types of evaluation. We observe that up to 23.5% of the duplicate reports are ignored when we 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 identify without benefiting from the proposed automated approaches.

> **Observation 1:** 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, we 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 Recall$_{top5}$, Recall$_{top10}$ and MAP of the automated approaches using the classical and realistic evaluation.

Table 6 presents the absolute and relative difference in performance metrics for both types of evaluation processes for the BM25F and REP approaches. We define the relative overestimation of the classical evaluation process of a performance metric as:

$$Overestimation_{\text{relative}} = (\frac{metric_{classical}}{metric_{realistic}} - 1) * 100 \quad (3)$$

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 6 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 Recall$_{top5}$, Recall$_{top10}$ and MAP, respectively. For the REP approach, the relative overestimation is up to 39.8%, 32.9% and 32.1% for Recall$_{top5}$, Recall$_{top10}$ and MAP, respectively.

> **Observation 2:** For the BM25F approach, the classical evaluation relatively overestimates the performance metrics with up to 85.7%, 74.2% and 74.5% for the Recall$_{top5}$, Recall$_{top10}$ 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 Recall$_{top5}$, Recall$_{top10}$ and MAP.

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

4.3 The Sensitivity of the Choice of the Tuning Data

As explained in Section 2, both REP and BM25F have parameters that are tuned to find their optimal values [26, 30]. The BM25F approach has 10 parameters while the REP approach has 19 parameters. In prior work on retrieving duplicate issue reports [1, 16, 20, 22, 30, 31], the number of the duplicate issue reports in the tuning data set was always fixed to $N = 200$. In this section, we do a preliminary investigation of whether the choice of N impacts the performance of the automated approaches for retrieving duplicate issue reports. We 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, we 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. We selected short-term duplicates as duplicates that were reported within six months, and we selected long-term duplicates as duplicates that were reported at least two years apart. We take the following steps to conduct our 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 time periods that are shown by Table 3.

We 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 3 shows the results of our investigation.

We 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 paper, we used the exact same tuning data for our runs of the realistic and classical evaluation. Hence, our results are not affected by the choice of tuning data. However, our 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 our proposed realistic evaluation.

5 Experimental Results

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

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

Motivation. In Section 4, we showed that there is a difference between the classical and realistic evaluation for the evaluated time periods that are used in prior work [1, 16, 20, 22, 30, 31, 36]. However, the observations for the evaluated time periods may not generalize to data from other periods. Therefore, we study whether the observations in the exploratory study generalize to all the available periods in the studied ITSs.

Approach. For each studied ITS, we randomly select 100 chunks of data. Each chunk of data follows the same structure as presented previously in Figure 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. We 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 3 are only used in our 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, we 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. We 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 [13]. We 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.

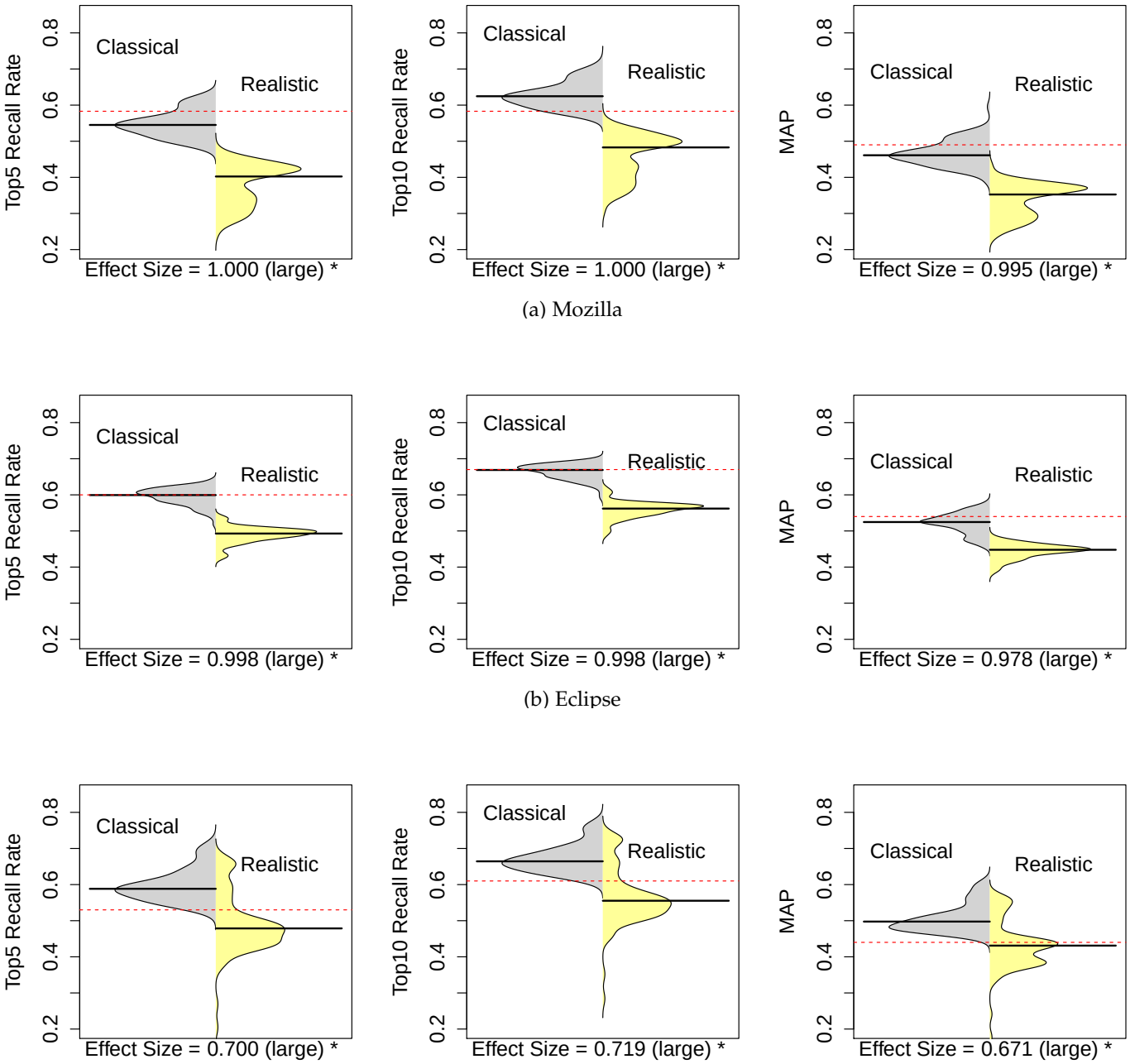
We reject H_0 and accept H_1, when $p < 0.01$. In addition, we calculate the *effect size*, as the effect size quantifies the difference between the two distributions. We use *Cliff's Delta* as it does not require the normality assumption of a distribution to quantify the effect size [21]. The following thresholds are used for the *Cliff's Delta (d)* [27]:

$$\begin{cases} \text{trivial} & \text{for } |d| \leq 0.147 \\ \text{small} & \text{for } 0.147 < |d| \leq 0.33 \\ \text{medium} & \text{for } 0.33 < |d| \leq 0.474 \\ \text{large} & \text{for } 0.474 < |d| \leq 1 \end{cases} \tag{4}$$

In addition, we calculate the relative overestimation (see Section 4) and kurtosis [18] 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.

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 4 and 5 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 6 shows the distribution of the relative overestimation of the classical evaluation for the performance metrics for all the 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 our 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 our proposed realistic evaluation.

The yielded performance varies greatly for both evaluation types. Figures 4 and 5 show that the yielded performance metrics for both evaluation types show a large variation for all the studies ITSs. Table 7 shows the kurtosis for the observed 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 (b) shows that the variations is still considerably large.



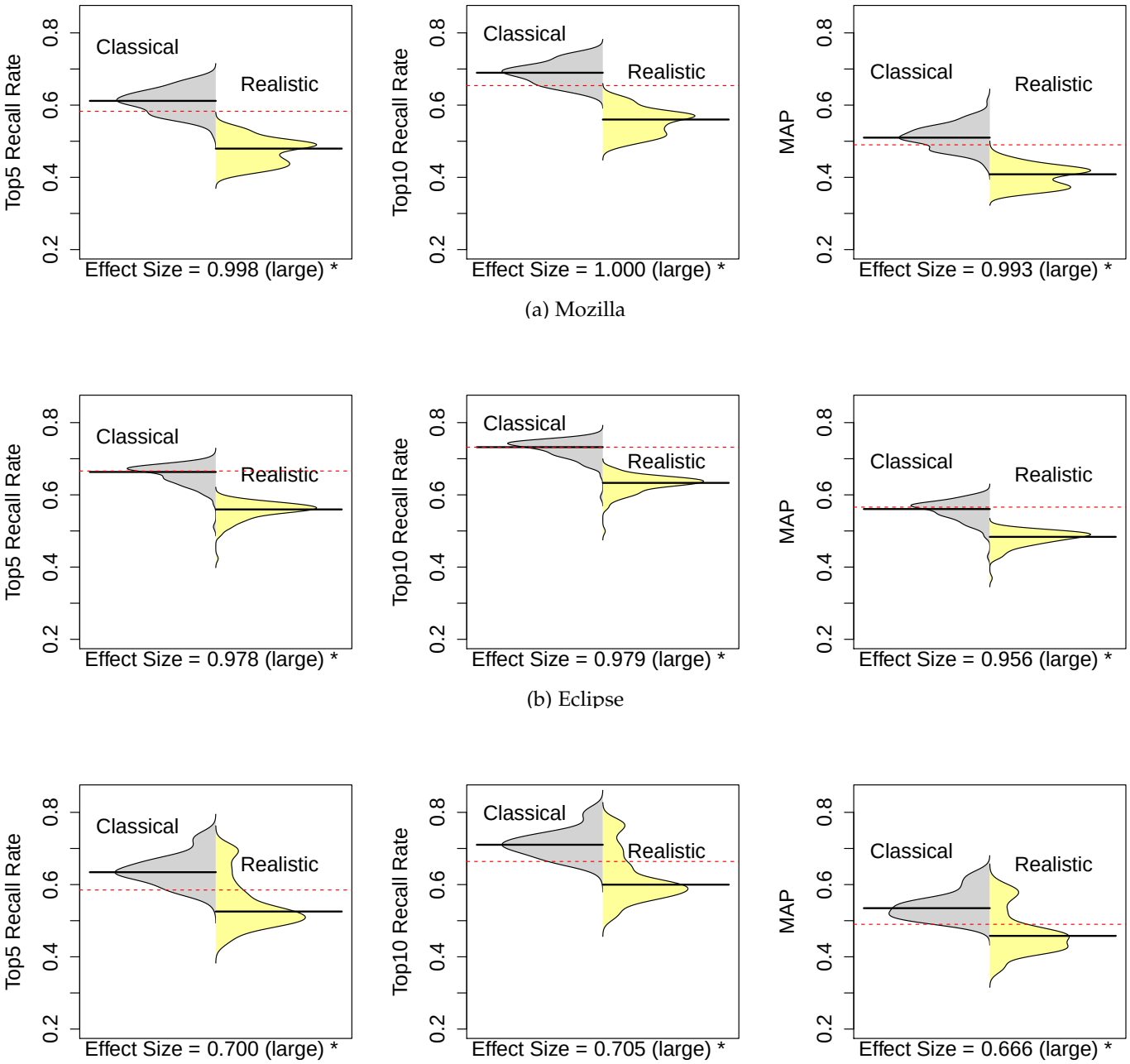
(a) Mozilla

(b) Eclipse

(c) OpenOffice

$^{*}\, p < 0.001$

Fig. 4: 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 4.1.



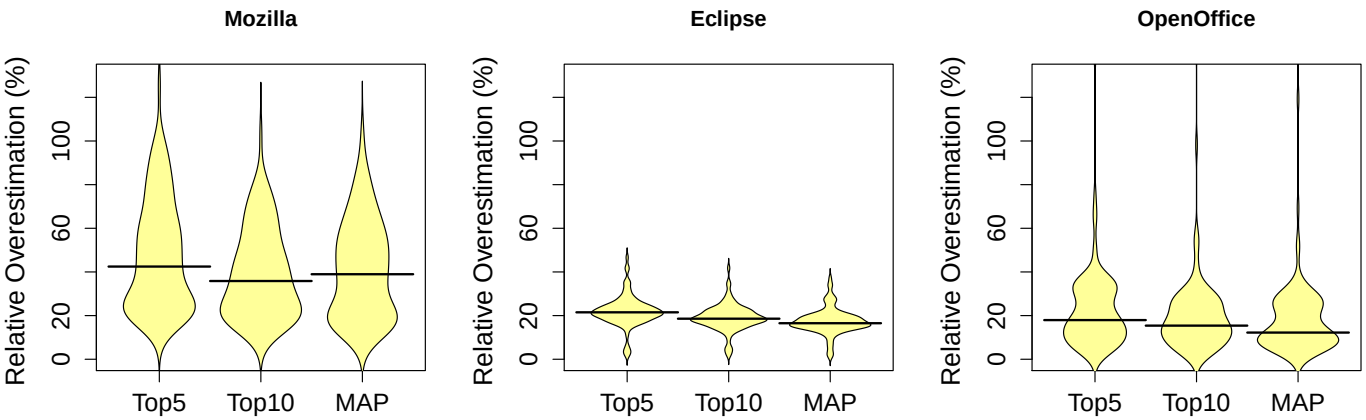
(a) Mozilla

(b) Eclipse

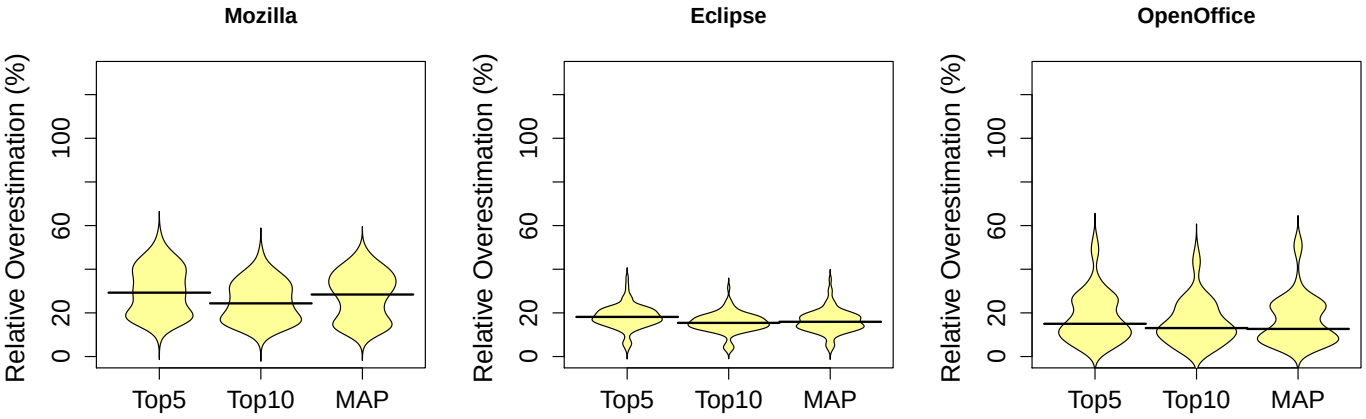
(c) OpenOffice

$^{*}\, p < 0.001$

Fig. 5: 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 4.1.



(a) BM25F



(b) REP

Fig. 6: Performance relative overestimation by the classical evaluation.

TABLE 7: 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**

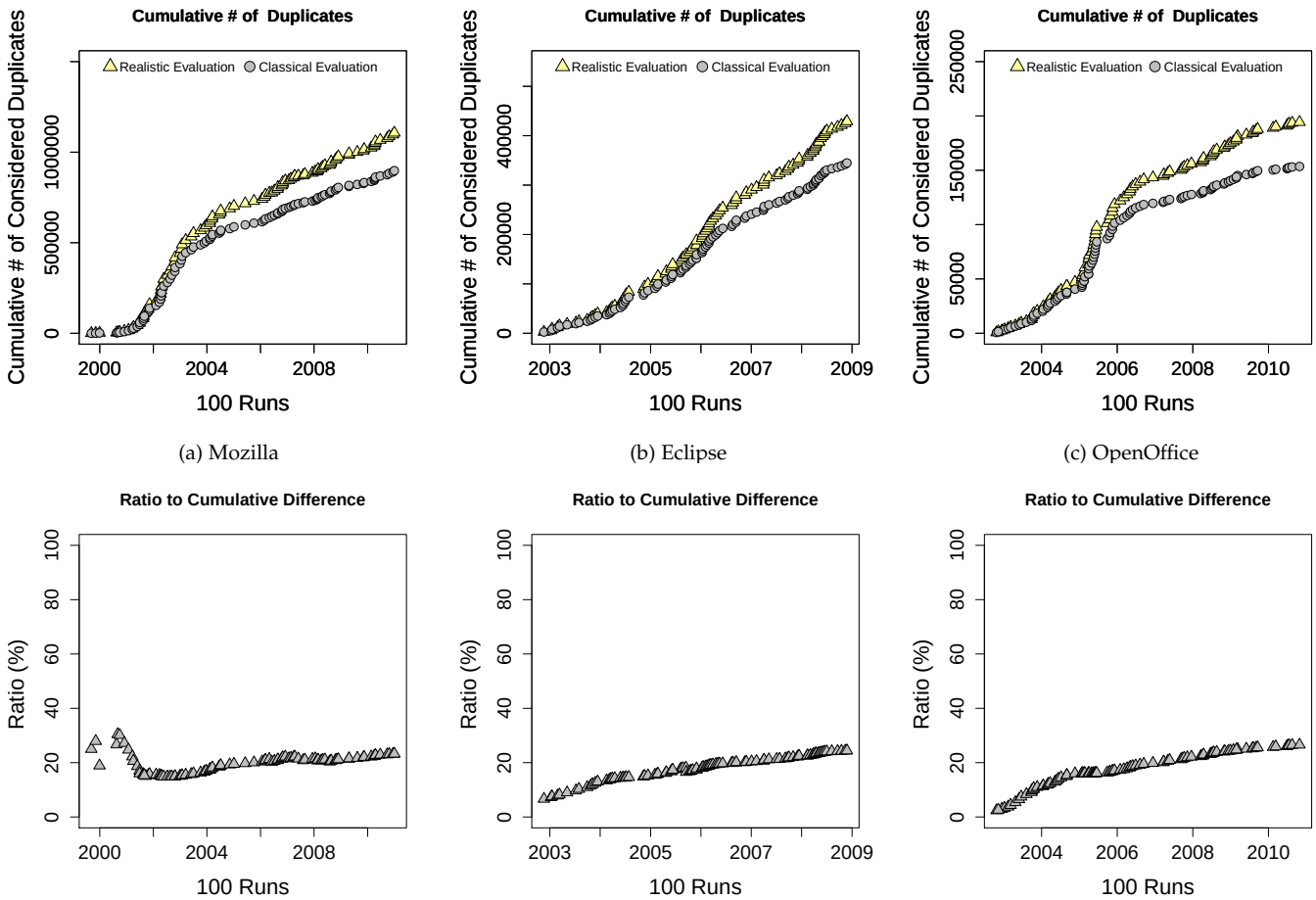
The values that are highlighted in **bold** indicate a distribution that is more varied than the normal distribution (i.e., kurtosis < 3).

Figure 4 and 5 show a dotted red line for the classical evaluation performance for the tested year periods by the prior work (see Section 4).

As clearly demonstrated by the figures and Table 7, 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 7 shows the number of duplicates that are considered by the classical and realistic evaluations. In the early stages of a project, 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



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

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

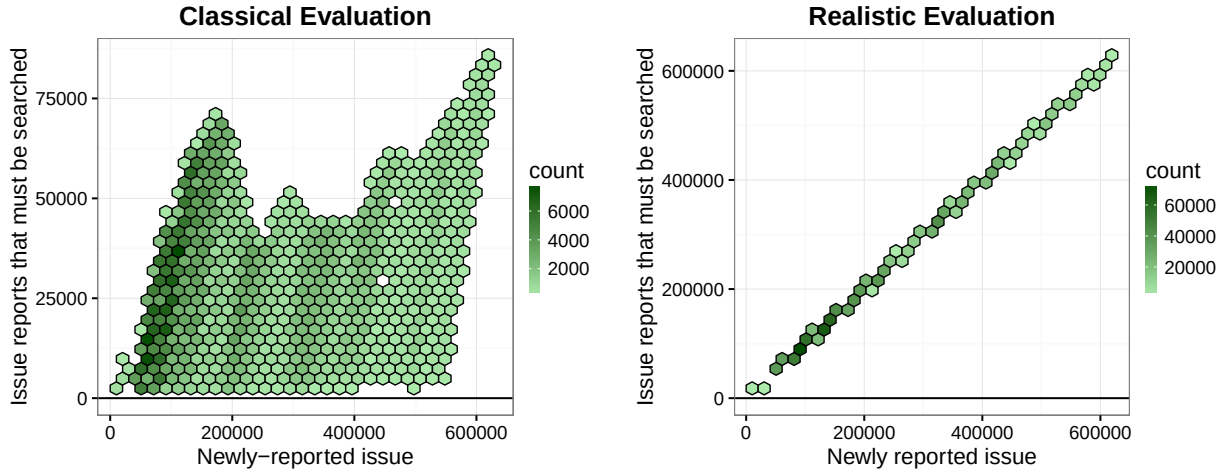
Fig. 7: The number of considered duplicates issues over the evaluated time periods (100 runs).

issue reports is negatively impacted by the number of issue reports that must be searched. Figure 8 shows a hexagon bin plot[6] 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 overplotting 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).

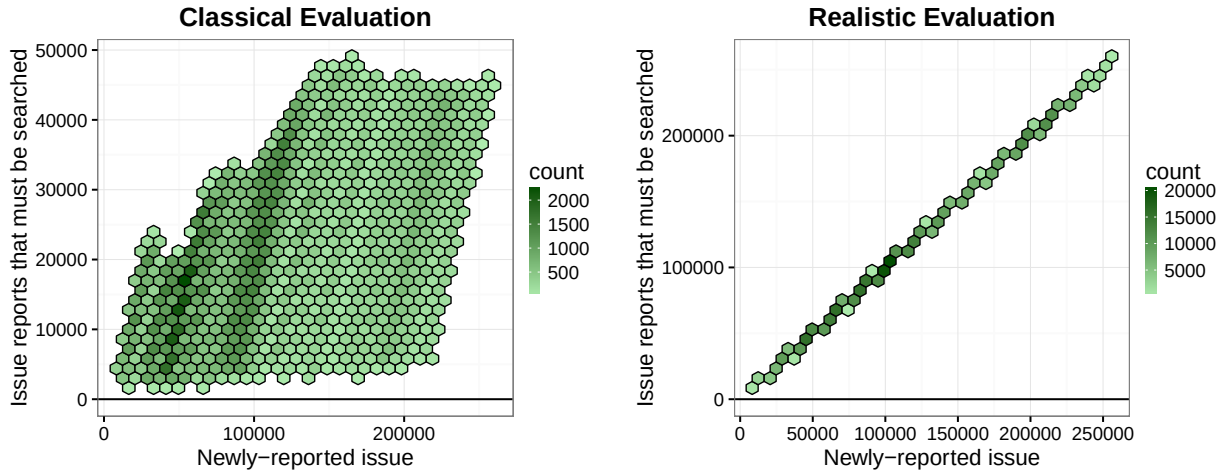
We observe that the realistic evaluation searches each previously-reported issue in the ITS to retrieve a duplicate. However, the number of issue reports that must be searched has a negative impact on the performance. Figure 9 shows the $\text{Recall}_{\text{top5}}$ over time for the BM25F and REP approaches. 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, we omit those figures from this paper. 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 a project 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.

Observation 3: 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.

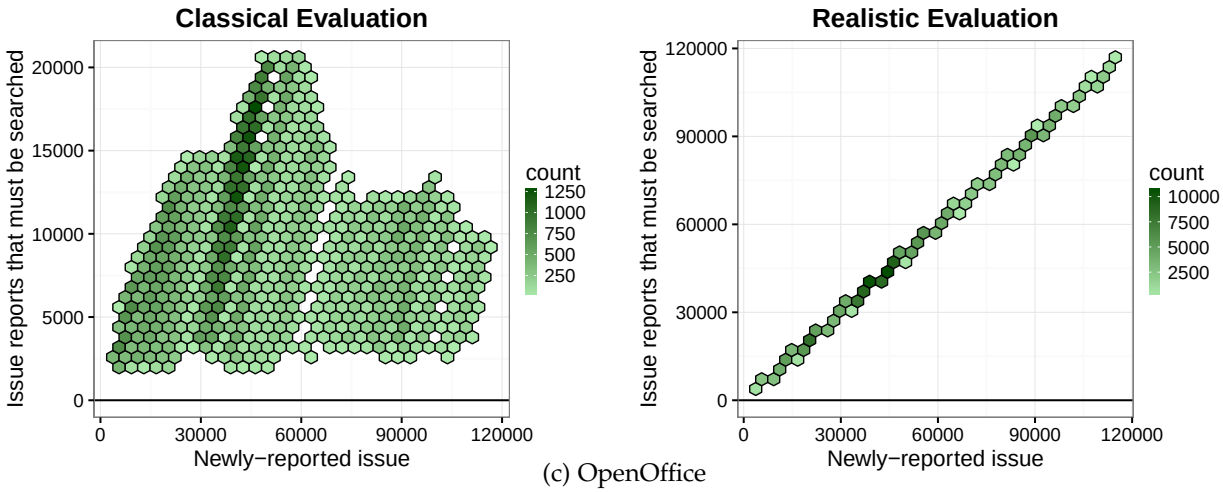
[6]Hexbin package: https://cran.r-project.org/web/packages/ggplot2/index.html



(a) Mozilla

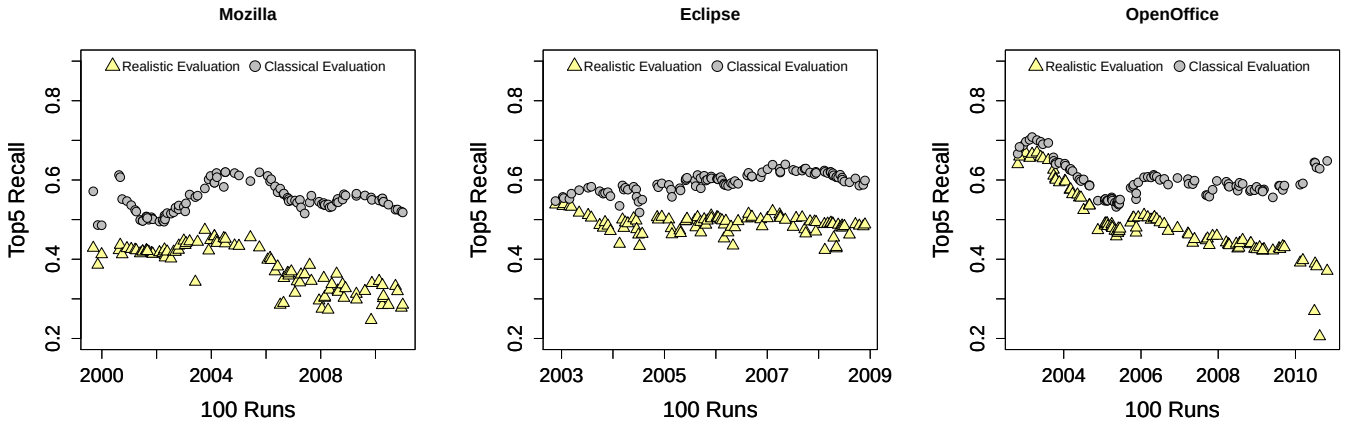


(b) Eclipse

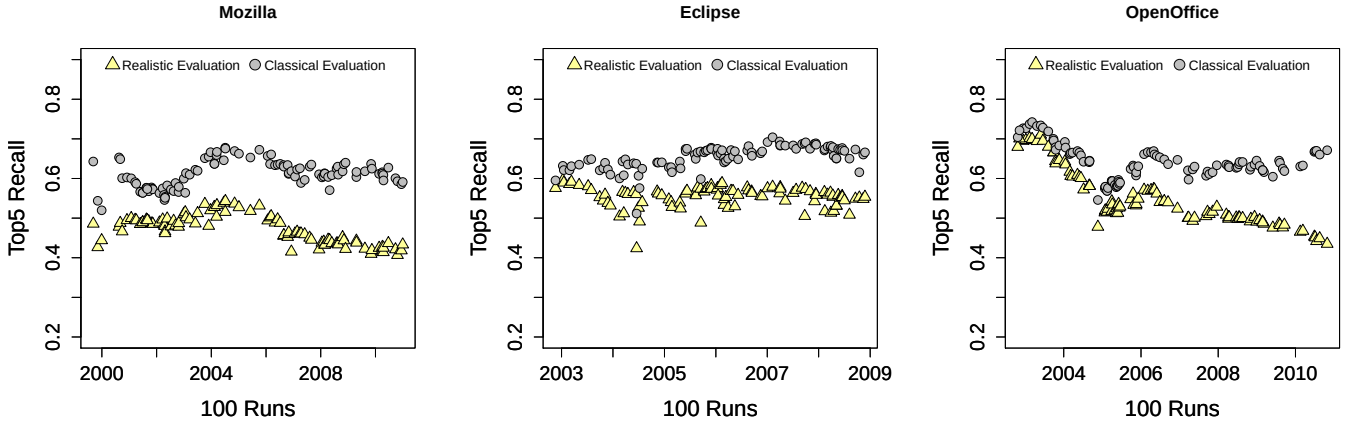


(c) OpenOffice

Fig. 8: The number of issue reports that must be searched for each newly-reported issue.

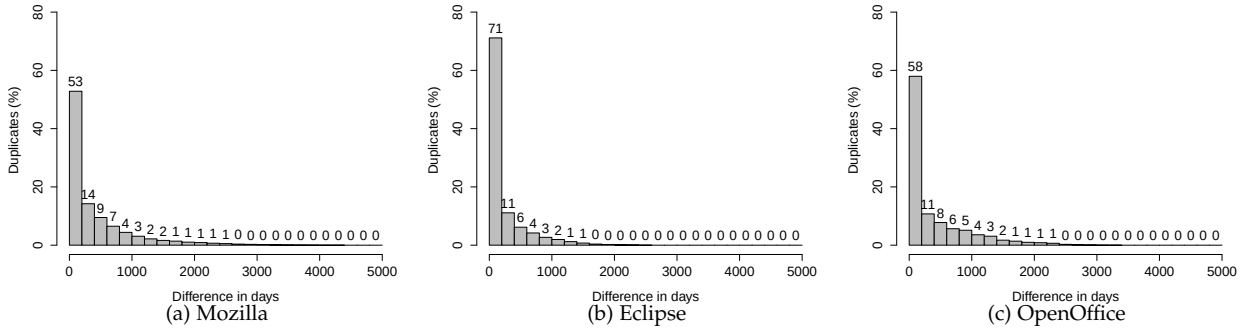


(a) BM25F



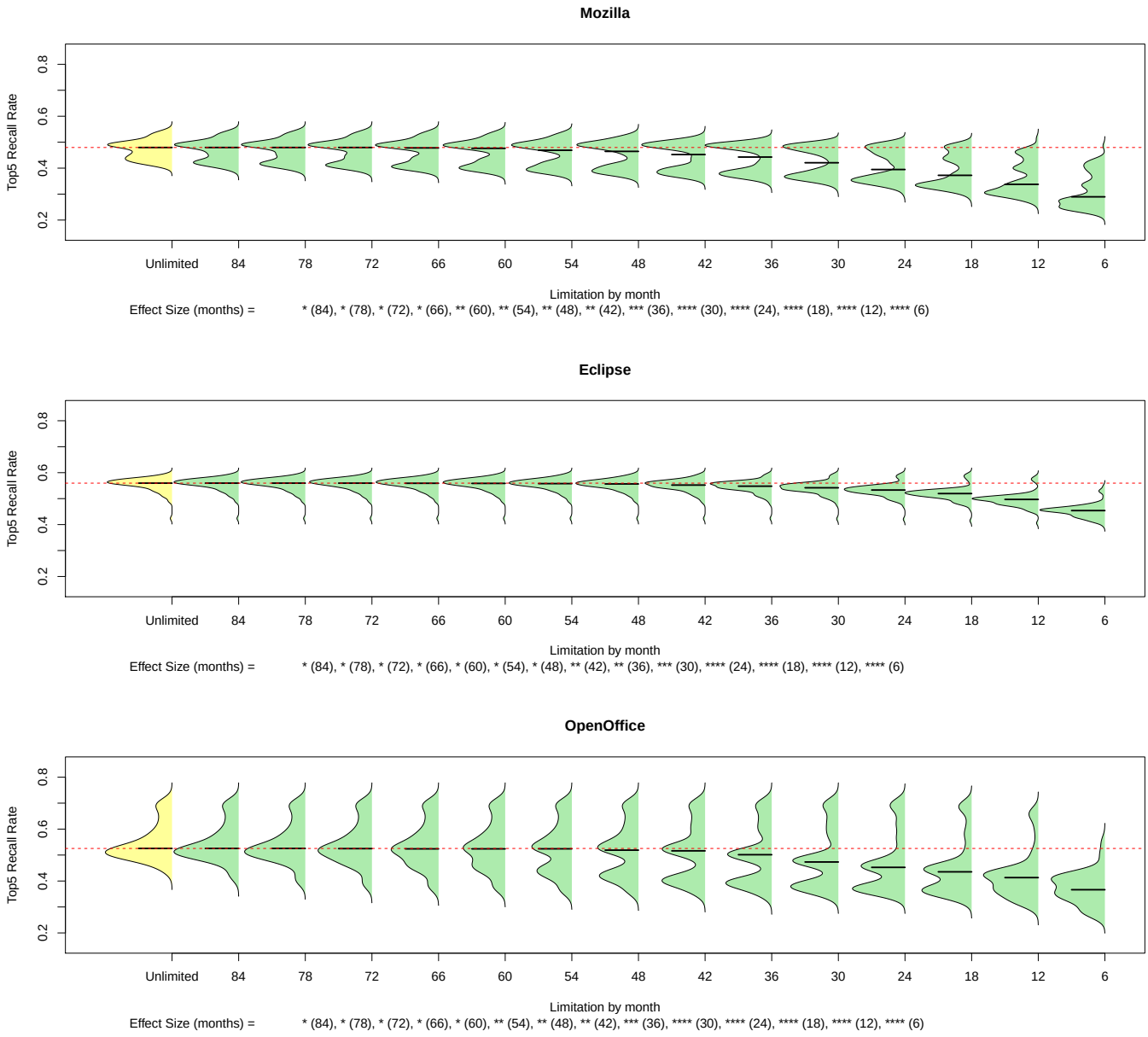
(a) REP

Fig. 9: The $\text{Recall}_{\text{top5}}$ yielded by the classical and realistic evaluation over time.



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

Fig. 10: 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.



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

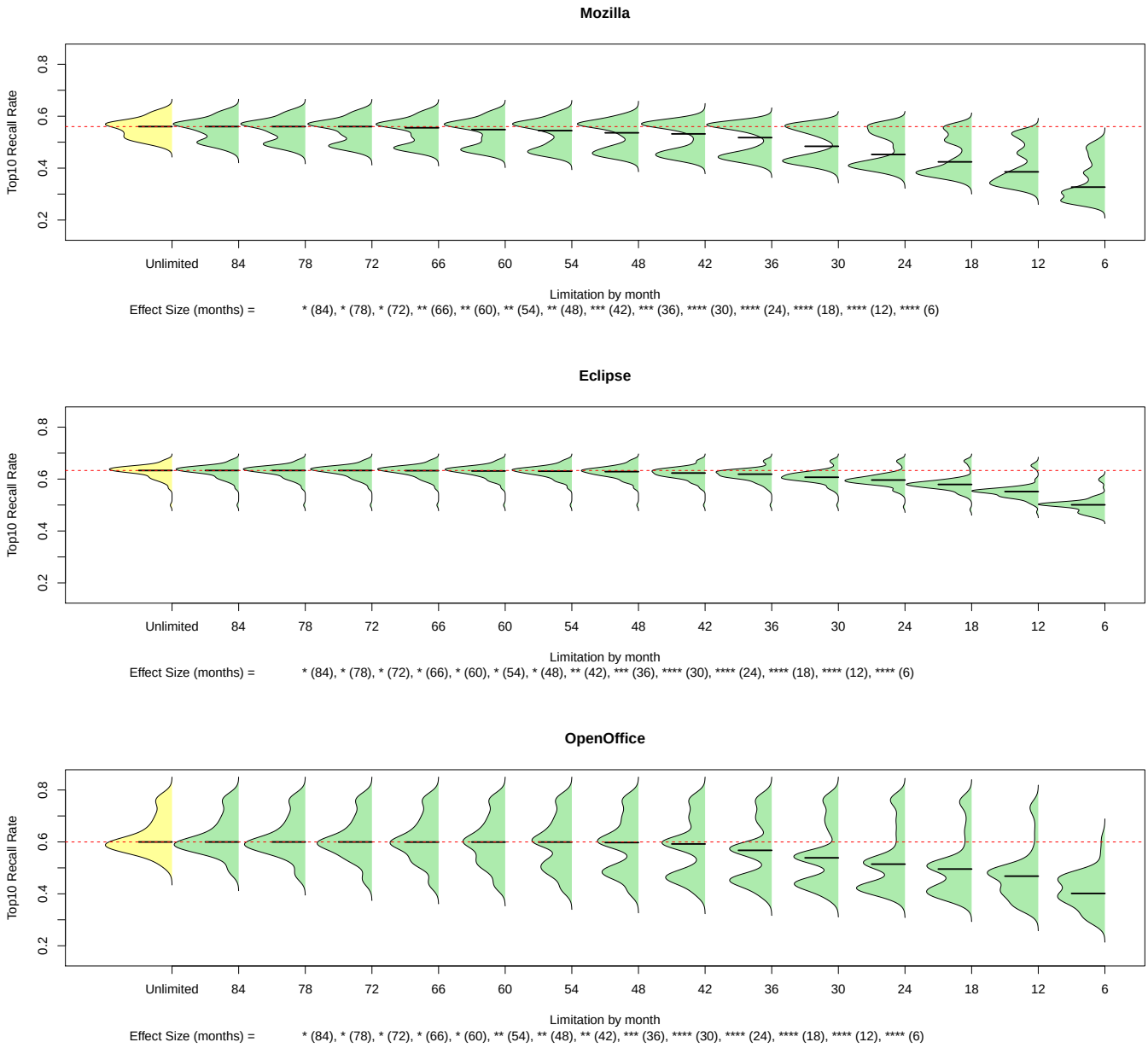
Fig. 11: The $\text{Recall}_{\text{top5}}$ yielded after limiting the issue reports that are searched by n-months for REP. The red dotted line is the median recall before applying any limitation.

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

Motivation. In our exploratory study and RQ1, we found that the classical evaluation overestimates the performance of approaches for the automatic retrieval of duplicate issue reports. The intuitive explanation for our finding is that the performance with the realistic evaluation drops as the ITS ages [6], 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 10 shows the distribution of the number of days between the reporting of a duplicate issue and its master report. As Figure 10 shows, the chances of a newly-reported issue being a duplicate of an old issue report are negligible. In this RQ, we 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, we reuse the same realistic evaluation runs of RQ1 (i.e., 100 randomly-selected runs over the lifetime of each studied ITS). However, we ignore some of the searched issue reports using an age-based threshold. For each newly-reported duplicate issue, we limit the searched issue reports to the ones that have a maximum age of n months. We evaluate values for n ranging from 6 months to 84 months (in steps of 6 months). Similar to RQ1, we use the Mann-Whitney U test and Cliff's Delta effect size to study the impact of limiting the issue reports that are searched.

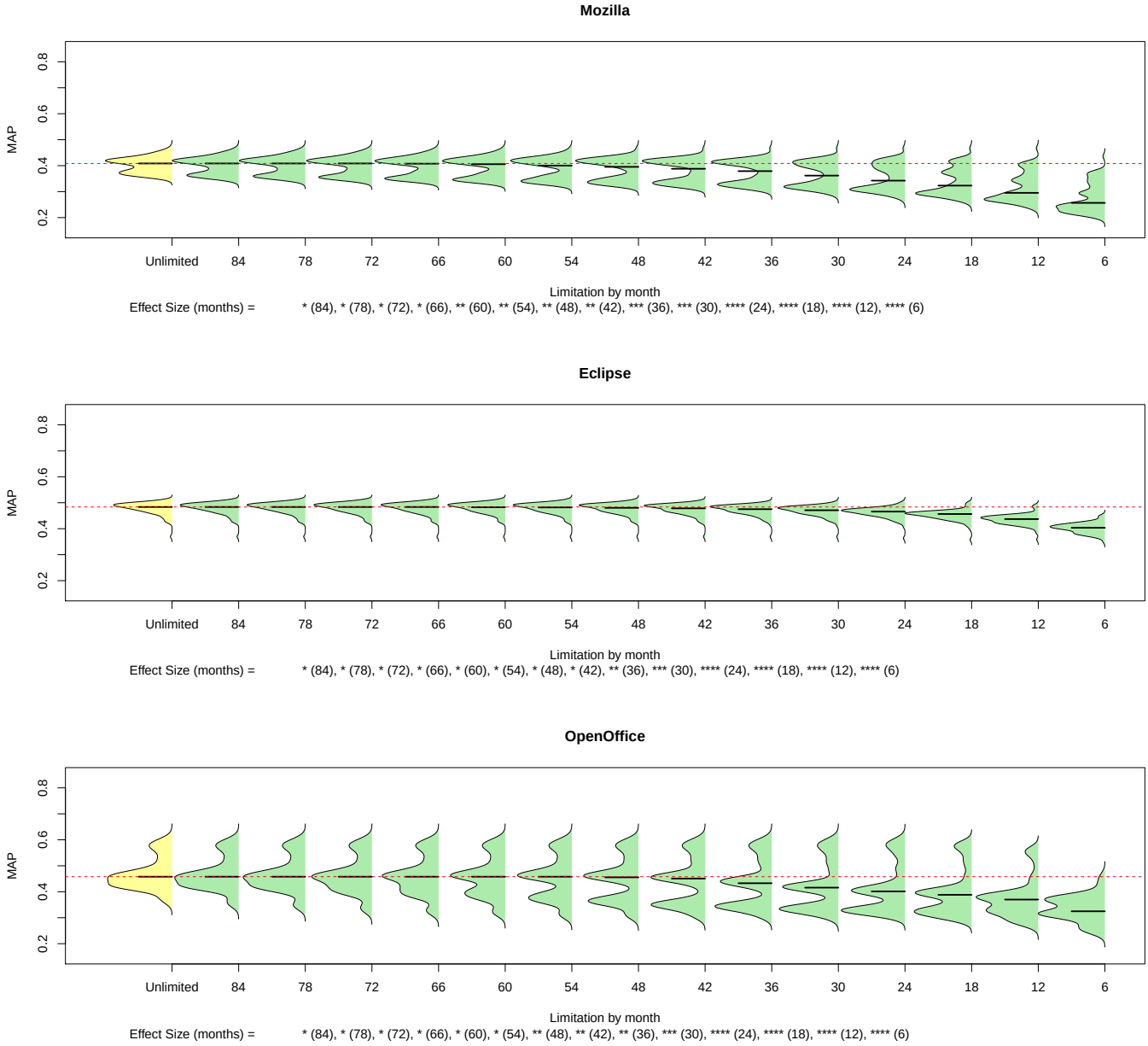


*Effect Size: * trivial, ** small, *** medium, **** large*

Fig. 12: 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.

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 11, 12 and 13 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 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 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 we omit these figures from this paper. 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



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

Fig. 13: 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.

Fig. 14: Overview of the threshold optimization approach.

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.

Observation 4: 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 report impact the performance according to the realistic evaluation?

Motivation. In the results of RQ2, we 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, we 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 issue report 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 10 shows, the majority of duplicates are short-term duplicates. However, long-term duplicates still need to be retrieved in some cases. Our hypothesis is that by filtering issue reports that are unlikely to be the master of a new issue report, we 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 we can prioritize certain reports in the ranking. We are the first to use the resolution field of the issue reports to improve the performance of the automated retrieval of duplicates.

Approach. Our approach is based on applying several filters to the ranked list of master candidates that is outputted by the automated approaches (i.e., BM25F and REP). These filters are based on the resolution field (e.g., FIXED or INVALID) and the time since the resolution of the issue reports in the returned candidate list at the time of each newly-reported duplicate issue. The idea of our approach is that we have a time-based threshold for each resolution value and each rank in the returned list of candidates. Then, we 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, we 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, we use a genetic algorithm [11]. We evaluate our proposed filtration on the same 100 chunks of data that were used in RQ1 while applying the realistic evaluation. We designed the following formula for the thresholds (in days):

$$threshold_{resolution,rank} = \frac{t_{resolution}}{(rank)^{r_{resolution}}} \quad (5)$$

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 issue report in the candidate list. The idea of using $r_{resolution}$ as the power of rank is to 'punish' the master candidates that have a low ranking (i.e., low similarity). An issue report 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, we extract the resolution history of each master candidate in the ITS. We then extract the resolution field at the time of reporting the new duplicate issue. By repeating this extraction for all master candidates for each newly-reported duplicate issue, we get an accurate view of the resolution field values.

Optimizing the thresholds. In order to find the optimal thresholds that are used in the filters, we use the NSGA-II [11] genetic algorithm to find the thresholds for each possible value for the resolution field.[7] Figure 15 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).

A genetic algorithm follows a process that closely resembles natural evolution to optimize a solution based on an objective. In our 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 15), 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 14 presents an overview of our threshold optimization process. Below we detail each step of our 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, we 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 16). 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

[7] In particular, we use the MOEA framework [14].

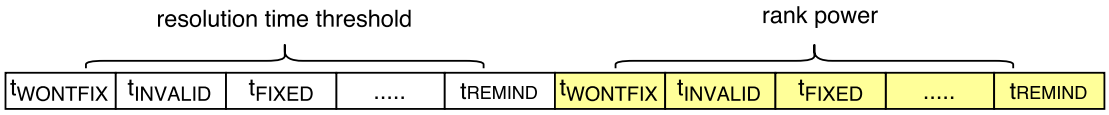


Fig. 15: The vector of threshold variables that is optimized by the NSGA-II algorithm.

TABLE 8: 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

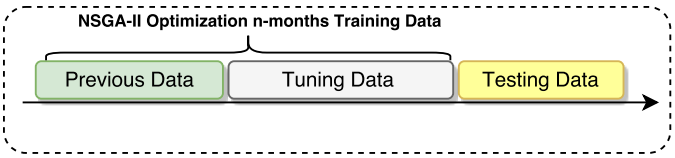


Fig. 16: Training periods selection for thresholds optimization for each chunk of the 100.

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 17 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. We used 1,000 iterations in our experiments because the relative improvement in performance did not change considerably after 1,000 iterations.

We used the default settings [14] 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 we 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, we run the algorithm for each studied ITS and automated approach.

An example of optimized thresholds. Table 8 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, we show the resulting thresholds for the top 3 ranks. We observe that the optimized thresholds for the WORKSFORME issue 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, we 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 18 and 19 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 20 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 [1, 16, 22, 30, 36]. Hence, our 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 a project since there are only a small number of resolved issue reports, 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.

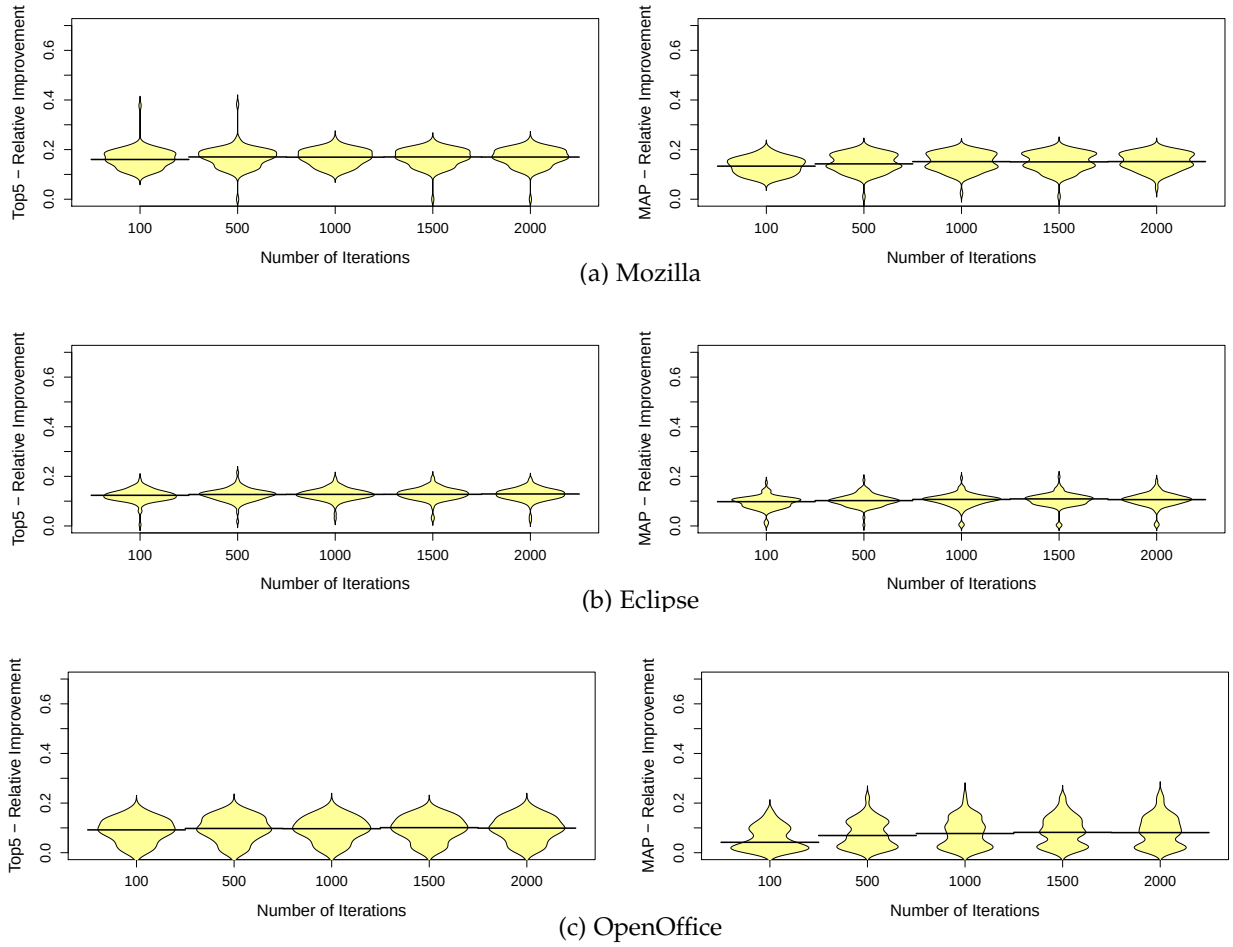


Fig. 17: The relative improvement in performance after running the genetic algorithm for various numbers of iterations for the REP approach.

Longer training periods do not increase the performance. Figure 21 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. We observed similar behaviour for the BM25F approach, hence we omit the figures for BM25F from the paper.

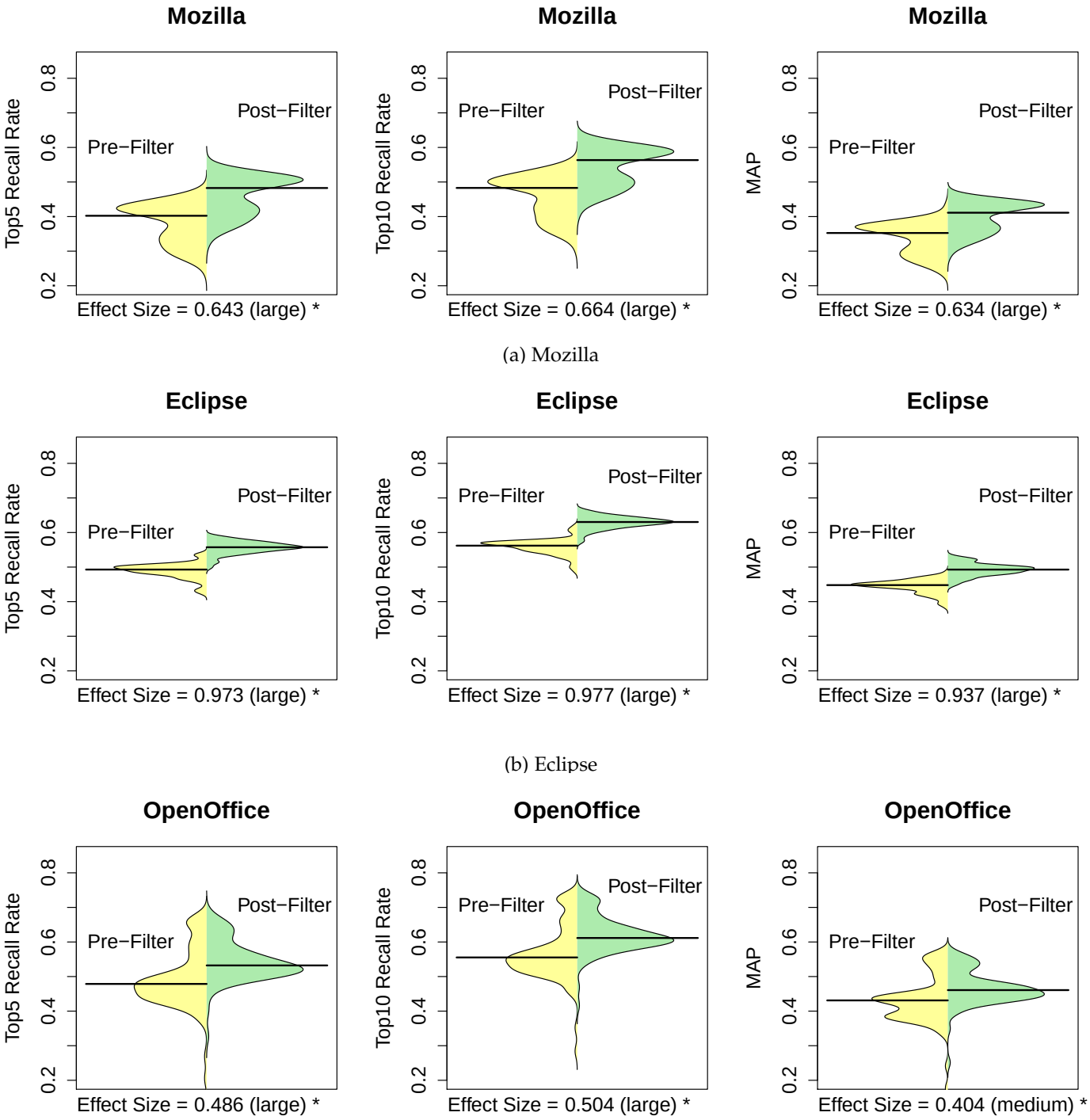
> **Observation 5:** While applying the realistic evaluation, leveraging the resolution field value and the rank to filter the returned master candidates improves the overall performance of automated approaches for the retrieval of duplicate reports with a median of 10-21.5% for recall and 7-18% for MAP. The thresholds that are used by the filters can be optimized automatically using a genetic algorithm.

6 Threats to Validity

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

External Validity. One of the external threats to our results is generalization. In this paper, we 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 from commercial software. In addition, all our studied projects make use of the Bugzilla issue tracking system. Our 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. Our study shows that we should not simply ignore 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 a significant one on a large number of projects.

Construct Validity. In our experiments, we tested the automated retrieval of duplicates using two approaches which can threaten the generality of our findings to other approaches. A majority of approaches for retrieving duplicate issue reports share the same base technique (TF-IDF). In this paper, we apply REP and BM25F as these are by far the most used TF-IDF-based approaches for the retrieval of duplicate issue reports [1, 16, 22, 30, 36]. REP [30]



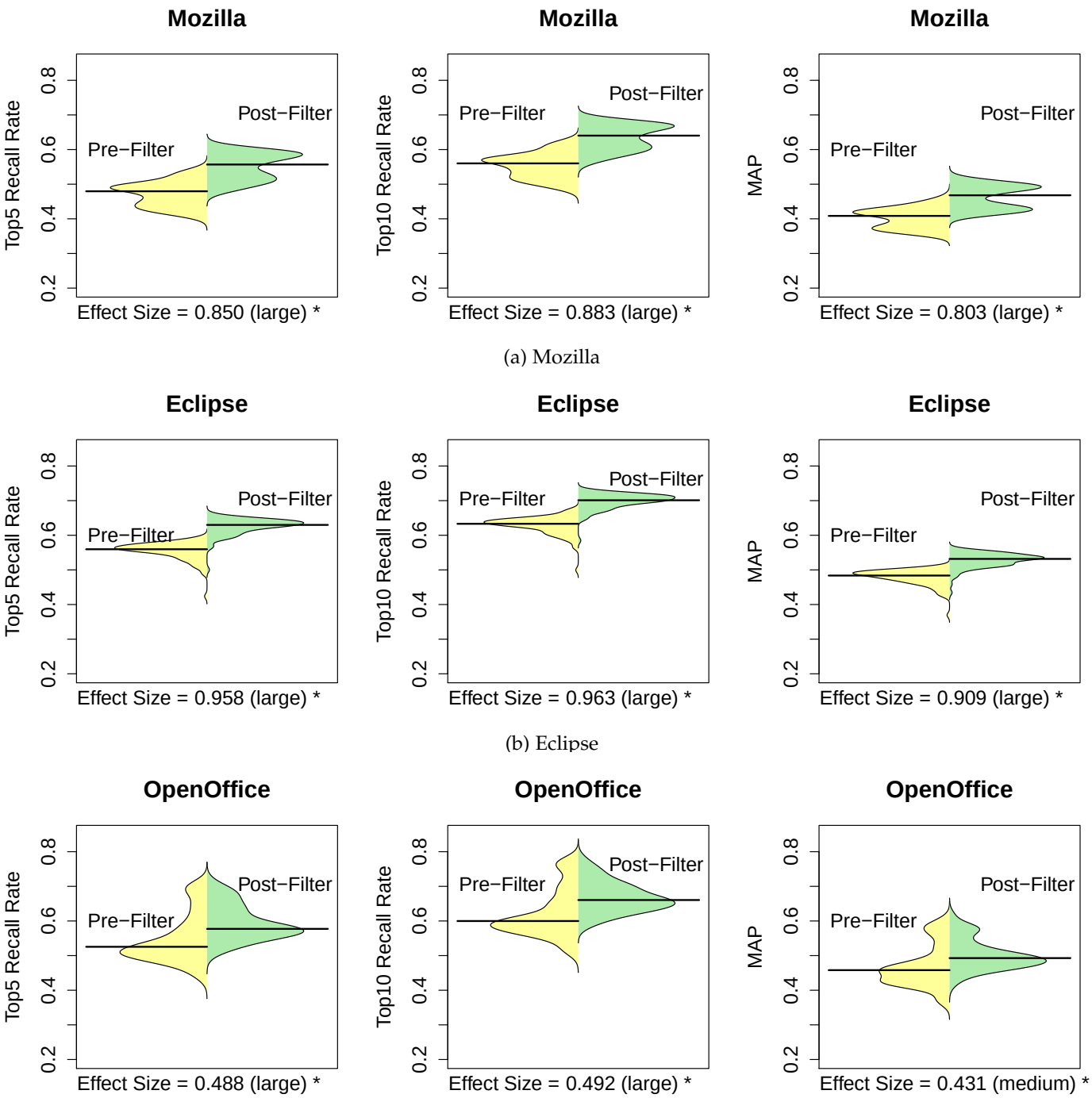
(a) Mozilla

(b) Eclipse

(c) OpenOffice

* $p < 0.001$

Fig. 18: The impact of using the resolution field on the results of the BM25F approach (2 months training data).



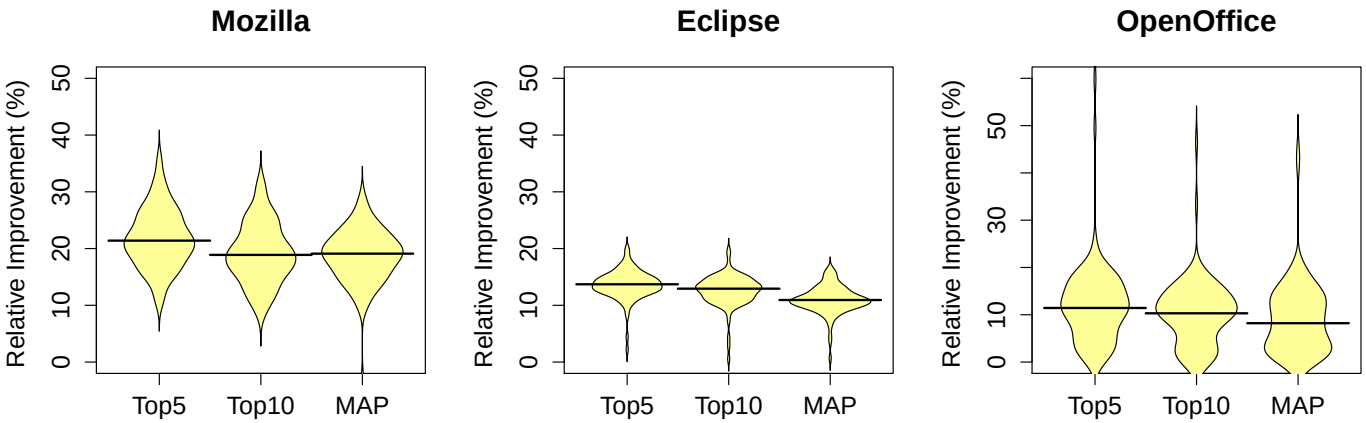
(a) Mozilla

(b) Eclipse

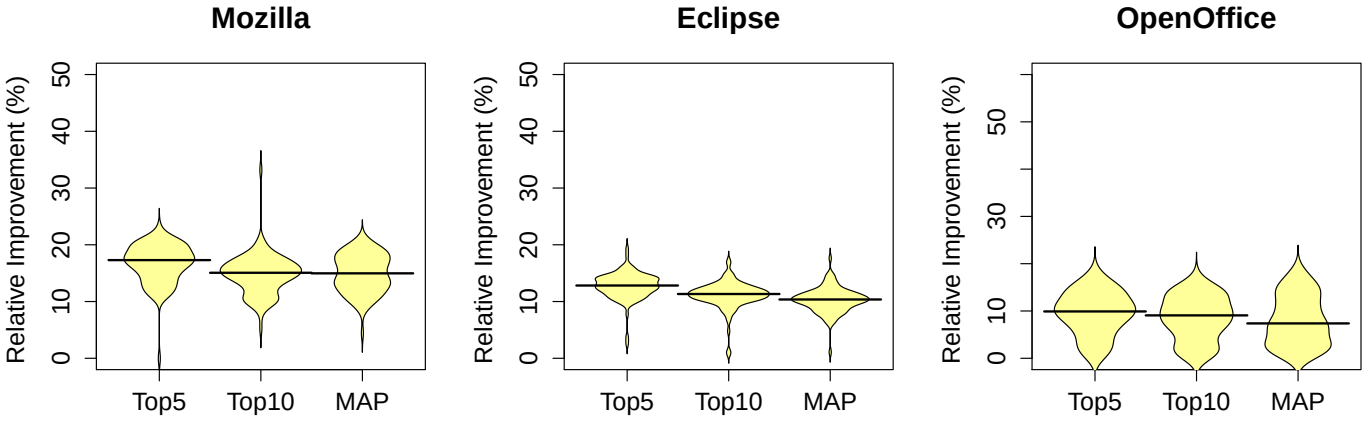
(c) OpenOffice

* $p < 0.001$

Fig. 19: The impact of using the resolution field on the results of the REP approach (2 months training data).



(a) BM25F



(b) REP

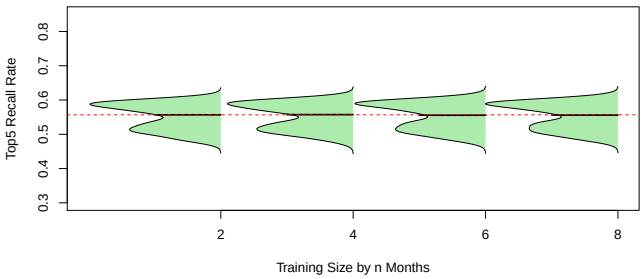
Fig. 20: The relative improvement in performance after filtering the results using our approach.

and BM25F [26] have always been treated as two separate approaches in prior research [22, 30, 36]. 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. [30]) along with categorical fields of issue reports. Nowadays, even the most recent approaches make only small increments to the REP or BM25F approach [1, 16, 22, 36]. Therefore, studying the REP and BM25F approach covers the majority of the spectrum of the approaches for retrieving duplicate issue reports. Hence, the observations in this paper are highly probable to hold for similar approaches in literature which increases the applicability and impact of our findings. Our 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 our results are indeed generalizable to the evaluation of approaches that are not based on BM25F or REP.

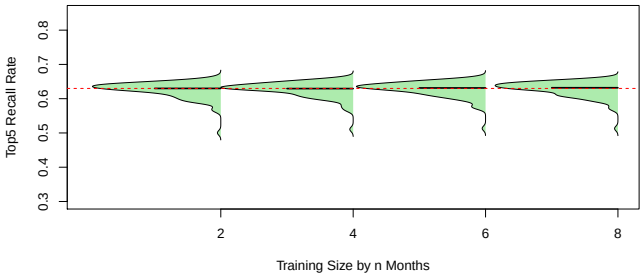
In the MAP metric calculation, we used 1,000 as maximum list size. The list size of 1,000 includes more than 90% of all duplicates which yields accurate results for calculating the MAP metric for all available duplicates. We did not use 100% MAP because this is extremely computationally intensive [22]. In addition, our 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, we can select from a large number of chunks of data with a one-year length (e.g., January, 2010 to December, 2010 and February, 2011 to January, 2012). However, running the approaches for retrieving duplicate issue reports is costly in terms of time, hence we limit the number of chunks that we 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.

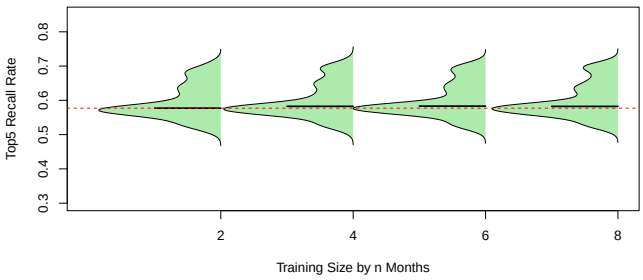
We evaluated only how leveraging the resolution field impacts the performance in RQ3. The main motivation behind the selection of the resolution field is that intuitively, the resolution is related to the chances of a newly-submitted report being a duplicate. For example, if an issue report was resolved as fixed several years ago, the chances of a



(a) Mozilla



(b) Eclipse



(c) OpenOffice

Fig. 21: 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.

newly-submitted report being a duplicate of that report are relatively small. Hence, we use the resolution field in combination with thresholds for the time and rank to give a lower priority. At this moment, we have no such intuition for the other available fields. We 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.

7 Conclusion

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 paper, we 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, we 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 paper, we 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 our paper are as follows:

1) In future studies on 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 paper 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.

We acknowledge that our proposed realistic evaluation requires considerably more computational power than the classical evaluation. However, the wide availability of computation clusters makes the realistic evaluation feasible for all researchers. In addition, the results presented in this paper 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.

8 Acknowledgments

This study would not have been possible without the tools shared by Sun et al. [30], as well as the High Performance Computing (HPC) systems that are provided by Compute Canada[8] and the Centre for Advanced Computing[9].

References

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

[2] A. Alipour, A. Hindle, and E. Stroulia. 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, 2013.

[8] https://www.computecanada.ca/
[9] http://cac.queensu.ca/

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

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

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

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

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

[8] D. Bertram, A. Voida, S. Greenberg, and R. Walker. 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, 2010.

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

[10] G. Chowdhury. *Introduction to modern information retrieval*. Facet publishing, 2010.

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

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

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

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

[15] L. Hiew. Assisted detection of duplicate bug reports, Jan 2006.

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

[17] N. Jalbert and W. Weimer. 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, 2008.

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

[19] N. Kaushik and L. Tahvildari. 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, 2012.

[20] A. Lazar, S. Ritchey, and B. Sharif. 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, 2014.

[21] J. D. Long, D. Feng, and N. Cliff. Ordinal analysis of behavioral data. *Handbook of psychology*, 2003.

[22] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. 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, 2012.

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

[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 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, 2002.

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

[26] S. Robertson, H. Zaragoza, and M. Taylor. 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, 2004.

[27] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine. Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and Cohens'd indices the most appropriate choices. In *annual meeting of the Southern Association for Institutional Research*, 2006.

[28] P. Runeson, M. Alexandersson, and O. Nyholm. 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, 2007.

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

[30] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. 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, 2011.

[31] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. 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, 2010.

[32] A. Sureka and P. Jalote. 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, 2010.

[33] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. 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, 2006.

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

[35] J. Zhou and H. Zhang. 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, 2012. ACM.

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

View publication stats