Studying the Needed Effort for Identifying Duplicate Issues

Mohamed Sami Rakha · Weiyi Shang · Ahmed E. Hassan

Abstract Many recent software engineering papers have examined duplicate issue reports. Thus far, duplicate reports have been considered a hindrance to developers and a drain on their resources. As a result, prior research in this area focuses on proposing automated approaches to accurately identify duplicate reports. However, there exists no studies that attempt to quantify the actual effort that is spent on identifying duplicate issue reports. In this paper, we empirically examine the effort that is needed for manually identifying duplicate reports in four open source projects, i.e., Firefox, SeaMonkey, Bugzilla and Eclipse-Platform. Our results show that: (i) More than 50% of the duplicate reports are identified within half a day. Most of the duplicate reports are identified without any discussion and with the involvement of very few people; (ii) A classification model built using a set of factors that are extracted from duplicate issue reports classifies duplicates according to the effort that is needed to identify them with a precision of 0.60 to 0.77, a recall of 0.23 to 0.96, and an ROC area of 0.68 to 0.80; and (iii) Factors that capture the developer awareness of the duplicate issue's peers (i.e., other duplicates of that issue) and textual similarity of a new report to prior reports are the most influential factors in our models. Our findings highlight the need for effort-aware evaluation of approaches that identify duplicate issue reports, since the identification of a considerable amount of duplicate reports (over 50%) appear to be a relatively trivial task for developers. To better assist developers, research on identifying duplicate issue reports should put greater emphasis on assisting developers in identifying effort-consuming duplicate issues.

Mohamed Sami Rakha, Weiyi Shang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL), School of Computing
Queen's University
Kingston, Ontario, Canada
Tel.: +1 613-533-6802
E-mail: {rakha, swy, ahmed}@cs.queensu.ca

1 Introduction

Issue tracking systems are widely used in practice to track the requests and concerns of users. Such systems provide a communication platform to help developers get information from users for the rapid resolution of issues [6]. These systems also allow users to track the status of their submitted issue reports.

Newly submitted issue reports usually go through a filtration process before they are assigned to developers. One of the common filtering steps is to identify whether a newly reported issue is a duplicate of an existing issue. Two issue reports are considered duplicates of each other, if they refer to a similar software problem or propose a similar feature. Due to the limited resources of software projects, duplicate issue reports are often considered a waste of developers' effort. Studies show that up to 20-30% of the issue reports are duplicates [9, 17]. Such large amounts of duplicate issue reports have motivated extensive studies on the automated identification of duplicate issue reports [2, 4, 17, 33, 38, 40, 44]. Various advanced approaches, such as Natural Language Processing (NLP), are commonly used to accurately identify whether a new issue report is a duplicate of previously reported issues [38].

While prior research aims to reduce the needed efforts for identifying duplicate issue reports, there exist no studies that examine the actual effort that is spent on manually identifying the duplicate issues in practice. In fact, the needed effort for identifying duplicate issue reports may vary considerably. For example, a Mozilla developer needed less than 15 minutes to identify that issue #312782[1] is a duplicate of a previously reported issue, while the duplicate identification of issue #65305[2] required around 44 days, and involved 20 comments from 11 people.

To better understand the effort that is involved in identifying duplicate issue reports in practice, this paper studies duplicate reports from four open source projects (Firefox, SeaMonkey, Bugzilla and Eclipse platform). In particular, we explore the following research questions:

RQ1: How much effort is needed to identify a duplicate issue report?
Half of the duplicate reports are identified in less than half a day. 50 - 60% of the duplicates are identified without any discussion.

RQ2: How well can we model the needed effort for identifying duplicate issue reports?
Random forest classifiers trained using factors derived from duplicate issue reports achieve a precision of 0.60 *to* 0.77*, and a recall of* 0.23 *to* 0.96*, along with an ROC area of* 0.68 *to* 0.80*.*

RQ3: What are the most influential factors on the effort that is needed for identifying duplicate issue reports?
The textual similarity between a new report and previously reported issues plays an important role in explaining the effort that is needed for identi-

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

fying duplicate reports. Another influential factor is the team's awareness of previously reported issues. Such awareness is measured using: a) The experience of a developer in identifying prior duplicates, b) The recency of the duplicate report relative to the previously filed reports of that issue, and c) The number of people that are involved in the previously filed reports of that issue.

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

The rest of the paper is organized as follows: Section 2 presents related work on the management of duplicate issue reports. Section 3 describes the studied projects. Section 4 provides an overview of our experimental setup. Section 5 motivates our research questions, discusses the approaches that we used to answer our questions and presents the results of our questions. Finally, Section 6 outlines some of the threats to the validity of our study and Section 7 concludes the paper.

2 Background and Related Work

In this section we present prior research that relates to our study.

2.1 Empirical Studies on Duplicate Issue Reports

Duplicate issue reports represent a large portion of issue reports. Anvik et al. [4] report that 20-30% of the issue reports in Eclipse and Firefox respectively are duplicates. Cavalcanti et al. [13] found that duplicate reports represent 32%, 43%, and 8% of all the reports in Epiphany, Evolution and Tomcat, respectively.

Duplicate reports are not always harmful. Bettenburg et al. [9] found that merging the information across duplicate reports produces additional useful information over using the information from a single report. In particular, Bettenburg et al. found that such additional information improves the accuracy of automated approaches for issue triaging[3] by up to 65%.

Several prior studies have explored the effort associated with duplicate reports. Davidson et al. [14] and Cavalcanti et al. [12,13] examined the effort that is associated with closing a duplicate report: Davidson et al. looked at the

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

time needed for closing a duplicate report after it was triaged, and Cavalcanti et al. examined the total time that is needed to close a duplicate report. Cavalcanti et al. also investigated the time spent by report submitters to check whether they are about to file an issue that is a duplicate of previously-reported issues. In our case, we consider the needed effort for identifying the duplicate report not the time to close it (since up to 50% of the studied duplicate issue reports are not closed).

2.2 Automated Identification of Duplicate Issue Reports

Duplicate issue report identification is the task of matching a newly reported issue with one or more previously reported ones. Prior research has proposed various duplicate identification approaches that leverage textual and non-textual information that are derived from issue reports.

Leveraging textual information. Anvik et al. [4] proposed an approach that uses cosine similarity to classify newly reported issues as either new reports or duplicate ones. Their approach can correctly identify 28% of the duplicate issue reports in the Firefox 1.0 project. Runeson et al. [38] proposed an approach to identify duplicate reports based on Natural Language Processing (NLP) and evaluated the approach on issue reports from Sony Ericsson. The prototype tool correctly classified 40% of the duplicate issue reports. Nagwani et al. [33] used string similarity measures, such as TD-IDF similarity, and text semantics measures, such as edge counting methods, to identify duplicate issue reports. Prifti et al. [35] proposed an approach based on information retrieval, while limiting the search for duplicates to the most recently filed reports. Prifti et al.'s approach is able to identify around 53% of the duplicate issue reports in their case study. Sun et al. [41] built a discriminative model to determine if two issue reports are duplicates. The output of the model is a probability score which is used to determine the likelihood that an issue is a duplicate. Sun et al.'s approach outperforms the prior state of the art approaches [17, 38, 44] by 17-31%, 22-26%, and 35-43% on OpenOffice, Firefox, and Eclipse datasets, respectively. Sureka et al. [42] used a character n-grams approach [21] to measure the text similarity between the titles and descriptions of issue reports. Sureka et al.'s approach achieved around 62% recall for 2,270 randomly selected reports from the Eclipse project.

Leveraging both textual and non-textual information. Other information in issue reports, such as the component, version, Bug_Id, and platform, are found useful in improving the accuracy of approaches for duplicate identification. Jalbert et al. [17] proposed an approach that combines textual similarity with non-textual features that are derived from issue reports. Their approach was able to filter out 8% of the duplicate issue reports while allowing at least one report for each unique issue to reach developers. Jalbert et al.'s approach achieved a recall of 51%. Kaushik et al. [23] compared the performance of word-based and topic-based information retrieval models using non-textual features (such as Component) and textual features. For the word based models, Kaushik et al. applied different term weighting approaches including the ones proposed by Runeson et al. [38] and Jalbert et al. [17]. While for the topic based models, Kaushik et al. applied Latent Semantic Indexing (LSI) [15], Latent Dirichlet Allocation (LDA) [10], and Random Indexing [22] approaches. The comparison is applied on Mozilla and Eclipse datasets. The results show that the word based approaches outperform the topic based approaches.

Sun et al. [40] leveraged a BM25F model [36] to combine both textual and non-textual information for identifying duplicate issue reports. Sun et al. achieved an improvement of 10-27% in recall in comparison to Sureka et al.'s approach [42]. Alipour et al. [2] proposed an approach which uses software dictionaries and word lists to extract the implicit context of each issue report. Alipour et al. showed that the use of contextual features improves the accuracy of duplicate identification by 11.55% over Sun et al. [40]'s approach on a large dataset of issues from several Android projects.

Lazar et al. [25] proposed a duplicate identification approach based on new textual features that capture WordNet augmented word-overlap and normalized sentences-length differences. Lazar et al. also used non-textual features such as component and open_date. The new set of textual features achieves an accuracy improvement between 3.25% and 6.32% over the Alipour et al.'s approach [2].

Wang et al. [44] combined textual information and execution traces to identify duplicate issue reports. For two issue reports, Wang et al. calculates two separate similarity measures: one measure for textual similarity and another measure to capture the similarity of the execution traces. By combining both similarity measures, a list of possible duplicate issue reports is suggested for each newly reported issue. Wang et al.'s approach achieved an accuracy of 67% to 93%. Lerch et al. [26] proposed an automatic duplicate identification approach based on the stack traces that are attached to issue reports. Lerch et al. used the TD-IDF similarity of stack traces for identifying duplicate reports (around 10% of reports contain stack traces).

Aggarwal et al. [1] proposed a duplicate identification approach based on the contextual information that is extracted from software-engineering textbooks and project documentation. Aggarwal et al.'s approach achieved lower accuracy than the approach by Alipour et al. [2]. However, Aggarwal et al.'s approach is simpler with up to six times less computation time on the same Android dataset.

Although prior research sought to accurately identify duplicate issue reports with advanced approaches, there exists no research which examines whether identifying duplicate issue reports is indeed consuming a large amount of developer effort. Hence in this paper, we do not aim to propose yet another duplicate identification approach. Instead, we study the needed effort for identifying duplicate issue reports in order to understand how and whether developers can make good use of automated approaches for duplicate identification in practice.

Table 1 An overview of the issue reports in the studied projects.

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

3 Case Study Setup

In this section, we present the studied projects and the process of preparing the data that is used for our case studies. We share our data and scripts as an online replication package[4].

3.1 Studied Projects

We used the issue reports that are stored in the Bugzilla issue tracking system from four open-source projects: Firefox, SeaMonkey, Bugzilla, and Eclipse-Platform. All four projects are mature projects with years of development history. Table 1 shows an overview of the studied projects. Firefox, SeaMonkey and Bugzilla are open-source projects from the Mozilla foundation. These projects have been frequently studied by prior research for evaluating the automated approaches for duplicates identification [4]. Eclipse-Platform is a sub-project of Eclipse, which is one of the most popular Java Integrated Development Environments (IDEs). The duplicate issue reports in the Eclipse project have also been studied by prior research. For example, Bettenburg et al. [8] used the Eclipse dataset to demonstrate the value of merging information across duplicate issue reports. We studied the projects up to the end of 2010 because Bugzilla 4.0 (released in February 2011) introduced an automated duplicate identification feature when filing an issue[5]. Such an automated identification feature may bias our measurement of the needed effort for identifying duplicate reports. For example, many trivial duplicate reports might not be filled since Bugzilla would warn about them at the filing time of the report.

3.2 Linking of Duplicate Issue Reports

In practice, the identification of duplicate reports is done manually. Whenever developers identify a duplicate issue report, they change the resolution status of a report to *DUPLICATE*, and add a reference (i.e., *Id-Number*) to the previously reported issue. An automatically-generated comment is attached to the discussion of the issue report in order to indicate that this issue report

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

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

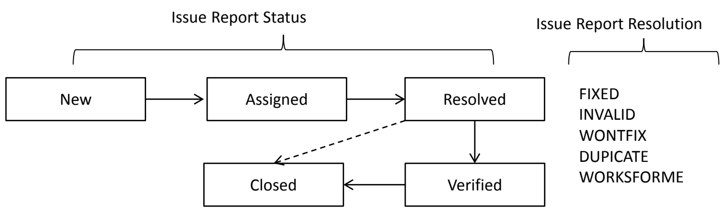


Fig. 1 The life cycle of an issue report in Bugzilla.

is a duplicate of another one. We use such automatically generated comments to identify duplicate issue reports. For example, if we find a message like ****This issue has been marked as a duplicate of B **** in the discussions of issue report A, we consider issue report A and B as duplicates of each other. We consider the latest *Id-Number* in case it was changed. We group duplicate issue reports. For example, if we find that issue report A and B are duplicates of each other and issue report B and C are duplicates of each other, we consider issue reports A, B and C as duplicates of each other. We repeat this grouping process, until we cannot add additional duplicate reports to a group.

3.3 Pre-processing of Duplicate Issue Reports

Figure 1 shows the life cycle of an issue report in Bugzilla [24]. First an issue is reported with status *NEW*. Next, a triaging process determines if an issue report should be assigned to a developer to fix. If the issue is assigned to a developer the status of the issue is changed to *ASSIGNED*. After the developer fixes the issue, the report status is changed to *RESOLVED-FIXED*. In some cases, an issue is not fixed because it is invalid (i.e., *RESOLVED-INVALID*), not reproducible (i.e., *RESOLVED-WORKSFORME*) or duplicate (i.e., *RESOLVED-DUPLICATE*). Finally, the issue is verified by another developer or by a tester (i.e., *VERIFIED-FIXED* or *CLOSED-FIXED*).

Figure 2 presents the typical process for managing a duplicate report. First, a duplicate report is triaged. Then, it gets identified as a duplicate. The un-duplication happens when the *DUPLICATE* resolution is removed. If a developer finds that a duplicate report is actually not a duplicate, the *DUPLICATE* resolution is removed. On average 2.5% of the duplicates in the studied projects are un-duplicated. We remove the un-duplicated issue reports from our analysis of duplicate reports. The removal is performed by ensuring that the last resolution status is one of the following statuses: *RESOLVED-DUPLICATE*, *VERIFIED-DUPLICATE* and *CLOSED-DUPLICATE*.

4 Results

We now present the results of our research questions. The presentation of each research question is composed of three parts: the motivation of the research

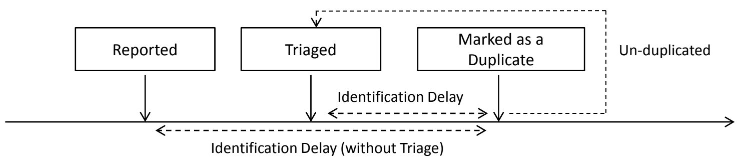


Fig. 2 A typical timeline for managing duplicate issue reports.

question, the approach that we used to address the research question, and our experimental results.

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

Motivation. There exists a considerable amount of duplicate reports. Prior research shows that around 20-30% of the issue reports in Mozilla and Eclipse, respectively, are duplicates [7]. Duplicate issue reports are typically considered as a waste of developers' time and resources. Thus, many researchers have proposed automated identification approaches for duplicates. However, the effort that is needed for manually identifying duplicate reports has never been investigated. Therefore, we would like to examine whether identifying duplicate reports consumes a considerable amount of developers' effort in practice.
Approach. To measure the effort of identifying duplicate issue reports, we calculate the following three measures:

– *Identification Delay.* We calculate the identification delay for each duplicate report (*see* Figure 2). This measure provides us a rough estimate of the needed effort. The more time it takes to identify a duplicate report, the more developers' effort is needed. For reports that are triaged, we measure the time in days between the triaging of the report and when the report is marked as a duplicate. For reports that are never triaged, we count the number of days between the reporting of the report and when the report is marked as a duplicate.
– *Identification Discussions.* We count the number of comments posted on an issue report before it is identified as a duplicate report. The more discussions about a particular report, the more complex and effort consuming it is to identify a duplicate. We ignore the comments that are not representing the issue report discussions. For example, all auto-generated comments, such as *Created an attachment*, are filtered. Then, in the case of duplicates with one remaining comment, we filter that comment if it is posted by the same reporter of the issue in order to add missing information.
– *People Involved.* We count the number of unique people that are involved in discussing each issue other than the reporter before the issue is marked as a duplicate. More people discussing a duplicate report indicates that more team time and resources are spent on understanding that particular duplicate report.

Table 2 Mean and Five number summary of the three effort measures.

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

Results. ***Most of the duplicate reports are identified within a short time.*** We find that the median identification delay for the studied projects is between 0.18 to 0.83 days (*see* Table 2). Such results show that more than 50% of the duplicate reports are identified in less than one day. We do note that our identification delay metric is an over estimate of the actual time that is spent on identifying a duplicate report. Developers are not likely to start examining a report as soon as it is reported. Instead, there are many reasons for such activity to be delayed (e.g., reports are filed while developers are away from their desks, or developers are busy with other activities).

Most of the duplicate reports are identified without any discussion. We find that over 50% of the issue reports are marked as duplicates with no discussions, and over 75% of the issue reports are marked as duplicates with just one or two comments (*see* Table 2).

Figure 3 shows the cumulative density function (CDF) plot of the discussions count for the four projects. The curve shows that at least 50% of the duplicates are identified without any discussions. We only show one CDF plot (the one for identification discussions) since the other CDF plots follow the same pattern.

Very few people are involved in identifying duplicates. Table 2 shows that for over half of the duplicate reports, the identification effort involves one person without counting the reporter of the issue.

Discussion. The results in Table 2 show that it takes little effort (i.e., time, discussions, and people) to identify the majority of the duplicate reports. However, it is also interesting to note that there are issue reports taking thousands of days to identify, with up to 230 comments, involving up to 75 people. Such issue reports consume a considerable amount of developers' efforts until developers realize that the issue reports are duplicates of previously-reported issues. Although various advanced approaches are proposed by prior research to automatically identify duplicate reports, those approaches do not differentiate between the easily-identifiable duplicate issue reports with the ones that are more effort-consuming to identify. Automated duplicate identification approaches should focus on helping developers identify duplicate reports that consume more effort to identify.

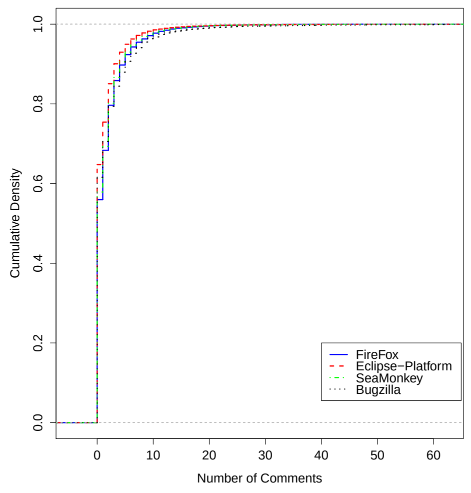


Fig. 3 Cumulative Density Function (CDF) of identification discussions for the four studied projects. The x-axis shows the identification discussions, and y-axis shows the cumulative density

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

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

Motivation. The results of RQ1 show that while the majority of the duplicated reports are identified with little effort, the duplicate identification of some reports requires considerable amount of efforts. Understanding the effort that is needed to identify duplicate reports is important for managing project resources. To the best of our knowledge, no study has ever examined such effort and whether it is predictable (i.e., whether one can determine if a particular issue will require more effort over another issue). Therefore, in this RQ, we would like to build classifiers to determine the needed efforts to identify duplicate reports.
Approach. In order to build a classifier for the effort that is needed of identifying duplicates, we compute a set of factors from the duplicate reports. The

Table 3 The factors used to model the efforts needed for identifying duplicate reports.

Dimension	Factors	Value	**Definition** (d) — **Rationale** (r)
Peers	Peers Count	Numeric	d: The total number of peers that an issue report has. We only consider the peers that existed before the reporting of the new issue.
			r: An issue report with many existing peers might be easier to identify as a duplicate.
	Title Similarity	Numeric (0-1)	d: The maximum similarity between the title text of an issue report and title text of its peers. The similarity is measured using trigram matching.
			r: Similar title text as a previously reported issue may assist in identifying that an issue report is a duplicate of an existing one.
	Description Similarity	Numeric (0-1)	d: The maximum similarity between the description text of an issue report and the description text of its peers. The similarity is measured using trigram matching.
			r: Similar description text as a previously reported issue may assist in identifying that an issue report is a duplicate of an existing one.
	Peers CC List	Numeric	d: The sum of the unique persons in the CC lists of an issue report's peers.
			r: A larger peers' CC lists could be a sign of better awareness of an issue across the development team.
	Peers Comments	Numeric	d: The total number of comments that belong to the peers of an issue report. We only consider the comments that existed before the reporting of the issue report.
			r: If an issue report has well discussed peers (with more comments), the issue report may be easier to identify as a duplicate.
	Reporting Recency of Peers	Numeric	d: The smallest difference in days between the triage or report date of an issue report to the date when one of its peers are reported.
			r: If a developer has seen a similar issue that is reported recently, it could speed up the duplicate identification process.
	Recency of Peer Identification	Numeric	d: The smallest difference in days between the triage or report date of an issue report to the date when one of its peers is identified as a duplicate.
			r: If a developer has seen a similar issue that is identified recently, it could speed up the duplicate identification process.
	Recency of Closed Issue	Numeric	d: The smallest difference in days between the triage or report date of a duplicate report to the date when any issue report closed within the same project and component is closed.
			r: If a developer has recently fixed an issue within the same project and component, it could indicate the developer's activity and it could speed up the duplicate identification process within the same project and component.
Workload	Waiting Load	Numeric	d: The total number of other issues reports that are open at the time when that issue is reported within the same component and project.
			r: A large number of open issues could make developers too busy to recognize whether an issue is a duplicate.
Issue Report	Severity	Numeric	d: A number representing the severity of an issue report.
			r: More severe issue reports are likely to attract attention from developers, thus such high severity issues are likely to be identified as a duplicate faster.
	Priority	Numeric	d: A number representing the priority of an issue report.
			r: Issue report with higher priority are likely to attract attention from developers, thus such high severity issues are likely to be identified as a duplicate faster.
	Component	Nominal	d: The component specified in an issue report.
			r: Particular components may be more important than others. Hence issues associated with them are more likely to be identified as duplicates faster.
	Title Size	Numeric	d: The number of words in the title text.
			r: Issues with longer title might provide more useful information which might ease the duplicate identification.
	Description Size	Numeric	d: The number of words in the description of an issue report.
			r: Issues with longer description could be easier to understand and might ease the duplicate identification.
	isBlocking	Boolean	d: A boolean value indicating whether an issue report is blocking other issues.
			r: A blocking issue could be more important to address and this might lead to faster identification of duplicates.
	isDependable	Boolean	d: A boolean value indicating whether an issue report depends on other issues.
			r: An issue depending on other issues may have more attention from developers and get identified as a duplicate easier.
	Reproduce Steps	Boolean	d: A boolean value indicating whether the description of an issue report contains steps for reproducing the issue.
			r: Issues with reproduce steps should be easier to understand and might ease the identification of duplicates.
Work Habits	isWeekend	Boolean	d: A boolean value representing whether an issue is reported on a weekend day (i.e., Saturday and Sunday) or not.
			r: The duplicate reports submitted on weekends might take longer time to be identified than the ones on week days.
Identifier Experience	Total Duplicates Identified	Numeric	d: The total number of duplicate reports that are previously identified by the same identifier per each duplicate report.
			r: An identifier with more experience in finding duplicate reports might affect the effort needed for identifying a new duplicate report.
	Duplicate Peers Identified	Numeric	d: The total number of peers duplicate reports that are previously identified by the same identifier.
			r: An identifier that has seen several peer duplicates might identify duplicates faster.
	Fixed Issues per Identifier	Numeric	d: The total number of issue reports that are previously marked as FIXED by the same identifier.
			r: An identifier with more experience in fixing issues might identify duplicates faster.

factors are from five dimensions: *Peers*, *Workload*, *Issue Report*, *Work Habits*, and *Identifier Experience*. We describe each dimension below:

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

report, the easier it is for developers to identify that an issue report is a duplicate.
- **Workload:** Workload refers to the number of open issues at the time of reporting or triaging of a duplicate report. If there is a high workload at the time of reporting an issue, the duplicate identification of a newly reported issue may be delayed.
- **Issue Report:** Issue report refers to the information that can be derived from an issue report at the time of its reporting. Better information in the issue report should help reduce the effort that is needed to identify that the issue report has duplicates.
- **Work Habits:** Work habit refers to the time when an issue is reported. If an issue is reported during the weekends or at night, identifying that the report is a duplicate may be delayed.
- **Identifier Experience:** Refers to the possible information that a duplicate report identifier has. Duplicate reports identified by experienced persons might require less effort.

Table 3 presents the list of our studied factors along with their descriptions and our rationale for studying such factors. To calculate the factors *Title Similarity* and *Description Similarity* from the *Peers Dimension*, we first apply simple pre-processing steps, such as removing special characters, long spaces and converting all text to lowercase. Then, we apply a trigram algorithm [25] to calculate the similarity between texts. For the *Reproduce Steps* factor, we use a similar approach as [8] by searching the description text for certain keywords in, such as *steps to reproduce*, *steps*, *reproducible* and *reproduce*. We build individual models for every project because each project may have different ways for handling duplicate issue reports. For example, The triaging processes for the Mozilla and Eclipse projects are different. For Mozilla, triaged issue reports are not initially triaged to any developer. On the other hand, Eclipse issue reports are triaged at filing to a sub-group of developers based on the project component. Only the developers responsible to a certain component receive the notifications about any newly reported issue report. Based on this triaging process, Eclipse's developers may have a better chance to see a newly reported issue earlier than the developers of Mozilla (though the Eclipse developers might be overwhelmed with many irrelevant reports).

Correlation analysis. We use the varclus package [29] in *R* to display Spearman rank correlations (ρ) between our factors given the large number of factors in our study. Varclus does a hierarchical cluster analysis on the factors using Spearman correlations. If the correlation between two factors is equal to or greater than 0.7, we ignore one of them since we consider a correlation of 0.7 as a high correlation [30, 32]. Figure 4 shows the output of varclus where the red dotted line marks the 0.7 Spearman correlation. Looking at Figure 4, we notice that the factors *Peers Counts*, *Peers CC List*, and *Peers Comments* are highly correlated with a correlation range of $0.85 - 0.91$. Hence, we need to pick just one of these three factors and discard the other two. Based on the results of Varclus (*see* Figure 4), we remove *Peers Comments*, *Peers Counts*,

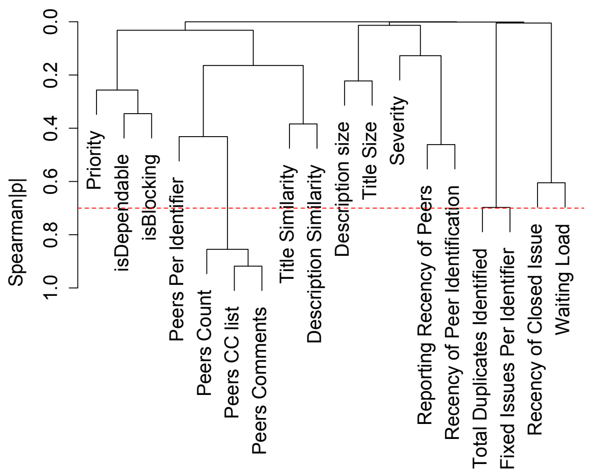


Fig. 4 Spearman hierarchical cluster analysis for the Firefox dataset.

Table 4 Spearman correlations between the three effort measures in the four studied projects.

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

and *Fixed Issues per Identifier* factors due to the high correlation especially in cases such as between *Peers CC List* and *Peers Comments*. After this step, 18 of 21 factors remain.

Modeling technique. We build two classifiers to understand the effort that is needed for identifying duplicate reports. In RQ1, we have three measures of the effort that is needed for identifying duplicate reports. However, the *People Involved* and the *Identification Discussions* factors are highly correlated. Table 4 presents the Spearman correlations between the three effort factors for the studied projects. Hence, we only build two classifiers, one classifier for *Identification Delay* and another one for *Identification Discussions*.

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

Table 5 Criteria for each class based on Identification Delay and Identification Discussions

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

Table 6 The number of duplicate reports from each project that belongs to each class.

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

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

Table 5 summarizes the criteria for each class based on the *Identification Delay* (Slow, Fast) and *Identification Discussions* (Not-Discussed, Discussed). Table 6 presents the distribution of duplicate reports for each class per each project.

We build our classifiers using a random forest [11] classifier. Random forest classifiers are known to be robust to data noise and often achieve a high accuracy in software engineering research [16, 18, 19, 27, 37]. We use the random forest algorithm provided by the *randomForest* R-package [28]. In this study, we train the random forest classifier using 100 decision trees.

An important benefit of a random forest classifier is that it has a mechanism to examine the relative influence of the underlying factors in contrast to other machine learning approaches (e.g., Neural Networks). Thus, we would like to point out that although random forest classifier has been used to build accurate classifier for prediction, our purpose of using a random forest classifier in this paper is not for predicting the needed effort for identifying duplicate issues. Our purpose is to study the important factors that impact the needed effort for identifying duplicate issues. Nevertheless, we first need to determine whether the problem at hand (i.e., understanding duplicate identification effort) is non-random in nature (and hence easy to model), before we can start examining the important (i.e., influential) factors in RQ3. To minimize the risks of over-fitting, we use the 10-fold cross validation technique. A k-fold cross validation divides the data into k equal parts. For each fold, k-1 parts are used as training data to build a classifier and the remaining part is used as testing data.

Evaluation metrics. To measure the performance of our classifiers, we use precision, recall, F-score and area under ROC curve to evaluate our classifier. We describe each measure below:

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

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

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

ROC area value measures the discriminative ability of a classifier in selecting instances of a certain class from a dataset. The ROC metric ranges between 0 and 1. ROC curve shows the trade-off between the true positive rate (also called *Sensitivity*) and false negative rate (also called *Specificity*). The closer the ROC value to 1, the less trade-off exists. The calculation of the ROC area is based on the probabilities of a test dataset for each class. A classifier that randomly guesses the effort class without any training has an ROC area of 0.5. We use the *pROC* R package [45] to calculate the ROC area.

Results. Our classifiers for *Identification Discussions* achieve an average precision between 0.60 to 0.72 and an average recall between 0.23 to 0.93. Table 7 shows the precision, recall, F-score and ROC area for each class per project. Note that the highest precisions for Bugzilla and Firefox are for the *Not-Discussed* class, whereas the *Discussed* class has the highest precision in SeaMonkey. The recall value is highest in SeaMonkey for the class *Not-Discussed*, whereas in Eclipse-platform the highest precision and recall are for the class *Discussed*. A possible reason for the high recall for the Eclipse-platform is that the dataset is more balanced (*see* Table 6 which shows for example that the number of *Not-Discussed* duplicates is twice as many as the number of *Discussed* duplicates in the Firefox dataset). The F-score for the *Non-Discussed* class ranges between 0.69 to 0.78 where as for the *Discussed*

Table 7 Evaluation results for all the studied projects.

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

class it ranges from 0.34 to 0.57. The ROC areas are between 0.68 to 0.72. Such results indicate that the Identification Discussions classifier is better than random guessing (ROC of 0.5) and indicate that our factors can be used to model and understand the discussion effort that is needed for identifying duplicate reports.

Our classifier for *Identification Delay* achieves an average precision between 0.63 to 0.77 and average recall between 0.23 to 0.96. From Table 7, the highest precision and recall for the *Fast* class are for Firefox and SeaMonkey, while both projects have low recall values (0.37 for Firefox and 0.23 for SeaMonkey) for the *Slow* class. The F-score for the *Fast* class ranges between 0.70 to 0.83, whereas for the *Slow* class it ranges from 0.35 to 0.66. Our *Identification Delay* classifier has ROC areas between 0.74 to 0.80 which are better than random guessing (ROC of 0.5). These results provide a sound starting point for modeling the effort that is needed for identifying duplicate reports, and for examining the important (i.e., influential) factors in the models.

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

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

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

Approach. To find the most influential factors to the effort that is needed for identifying duplicate reports, we compute the *variable importance* for each of the factors that are used in the RandomForest classifiers that are built in RQ2. In order to calculate the *variable importance*, we use the same *randomForest* R package. A Random Forest [11] classifier consists of a number of decision trees. Each tree is constructed using a different sample from the original data. Around a third of the data is left out from the construction sample per tree. At the end of each iteration the left out data is used to test the classification and estimate what is called the *out-of-bag* (oob) error [31]. Random Forest follows a fitting process where the target is to minimize the out-of-bag error. To measure the importance of each factor after building the random forest, the values of each factor are permuted one at a time within the training set and the out-of-bag error is calculated again. The importance of a factor is computed by calculating the average difference in out-of-bag error. We compute the factor importance values based on the change in the out-of-bag error. We take the average of the 10-folds classifiers for all the *variable importance* values for each factor. The factor with the highest rank is identified to be the most influential factor for modeling the identification effort.

To compute a statistically stable ranking of the factors, we use the Scott-Knott test [39,43] (with $p < 0.05$). The Scott-Knott test is a statistical multi-comparison cluster analysis technique. The Scott-Knott test clusters the factors based on the reported importance values across the ten folds. The Scott-Knott test divides the factors into two different clusters, if the difference between the two clusters is statistically significant. The Scott-Knott test runs recursively until no more clusters can be created.

Results. ***Factors from the Identifier Experience and Peers dimensions have the largest influence on modeling the needed effort for identifying duplicates.*** Table 8 shows the results from Scott-Knott test for the four projects. The test groups the 18 factors into 11 significant groups on average. The clusters show that the factors from the *Identifier Experience* and *Peers* dimensions are in the top groups of influencing factors for the needed effort for identifying duplicate reports across all the studied projects. For Bugzilla, we find that the Priority factor has a large influence in both effort classifiers.

Table 8 Scott-Knott test results when comparing the importance per factor for the four studied projects. Factors are divided into groups that have a statistically significant difference in the importance mean ($p < 0.05$).

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

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

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

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

Discussion. On the impact of the Priority of an Issue. Our results show that the *Priority* of a report is an important factor only for the Bugzilla project in contrast to the other projects. By looking at the data, we found that 98.6% of the duplicate reports for Firefox have an empty priority value. The high percentages of empty values imply that the *Priority* field is rarely used in Firefox. Even though SeaMonkey and Bugzilla have less empty priority values (83.7% and 85%). SeaMonkey's priority values are barely used, since around 88% of the non-empty priority values at SeaMonkey are set to $P3$ (the default priority). On the other hand, only 54% of the non-empty priority values in Bugzilla are $P3$. Eclipse-Platform dataset does not support empty priority values. But around 91% of the filled priority values at Eclipse-Platform are set to $P3$. These results give us a good reason behind the unique role of the *Priority* factor in the Bugzilla classifiers.

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

5 Discussion

5.1 On the Textual Similarity of Duplicate Issue Reports

Description Similarity is one of the most important factors to influence the effort that is needed for identifying a duplicate report. The majority of the

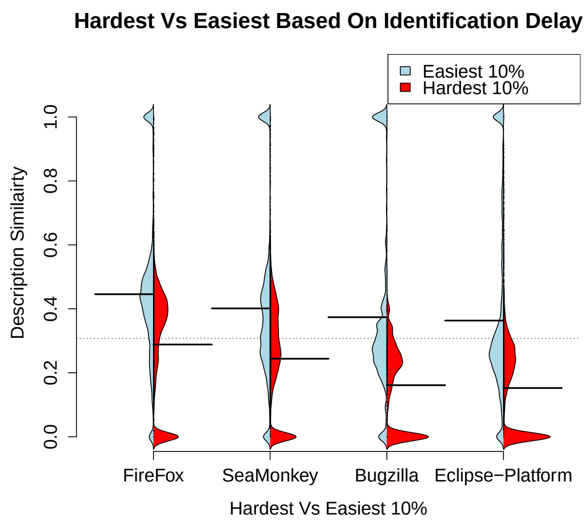


Fig. 5 Description Similarity Beanplot for the Hardest and Easiest duplicate reports in all the studied projects.

automated approaches for duplicate issue reports identification [2, 4, 17, 25, 33, 35, 38, 40, 42] are based on textual similarity (e.g., description similarity). Our results imply that the Description Similarity is one of the most important factors to influence the needed effort for identifying a duplicate report. However, our results imply that the more similar the description, the easier developers can identify the duplicate issue report. Thus, the identified duplicate reports by automated approaches may be the ones that require the least effort to identify. Figure 5 shows a Beanplot [20] of the description similarity distributions for the 10% hardest and 10% easiest duplicate reports for the identification delay metric. The similarity score ranges between 0 (very dissimilar) to 1 (very similar). The closer the score to 1 the more similar the issue reports. Figure 5 shows that the hardest 10% duplicate reports have a lower text similarity median with a peak at the lowest similarity (i.e., no textual similarity), while the easiest 10% have a peak at the highest similarity (i.e., identical text). The same observation holds for identification discussions.

The results in Figure 5 show that duplicates that require more effort to identify tend to be less similar. Such a finding implies that current state of the art automated approaches are likely to miss effort-consuming duplicates.

5.2 On the Difficulty of Duplicate Issue Reports.

We manually examined the top 20 reports for both effort metrics (i.e, identification delay and identification discussions) across the four studied projects (160 issues in total). We observed that these reports correspond to difficult

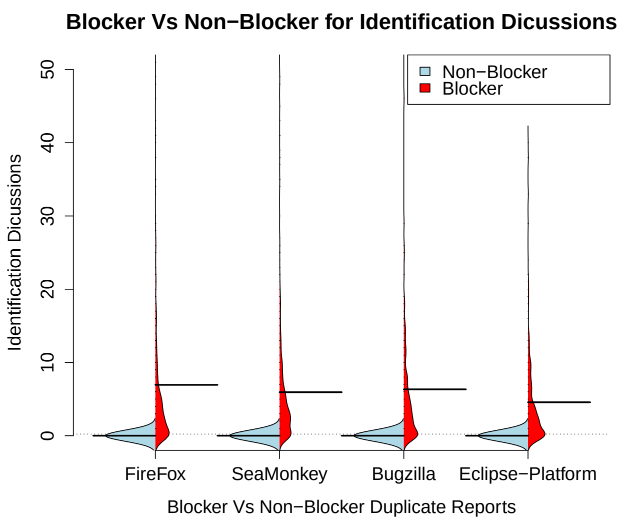
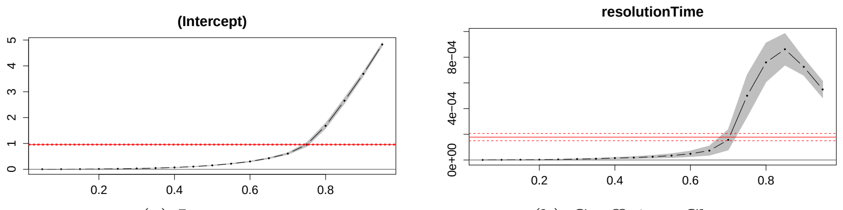


Fig. 6 The Identification Discussions Beanplot of Blocker vs Non-Blocker duplicate reports.



(a) Intercept

(b) Coefficient Change

Fig. 7 Quantile regression Intercept and Coefficient Change for the Resolution Time when modeling the Identification Delay in Firefox.

issues. In particular, many of the reports correspond to issues that take a long time to resolve and around 50% of the corresponding issues are blocker issues. Figure 6 shows the difference between blocker and non-blocker duplicate reports based on identification discussions. To define the blocker issues, we use the *Blocks* field that is provided by Bugzilla. Blocker duplicate reports have a larger identification discussions median in all the studied projects ($p < 0.001$ for Student T-test).

To understand the relationship between duplicate report identification delay and the complexity of an issue (i.e., overall resolution time of an issue), we built a Quantile regression model [3]. The model examines the relation between a group of predictor variables and specific percentiles of the response variable. Quantile regression allows us to understand how some percentiles of the response variable are affected differently by different percentiles of the predictor variables. Such regression is appropriate given our desire to understand the impact of the full range of our metrics. In our case, the response

Table 9 Evaluation results for the global models

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

variable is the *Identification Delay* of a report, and our predictor variable is the overall *Resolution Time* of the issue. Figure 7 shows the Quantile regression results for the Firefox project. The red horizontal lines are the Linear Regression model [34] coefficients with a red dotted confidence interval. The black curves are the Quantile regression coefficients with a gray confidence interval. Figure 7 shows that the resolution time has higher coefficients at higher quantiles, which shows that the complexity of an issue plays a role in the identification delay of duplicate reports. The same pattern holds for the other studied projects.

5.3 On the Generality of Our Observations

To explore the generality of our observations, we built two global models instead of building project specific models as done in Section 4. We built one model for identification delay and another model for identification discussion. To build a global model, we put data from all the studied projects into one dataset.

The goal of the global model is to uncover the generalization of our observations. However, due to the varying sizes of the studied projects, we took random samples of equal size from each project. The size of the random sample from each project is equal to the size of smallest dataset (3,099 issues for Bugzilla). Such an approach to create our dataset for the global model ensures that no data from one specific project can over take the global trends across the studied projects. The accuracies of the two global models are close to the project-specific models (*see* Table 9).

Table 10 presents the results of the Scott-Knott test for the studied factors of the global models. Similar to the project-specific models, the *Total Duplicates Identified* and *Description Similarity* factors rank as the most impacting factors. However, the *Priority* and *Reporting Recency of Peers* factors that are important in the individual models of Bugzilla and Eclipse-Platform (see Section 4) do not rank as one of the top important factors in the global model. If we only built a global model, we would not observe the importance of the *Priority* and *Reporting Recency of Peers*. Such results highlight the value of building project specific models in order to uncover the different ways for handling duplicate issue reports among the studied projects. Our global model highlights the general trends across the projects.

Table 10 Scott-Knott test results when comparing the importance per factor for the global models. Factors are divided into groups that have a statistically significant difference in the importance mean ($p < 0.05$).

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

6 Threats to Validity

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

External validity. Our study is conducted on four large open source projects. While three of the projects (i.e., Firefox, SeaMonkey, Bugzilla) are incubated by the Mozilla foundations, these projects share only 9% of the top developers (the ones who identify over 80% of duplicate reports in total). Our earlier findings (*see* Section 4) highlight as well that each one of these projects follows a different process for managing their issue reports. As usual additional case studies are always desirable. However for the purpose of this paper we do believe that highlighting our raised observations even in a single project is sufficient to raise concerns about future research for duplicate identification.

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

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

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

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

We choose one day as a threshold to determine rapidly identified reports, since we want to ensure globally distributed developers all have a chance to see the issue report. Using more granular time periods (e.g., minutes) to measure the identification delay may introduce more noise into the data because there might be cases that a developer is too busy with other activities to immediately look into a newly filed issue report.

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

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

7 Conclusion

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

1. More than half of the duplicate reports are identified within one day, with no discussion and without the involvement of any people (other than the reporter); nevertheless there exists a small number of reports that are identified as duplicates after a long time, with many discussions and with the involvement of as much as 80 developers.
2. The developer experience and the knowledge about duplicate peers of an issue report, such as description similarity, play an important role in the effort that is needed for identifying duplicates

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

Moreover, our results show that the strong reliance on textual similarity by state of the art duplicate identification approaches will likely lead to the identification of duplicates that developers are already able to identify with minimal effort. While, the duplicate reports that need minimal effort are still worthy to be handled by current automatic identification approaches; our results highlight the need for duplicate identification approaches to put additional emphasis on the issue reports that are more difficult to identify as a duplicate.

References

1. Aggarwal, K., Rutgers, T., Timbers, F., Hindle, A., Greiner, R., Stroulia, E.: Detecting duplicate bug reports with software engineering domain knowledge. In: SANER 2015: International Conference on Software Analysis, Evolution and Reengineering, pp. 211–220. IEEE (2015)
2. Alipour, A., Hindle, A., Stroulia, E.: A contextual approach towards more accurate duplicate bug report detection. In: MSR 2013: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 183–192 (2013)
3. Angrist, J.D., Pischke, J.S.: Mostly harmless econometrics: An empiricist's companion. Princeton university press (2008)
4. Anvik, J., Hiew, L., Murphy, G.C.: Coping with an open bug repository. In: Eclipse 2005: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange, pp. 35–39. ACM (2005)
5. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: ICSE 2006: Proceedings of the 28th International Conference on Software Engineering, pp. 361–370. ACM (2006)
6. Bertram, D., Voida, A., Greenberg, S., Walker, R.: 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, pp. 291–300. ACM (2010)
7. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: Quality of bug reports in eclipse. In: Eclipse 2007: Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange, pp. 21–25. ACM, New York, NY, USA (2007)

8. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 308–318. ACM, New York, NY, USA (2008)
9. Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.: Duplicate bug reports considered harmful really? In: ICSM 2008: Proceedings of the IEEE International Conference on Software Maintenance, pp. 337–345. IEEE (2008)
10. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. the Journal of machine Learning research **3**, 993–1022 (2003)
11. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
12. Cavalcanti, Y.C., Da Mota Silveira Neto, P.A., de Almeida, E.S., Lucrédio, D., da Cunha, C.E.A., de Lemos Meira, S.R.: One step more to understand the bug report duplication problem. In: SBES 2010: Brazilian Symposium on Software Engineering, pp. 148–157. IEEE (2010)
13. Cavalcanti, Y.C., Neto, P.A.d.M.S., Lucrédio, D., Vale, T., de Almeida, E.S., de Lemos Meira, S.R.: The bug report duplication problem: an exploratory study. Software Quality Journal **21**(1), 39–66 (2013)
14. Davidson, J.L., Mohan, N., Jensen, C.: Coping with duplicate bug reports in free/open source software projects. In: VL/HCC 2011: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 101–108. IEEE (2011)
15. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. JAsIs **41**(6), 391–407 (1990)
16. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: ICSE 2015: Proceedings of the 37th International Conference on Software Engineering (2015)
17. Jalbert, N., Weimer, W.: Automated duplicate detection for bug tracking systems. In: DSN 2008: Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, pp. 52–61. IEEE (2008)
18. Jiang, Y., Cukic, B., Menzies, T.: Can data transformation help in the detection of fault-prone modules? In: Proceedings of the 2008 workshop on Defects in large software systems, pp. 16–20. ACM (2008)
19. Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.i., Adams, B., Hassan, A.E.: Revisiting common bug prediction findings using effort-aware models. In: ICSM 2010: IEEE International Conference on Software Maintenance, pp. 1–10. IEEE (2010)
20. Kampstra, P., et al.: Beanplot: A boxplot alternative for visual comparison of distributions (2008)
21. Kanaris, I., Kanaris, K., Houvardas, I., Stamatatos, E.: Words versus character n-grams for anti-spam filtering. International Journal on Artificial Intelligence Tools **16**(06), 1047–1067 (2007)
22. Kanerva, P., Kristofersson, J., Holst, A.: Random indexing of text samples for latent semantic analysis. In: Proceedings of the 22nd annual Conference of the cognitive science society, vol. 1036. Citeseer (2000)
23. Kaushik, N., Tahvildari, L.: 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, pp. 159–168. IEEE Computer Society (2012)
24. Koponen, T.: Life cycle of defects in open source software projects. In: Open Source Systems, pp. 195–200. Springer (2006)
25. Lazar, A., Ritchey, S., Sharif, B.: 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, pp. 308–311. ACM (2014)
26. Lerch, J., Mezini, M.: Finding duplicates of your yet unwritten bug report. In: CSMR 2013: 17th European Conference on Software Maintenance and Reengineering, pp. 69–78. IEEE (2013)

27. Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings. Software Engineering, IEEE Transactions on **34**(4), 485–496 (2008)
28. Liaw, A., Wiener, M.: Random Forest R package. http://cran.r-project.org/web/packages/randomForest/randomForest.pdf
29. Marie Chavent, V.K., Benoit Liquet, J.S.: Variable Clustering. http://svitsrv25.epfl.ch/R-doc/library/Hmisc/html/varclus.html
30. McIntosh, S., Kamei, Y., Adams, B., Hassan, A.E.: An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering pp. 1–44 (2015)
31. Mitchell, M.W.: Bias of the random forest out-of-bag (oob) error for certain input parameters. Open Journal of Statistics **1**(03), 205 (2011)
32. Mockus, A., Weiss, D.M.: Predicting risk of software changes. Bell Labs Technical Journal **5**(2), 169–180 (2000)
33. Nagwani, N.K., Singh, P.: Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs. In: ICAC 2009: Proceedings of the International Conference on Advances in Computing, Communication and Control, pp. 202–207. ACM (2009)
34. Neter, J., Kutner, M.H., Nachtsheim, C.J., Wasserman, W.: Applied linear statistical models, vol. 4. Irwin Chicago (1996)
35. Prifti, T., Banerjee, S., Cukic, B.: Detecting bug duplicate reports through local references. In: PROMISE 2011: Proceedings of the 7th International Conference on Predictive Models in Software Engineering, pp. 8:1–8:9. ACM (2011)
36. Robertson, S., Zaragoza, H., Taylor, M.: Simple bm25 extension to multiple weighted fields. In: CIKM 2004: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, pp. 42–49. ACM (2004)
37. Robnik-Šikonja, M.: Improving random forests. In: Machine Learning: ECML 2004, pp. 359–370. Springer (2004)
38. Runeson, P., Alexandersson, M., Nyholm, O.: Detection of duplicate defect reports using natural language processing. In: ICSE 2007: Proceedings of the 29th International Conference on Software Engineering, pp. 499–510. IEEE Computer Society (2007)
39. Scott, A., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. Biometrics pp. 507–512 (1974)
40. Sun, C., Lo, D., Khoo, S.C., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: ASE 2011: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 253–262. IEEE (2011)
41. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: ICSE 2010: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering, pp. 45–54. ACM (2010)
42. Sureka, A., Jalote, P.: Detecting duplicate bug report using character n-gram-based features. In: APSEC 2010: Proceedings of the Asia Pacific Software Engineering Conference, pp. 366–374. IEEE Computer Society (2010)
43. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Ihara, A., Matsumoto, K.i., Ghotra, B., Kamei, Y., Adams, B., Morales, R., Khomh, F., et al.: The impact of mislabelling on the performance and interpretation of defect prediction models. In: ICSE 2015: Proceedings of the 37th International Conference on Software Engineering (2015)
44. Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: ICSE 2008: Proceedings of the 30th International Conference on Software Engineering, pp. 461–470. ACM (2008)
45. Xavier Robin, N.T., Alexandre Hainard, N.T., Frdrique Lisacek, J.S., Mller, M.: pROC R package. http://cran.r-project.org/web/packages/pROC/pROC.pdf