Duplicate Bug Reports Considered Harmful ... Really?

Nicolas Bettenburg
Saarland University
nicbet@st.cs.uni-sb.de

Rahul Premraj
Saarland University
premraj@cs.uni-sb.de

Thomas Zimmermann
University of Calgary
tz@acm.org

Sunghun Kim
MIT CSAIL
hunkim@csail.mit.edu

Abstract

In a survey we found that most developers have experienced duplicated bug reports, however, only few considered them as a serious problem. This contradicts popular wisdom that considers bug duplicates as a serious problem for open source projects. In the survey, developers also pointed out that the additional information provided by duplicates helps to resolve bugs quicker. In this paper, we therefore propose to merge bug duplicates, rather than treating them separately. We quantify the amount of information that is added for developers and show that automatic triaging can be improved as well. In addition, we discuss the different reasons why users submit duplicate bug reports in the first place.

1. Introduction

In software maintenance, users inform developers via bug reports which part of a software product needs corrective maintenance. For large projects with many users the amount of bug reports can be huge. For example as of April 2008, MOZILLA had received more than 420,000 and ECLIPSE more than 225,000 bug reports. However, not all of them are unique because often users submit reports that describe problems already filed, also called *duplicate reports* or simply duplicates. For MOZILLA about 30% and for ECLIPSE about 20% of all bug reports are duplicates (as reported by Anvik et al. [1]).

A common belief in software development is that entering bug duplicates is bad and therefore considered harmful. Frequent arguments against duplicates are that they strain bug tracking systems and demand more effort from quality assurance teams—effort that could instead be utilized elsewhere to improve the product.

In previous work, we conducted a survey on the quality of bug reports and frequent problems in bug reporting [3,4]. One ECLIPSE developer pointed out the following:

> *"Duplicates are not really problems. They often add useful information. That this information were filed under a new report is not ideal [...]"*

On his blog Alan Page, Director of Test Excellence at Microsoft, lists the following three arguments why "worrying about them [duplicates] is bad" [14].

1. Often there are negative consequences for users who enter duplicates. As a result *they might err on the side of not entering a bug,* even though it is not filed yet.
2. *Triagers are more skilled in detecting duplicates* than users and they also know the system better. While a user will need a considerable amount of time to browse through similar bugs, triagers can often decide within minutes whether a bug report is a duplicate.
3. Bug *duplicates can provide valuable information* that helps diagnose the actual problem.

In this paper, we provide empirical evidence for the third argument and show that duplicates indeed provide additional information. With respect to the first argument, we cannot provide hard evidence of the consequences for users, e.g., gaining a bad reputation. However, we would like to point out that a substantial amount of bugs for MOZILLA start with "Apologies if this is a duplicate". Many users also feel the need to explicitly mention that they did several queries and could not find a similar bug. It is also a frustrating experience when their reports get resolved as duplicates and all provided details simply get discarded as pointed out by a user in our previous survey [4]:

> *"Typically bugs I have reported are already reported but by much less savvy people who make horrible reports that lack important details. It is frustrating to have spent lots of time making an exceptionally detailed bug report to only have it marked as a duplicate because someone (usually not technically qualified) already made a report."*

After presenting the terminology (Section 2) and discussing the data collection (Section 3), we investigate the following two hypotheses:

H1 Duplicate bug reports provide developers with information that was not present in the original report.

H2 The information in bug duplicates can improve automated triaging techniques, i.e., who should fix a bug.

To test the first hypothesis H1, we compare the information in the original reports with duplicates to quantify how much information is new (Section 4). To test hypothesis H2, we build prediction models for automate bug triaging [2] and compare the results of models trained with and without duplicate reports (Section 5). Next, we present several reasons why users enter duplicates in the first place and support them with examples from ECLIPSE (Section 6). We conclude the paper with a discussion of threats to validity (Section 7), related work (Section 8), and recommendations on how to improve bug tracking systems (Section 9).

2. Terminology

Often when a problem is encountered with a software product, a bug report is created in the bug database. This report contains a description of the *failure*[1] and is subsequently used by the developers to correct the *defect*, i.e., the error in the source code. Often several reports are submitted for the same failure or the same defect, which are called *duplicate reports*. In the presence of duplicates, developers choose one bug report as the *master report*. In Section 6 we discuss several factors that lead to the creation of duplicates.

Deciding which reports are duplicates is a tedious task, often referred to as *duplicate detection*. To assist triagers, research proposed to use natural language processing [10, 17], sometimes combined with execution information [19].

Once a duplicate report is identified, it is typically closed and all information is discarded. In this paper, we argue that this is a bad practice: instead of discarding duplicate reports, they should rather be *merged* with the master report. We refer to a master report that is combined with its duplicate reports as *extended master report*. To quantify the amount of value added we use *information items*, which are for example predefined fields (product, component, operating system), attachments, screenshots, or structural elements (stack traces, patches, source code).

3. Data Collection

For the experiments in this paper, we used the bug database of the ECLIPSE project, which is available as an XML export on the web-site of the MSR Mining Challenge 2008 [12]. The database consists of a total of 211,843 bug reports, which have been reported between October 10, 2001, and December 14, 2007. Out of these, 16,511 are master reports with 27,838 duplicate reports.

3.1. Information Items in Bug Reports

Bug reports consist of a *description*, which describes the bug and how to reproduce it, a *title* (also called *summary*), which is a one-line abstract of the description. They also have several predefined fields such as *component, product, priority, affected version, operating system,* and *target milestone* (when a bug is expected to be resolved). In addition, the user who submitted and the developer who is assigned to the bug report are recorded.

Developers and users can add *attachments* to bug reports, which often are *screenshots* or *patches*. To recognize the type of an attachment, we used the UNIX tool "file", which returns the mime-type of a file. Whenever the mime-type of a file starts with "image/", we consider it to be a screenshot. To detect patch files, we defined regular expressions.

The description field of bug reports often contains structural information such as *stacktraces* and *patches*, which help developers to better understand the nature of the bug. Duplicates often contain additional structural information (different in type or content) from what is present in the master reports. We used a tool called infoZilla [5] to detect structural information in bug reports. infoZilla uses a multitude of regular expressions to detect stacktraces and patches. For more information on infoZilla, we refer the reader to our previous work [5].

3.2. Duplicate Reports

Whenever developers detect a duplicate, they resolve the report as DUPLICATE and add a reference to the master report. In the XML export, this reference is saved in the `dup_id` element, which we used to identify duplicates and their master reports. An alternative approach to retrieve duplicates is to scan comments for automatically generated messages such as *"*** This bug has been marked as a duplicate of 42 ***"*. Scanning comments has the disadvantage that it sometimes recognizes incorrect master reports because the messages are not updated when the master report is changed. For ECLIPSE, scanning comments would have added 1,009 false (outdated) links to master reports. In contrast, the references in the `dup_id` element of the XML export are always up-to-date.

Previous research on duplicates encountered the *inverse duplicate problem* [19]: after developers detect two duplicates, they do not always mark the earlier report as master report. Instead, they choose the master report based on the amount of progress that has been made: "whichever bug is further along in the process of getting fixed should *not* be made a duplicate" [16]. In ECLIPSE about 31% of all master reports are newer than one of their duplicate reports. In contrast to research on bug duplicate detection, which defines the master report as the earliest report [10, 17, 19], we decided to use the original master reports as defined by developers. The rationale behind this decision is that we want to quantify the amount of additional information that is hidden in what developers consider (intentionally) as duplicates.

[1] This paper's terminology follows Zeller [23] and Runeson et al. [17].

Another issue when processing BUGZILLA databases for duplicates is the *lack of explicit transitivity* in the duplicate relation. When a bug report A is resolved as duplicate of B and later bug report B is resolved as duplicate of C, bug report A should be marked as a duplicate of C as well. However, in BUGZILLA this is currently realized as chains of duplicates A→B→C. For our experiments we resolved this issue and considered C as the master reports for both A and B. In total this affected 1,880 out of the 27,838 duplicate links; the longest chain of duplicates that we encountered consisted of five links, bug 1523→1555→1568→1570→1619→1667.

A minor issue was the presence of *cycles*, for example, bug report A is a duplicate of B and vice versa. For ECLIPSE we observed only one instance, 29986↔29989. We could manually resolve the cycle based on the developers' comments (bug 29986 was a duplicate of 15307).

4. How much Information Duplicates Add

In this section we quantify and compare the information items for master reports with the ones for bug duplicates. Table 1 consists of the following columns:

- The first column presents all information items that we extracted (see Section 3.1). The items fall in four categories: *predefined fields* such as product and component, *patches*, *screenshots*, and *stacktraces*. Patches and stacktraces are often found in the description field or as separate attachments.

- The second column "Master" lists the average count of each information item in the original master reports, i.e., when bug duplicates are ignored. This count corresponds to a practice that is found in many projects: once duplicates are detected, they are simply closed and all information that they provided is discarded.

- The third column "Extended" lists the average count of each information item in the *extended* master reports. This count would correspond to a practice where bug duplicates are merged with master reports and all information is retained.

- The fourth column "Change" is the difference between "Extended" and "Master" and represents the average number of information items that bug duplicates would add per master report.

In order to quantify the amount of additional *"new"* data for master reports, we counted *unique* items whenever possible. For predefined fields we counted the unique values; for patches the unique files that were patched; for screenshots the number of unique filenames and file sizes.

Table 1. Average amount of information added by duplicates per master report.

	Average per master report		
Information item	Master	Extended	Change*
PREDEFINED FIELDS			
– product	1.000	1.127	+0.127
– component	1.000	1.287	+0.287
– operating system	1.000	1.631	+0.631
– reported platform	1.000	1.241	+0.241
– version	0.927	1.413	+0.486
– reporter	1.000	2.412	+1.412
– priority	1.000	1.291	+0.291
– target milestone	0.654	0.794	+0.140
PATCHES			
– total	1.828	1.942	+0.113
– unique: patched files	1.061	1.124	+0.062
SCREENSHOTS			
– total	0.139	0.285	+0.145
– unique: filename, filesize	0.138	0.281	+0.143
STACKTRACES			
– total	0.504	1.422	+0.918
– unique: exception	0.195	0.314	+0.118
– unique: exception, top frame	0.223	0.431	+0.207
– unique: exception, top 2 frames	0.229	0.458	+0.229
– unique: exception, top 3 frames	0.234	0.483	+0.248
– unique: exception, top 4 frames	0.239	0.504	+0.265
– unique: exception, top 5 frames	0.244	0.525	+0.281

* For all information items the increase is significant at $p < .001$.

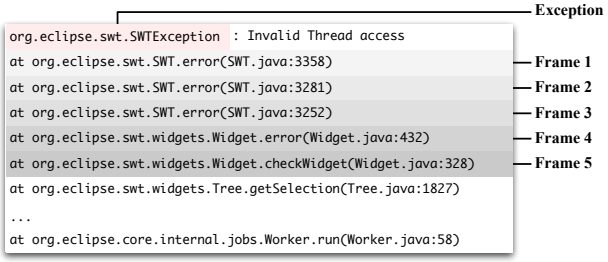

Figure 1. A sample stacktrace.

For stacktraces we counted the number of unique exceptions and unique top n stack frames (for $n = 1, \ldots, 5$). For an example of exception and frames for a stacktrace, we refer to Figure 1.

Coming back to Table 1, every master report contains exactly one operating system (as indicated by the 1.000 in "Master"). When merged with their duplicates, the average number of unique operating systems in extended master reports increases to 1.631 ("Extended"). This means that duplicates could add on average 0.631 operating systems to existing bugs as long as duplicates are not just discarded.

Most duplicates are filed by users who are different from the ones who filed the original master report,[2] which explains the large increase in number of unique reporters in Table 1. A reporter's reputation can go a long way in influencing the future course of a bug report. To quote a respondent from our previous survey on bug report quality [4]:

> *"Well known reporters usually get more consideration than unknown reporters, assuming the reporter has a pretty good history in bug reporting. So even if a "well-known" reporter reports a bug which is pretty vague, he will get more attention than another reporter, and the time spent trying to reproduce the problem will also be larger."*

This suggests that a master report may get more attention if a duplicate filed by a known reporter gets noticed.

Besides reporters, duplicates also provide substantial additional information for operating system, version, priority, and component. We also found that duplicates add on average 0.113 patches and 0.062 additionally patched files per master report. Similarly, bug duplicates add on average 0.143 screenshots.

We compared stacktraces by considering the exception and the first five stack frames. On average, 0.918 stacktraces were found in the duplicate bug reports. Within these, we found on average 0.118 occurrences of additional exceptions in duplicates and 0.281 stacktraces that contained code locations (in the top five frames) that have not been reported before.

Our findings show that duplicates are likely to provide different perspectives and additional pointers to the origin of bugs and thus can help developers to correct bugs. For example, having more stacktraces reveals more active methods during a crash, which helps to narrow down the suspects. To test for statistical significance of our results, we conducted a one-sided paired t-test [18, 20]. For all information items the increase in information items caused by duplicates was significant at $p < .001$.

Overall, the findings of this part support our first hypothesis that *duplicate bug reports provide developers with information that was not present in the original report (H1)* and make a case for re-evaluation of the treatment and presentation of duplicates in bug tracking systems.

5. Improved Triaging with Duplicates

Bug triaging is the assignment of bug reports to a developer in the project who is potentially most suited to fixing the bug. This task is undertaken by a member of the project, referred to as the triager. Often, projects such as ECLIPSE receive a large number[3] of bug reports daily that need to be triaged manually. Anvik et al [2] met with some success at automating this process by training a machine learner that maps the contents of bug reports to the developer to fix the bug. We use bug triaging as an example application to demonstrate the value of duplicates by showing that prediction accuracy can be increased by their inclusion in training the machine learners.

5.1. Data Preparation

In Section 3, we stated that 16,511 master reports and 27,838 duplicate reports are considered for this study. For our bug triaging experiments, we filtered the data to include only those master reports that were fixed by developers who previously fixed at least fifty bugs. This criterion ensured the presence of sufficiently large number of bug reports fixed by a developer in the training set. After filtering, we had 8,623 master reports and 15,377 corresponding duplicates to work with.

Using the master and duplicate bug reports, we formed two datasets: *Master Data* and *Extended Data*. The former dataset only included the master reports (8,623 reports), while the latter included the master and their corresponding duplicate reports (23,990 reports). Also, in the *Extended Data*, we changed the *assigned to* field of the duplicate reports to the developers who fixed the respective master reports. The developers served as output or *classes* to be predicted by the machine learners.

The *title* and *description* fields of the bug reports were used for experimentation. They served as the inputs to the machine learners. Since both fields are entered as raw text in the bug database, we converted them into a word vector using the `StringToWordVector` class in the Weka toolkit [22]. Prior to the conversion to the word vector, we removed all stop words from the reports, but did *not* perform stemming because previous research showed that it had little benefit for bug triaging [8].

We applied a *term frequency inverse document frequency* (TFxIDF) weighting scheme to the word vector. TFxIDF is often used in text-based machine learning methods to prevent biased results [15]. Word duplication in the bug reports would create a bias towards certain developers. To counter this problem, the weight vector is normalized using the Euclidian norm. Second, the distribution of words is heavy-tailed: a word is more likely to be encountered throughout the document once it has first appeared. This is countered by a logarithmic scaling of the weights for single words (*term frequency weighting*). Words that appear in many bug reports are considered to have less distinctive value. To prevent those words from achieving large weights, the *inverse document frequency* is used.

[2] However, this is not always the case as discussed in Section 6.

[3] On average, 94 new bugs were reported daily in the ECLIPSE bug database between Oct. 2001 – Dec. 2007.

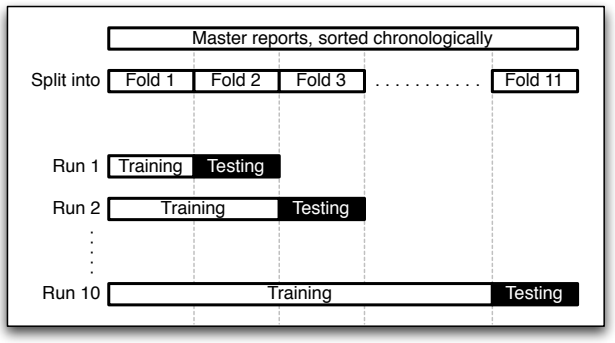

Figure 2. Experimental setup.

5.2. Experimental Setup

We conducted our experiments using a setup that preserves the chronological order of when the bug reports were filed. The setup is illustrated in Figure 2, which shows the bugs in the *Master Data* ordered by their filing date and split into folds. While this setup ensures that only past bugs are used to predict the bugs reported in the future, this is only true for the experiments using the *Master Data*. The arrangement is not preserved for the *Extended Data* because the duplicates are inserted into the folds in which the respective master reports are present; and reports marked as duplicates may have been reported before or after the master.

The experiments were conducted using ten runs (to reduce experimental bias [22]), which amounted to having eleven folds. The folds were generated by first ordering the 8,623 master reports, and then distributing the reports into eleven equally sized folds. As a result, each fold comprised 783 master reports totaling to 8,613 reports. The remaining last 10 of the 8,623 master reports were discarded to ensure equally sized folds.

Once the folds are ready, the experiments can be run iteratively using different folds as training and test sets. In Figure 2 we see that *Run 1* is executed by treating the first fold as a training set and the second, as the test set. In *Run 2*, the training set window is increased to include the first and second folds, while the third fold is used as the test set. This setup is repeated until *Run 10*, where the eleventh fold is the test set and the first ten are used as the training set.

Experiments using the two datasets, *Master Data* and *Extended Data*, were conducted separately. Note that for each set of experiments, the folds from the respective datasets were used to train the machine learners. But only the master reports from the *Master Data* were used as test sets. This allowed comparison of results from models that were trained using different data to make predictions on the same reports in the test folds.

We used two machine learners, namely support vector machines (SVM) and Naïve Bayes, to model the data. The Weka toolkit [22] implementations of both learners were used to conduct the experiments.

5.3. Results and Discussion

The results from the experiments are presented in Table 2. We present results from all ten runs and both machine learners; accuracy is measured by considering the Top 1, Top 3, and Top 5 predictions made by the learners. To elaborate, each learner makes predictions by ranking the output classes or developers for the test bug reports. We considered the top one (meaning exact match), three, and five highest ranked developers and checked if the actual developer was included in the lists. If so, we treated the prediction for the tested bug report as correct. The reported numbers are the percentage of bug reports in the test folds correctly predicted and the All column combines the predictions results from all runs and reports the percentage of correct ones. Predictions using *Master Data* or *Extended Data* are indicated in the third column.

It is no surprise to see that the accuracy increased over nearly all runs for both models. The machine learners had more training data available in the higher numbered runs, which increased their prediction accuracy. The Top 5 measure delivered highest accuracy for each run, also indicated by the average accuracy. On the whole, SVM performed better than Naïve Bayes; the highest accuracy reached was approximately 65%.

More relevant to our study is the comparison of accuracy between using *Master Data* and *Extended Data* as training data. In nearly all cases, the prediction accuracy increased with the use of duplicate bug reports in the training data. The differences between accuracy were as large as up to 9% (Top 5, Run 10 using SVM). We also conducted the McNemar statistical test ($p = .05$) [20] to verify if the accuracy using *Extended Data* data significantly increased. All runs where the McNemar test results were significant are marked in the table with a $(^*)$ alongside the percentage values. Most pairs were significantly different using SVM, while the same was true for fewer pairs using Naïve Bayes. Importantly, all but the Top 1 results using Naïve Bayes in the last column were significant too.

Thus, our results demonstrate that bug reports can be better triaged by considering a larger set of existing bug reports, inclusive of duplicates and support our second hypothesis (H2) that *the information in bug duplicates can improve automated triaging techniques*. Furthermore, our results suggest that other similar empirical experiments can benefit from the use of duplicates too, but this remains to be verified.

Table 2. Percentages of bug reports correctly triaged to developers.

Model	Result	Training	Run 1	2	3	4	5	6	7	8	9	10	All
SVM	*Top 1*	Master	15.45	19.28	19.03	19.80	25.80	26.44	22.09	27.08	27.71	29.12	23.18
		Extended	18.39*	20.95	22.22*	21.46	27.84	28.48	23.37	30.52*	30.78*	30.52	25.45*
	Top 3	Master	32.44	37.42	40.87	39.72	46.10	46.36	38.95	44.70	48.53	47.25	42.23
		Extended	38.70*	42.78*	43.30	39.34	50.83*	49.55*	42.40*	50.32*	50.32	55.04*	46.25*
	Top 5	Master	41.89	46.87	47.38	47.64	54.66	56.96	47.51	52.36	56.58	56.45	50.83
		Extended	47.38*	52.11*	53.00*	51.85*	60.54*	59.90*	51.09*	58.11*	60.28*	65.26*	55.95*
Bayes	*Top 1*	Master	14.81	16.60	17.75	17.75	22.73	21.20	20.56	23.50	27.71	28.22	21.08
		Extended	15.45	17.11	20.56*	18.01	19.80*	19.80	22.99	27.08*	26.82	30.40*	21.80
	Top 3	Master	29.12	32.31	35.12	34.99	40.36	38.06	35.76	43.55	45.59	46.87	38.17
		Extended	36.53*	33.08	38.83*	35.50	39.08	39.08	39.97*	46.23	45.85	50.45*	40.46*
	Top 5	Master	38.44	42.40	45.72	45.21	50.70	47.64	44.06	51.85	54.92	55.17	47.61
		Extended	45.72*	44.70	48.02	43.55	48.91	50.45*	49.43*	55.30*	54.28	58.49*	49.88*

* Increase in accuracy is significant at $p = .05$

6. Reasons for Bug Duplicates

While preparing the data for our experiments, we made the following observations.

1. *For master reports, duplicates are often submitted by the same users.* As Table 3 shows, often a single user submits are large number of bug reports that are duplicates of each other. In the table, column "#bugs corresponds to the number of bugs in a duplicate group and "#users" corresponds to the number of unique users who submitted these reports.

2. *Often duplicates are submitted at the same time (by the same users).* In column "time apart", Table 4 lists the time that the first and last bug report are apart in the corresponding duplicate groups. The table is sorted in ascending order by "time apart" and show that many duplicates are reported at exactly the same time as the original master report.

3. *Some bug reports are exact copies of each other, but not always marked as duplicates.* We grouped bug reports with exactly the same title and description into clusters. In Table 5, we list the largest clusters; the size of each cluster is indicated by the column "#bugs". The column "#dups" shows how many reports have been marked as duplicates. Within a cluster (i.e., same title and description), one would expect that all bugs except for one master report are marked as duplicates; however, this is the case for only few clusters. When bugs are not marked as duplicates (e.g., for the cluster *"4.0 Porting Guide"*), they were reported for different components to reach different development teams.

Based on these three observations, we performed a more detailed analysis on why duplicates are submitted in the first places. Here are the reasons that we could identify.

- *Lazy and inexperience users.* Some users simply are not willing to spend time to search for duplicates. Others are not yet experienced enough with bug tracking.

- *Poor search feature.* In previous work, we conducted a survey on bug report quality, in which several users and developers recommended improvements for the search feature of BUGZILLA [3,4]. One example of bug where the search feature likely failed is #24448 "Ant causing Out of Memory". It was reported by 33 different users over a period of almost 900 days.

- *Multiple failures, one defect.* Sometimes it is not obvious that two failures (error in the program execution) belong to the same defect (error in the program code). For example for bug #3361, a total of 39 duplicates have been reported by the same user, likely a tester. In the case of #3361, the reports have been created automatically because it took the tester only one minute to submit 40 bug reports.

- *Intentional resubmission.* Some users intentionally resubmit a bug, often out of frustration that it has not been fixed so far. For example the duplicate #54603 was submitted more than eight months after the creation of the corresponding master bug #39064. The duplicate started with "I raised this issue several times, but it is still a problem."

- *Accidentally resubmission.* A substantial number of duplicate reports is created by users who accidentally

Table 3. Many duplicate reports are submitted by the same users.

	Master report	#bugs	#users
3361	JCK 1.4 - ICLS - field from outer [...]	40	1
111508	Extension Point and Schema Cleanup	18	2
102615	jaxb code assit	8	1
101189	Web Service Explorer does not work	8	1
66841	Font in preference dialogs is system font	7	1
75450	SWT-mozilla issue	6	1
127636	Fix gmf.tests junit failure due to [...]	6	1
169346	[ErrorHandling] Changes in [...]	6	1
188031	SWT in 3.2.1 have a different [...]	6	1
134973	eclipse.ini is the most sensitive, [...]	6	1

Table 4. Many duplicate reports are submitted approximately at the same time.

	Master report	#bugs	time apart
185582	[Manifest][Editors] Disable code [...]	5	same time
28538	EditorList close selected should [...]	4	same time
96565	At install, prompt user to [...]	4	same time
97797	Includes not found	3	same time
86540	NPE in CProjectSourceLocation	3	same time
142575	[Test failure] Broken HTML links	3	same time
...	...		
...	*245 more master reports*		
...	...		
3361	JCK 1.4 - ICLS - field from [...]	40	1 minute
167895	Error with Expressions and [...]	5	1 minute
171387	Dialog layout fix	4	1 minute
89465	Templates are not being generated	4	1 minute

Table 5. Many bug reports are exactly the same, but not always marked as duplicates.

Title	#bugs	#dups
4.0 Porting Guide	55	1
No version range specified when requiring [...]	14	0
test	13	0
jaxb code assit	13	7
Add keywords to preference pages	10	1
need to adopt ICU4J APIs	10	0
Web Service Explorer does not work	8	7
Convert plugins to use manifest.mf and single [...]	8	1
Using new registry API	8	0
Manifest files and plugin packaging	7	0
should adopt ICU Collator and use new APIs [...]	6	0
Fix gmf.tests junit failure due to domain [...]	6	5
SWT in 3.2.1 have a different behavior as [...]	6	5
eclipse.ini is the most sensitive, undocumented [...]	6	5
SWT-mozilla issue	6	5

clicked the submit button multiple times. For example, bugs #28538 and #96565 each have four confirmed duplicates, which have been submitted by the same users at exactly the same time. For ECLIPSE we found 521 confirmed duplicates that were likely caused by an accidental resubmission, i.e. duplicate reports which had the same title, same description, same product, and same component as a bug report, which was submitted less than ten minutes before - by the same user.

Some of the above reasons for bug duplicates could be easily addressed by better tool support, e.g., by an improved search and a warning when a user is submitting a bug again.

7. Threats to Validity

This section addresses the threats to validity of our study.

Threats to external validity. These threats concern our ability to generalize from this work to general software development practice.

- *Results may not generalize to other projects.* In this study we examined a total of 44,349 bug reports from the ECLIPSE project. Although these reports span a total of 37 sub-projects of ECLIPSE, like ASPECTJ or JDT, they may not be representative of other open- or closed-source software projects. Different domains and software processes may call for alternate handling of duplicate reports.

- *Additional information may also be harmful.* Hypothesis H1 researched in this study postulated the beneficial value of additional information provided by duplicate bug reports to the bug fixing process. However, additional information may distract the developer from defect location and repair.

Threats to internal validity. These threats concern the appropriateness of our measurements, our methodology and our ability to draw conclusions from them.

- *Correctness of data.* We assume that human triagers always assign bug reports to the correct developer capable of resolving the problem and keeping that information up-to date. If this data is incorrect, it could potentially invalidate our results.

- *Obsolete attachments.* When quantifying the amount of additional information provided by duplicate reports, we do not take into account whether an attachment has become obsolete or not. This information is on one hand time dependent and on the other, subjective.

- *Implicit assumptions on the triaging process.* Bug reports, when filed are not initially assigned to a specific developer. Instead new reports are first assigned to a default email address. In most cases, this information is not changed for reports that are resolved as being duplicates. In our studies, we hence assumed that the developer assigned to the master report should also be responsible for the respective duplicate reports. This need not generally to be true.

- *Validation setup.* Although many variants for experimental setups exist such as random splits and cross-validation, we chose a special setup that preserved the chronological order of the bug reports. This setup provided a realistic simulation of bug tracking environments and consequently, more realistic assessment of our experiments. We suspect that even better results can be achieved by using other traditional experimental setups.

- *Chronological order of duplicates.* Due to the inverse duplicate problem described in Section 3.2, the creation of the Extended Dataset violates the setup to strictly predict the future using historical data. However, we expect this to have negligible impact, since we use the Extended Dataset only for training.

- *Biased results.* Developers, who are assigned to a large number of reports, will be substantially more likely to be listed in the Top 5 list. We neutralize this by using a TFxIDF transformation [15] to ensure those developers will not be favored by the machine learners.

8. Related Work

To our knowledge, no other work has empirically studied the value of bug duplicates in software development. Our results indicate that duplicates contain additional information, which could help developers to fix bugs more efficiently. We also found evidence that information from bug duplicates can improve existing techniques to automatically assign developers to bug reports [2, 7, 8].

Other automated triaging techniques include the assignment of locations to bug reports [6], detection of bug duplicates [10, 17, 19], and prediction of effort for bug reports [21]. We are confident that all these approaches would benefit by additionally taking information from duplicate reports into account.

To some extent, our results substantiate the importance of duplicate detection [1, 13], for which research used natural language processing [10, 17]. Duplicate detection is mostly concerned with the discovery of duplicate reports, however, it stops once a duplicate is detected. A common practice in software development is to close duplicate reports and discard the information that duplicates provided. Our results indicate that a better handling of duplicates, e.g., merging them with master reports, would increase the information readily available for developers.

9. Conclusions and Consequences

In this paper, we presented empirical evidence that bug duplicates contain information that is not present in master reports. This additional data is potentially helpful for developers when fixing bugs and can also improve automated triaging techniques such as deciding who should fix a bug. We also discovered that many duplicates are submitted because of shortcomings of bug tracking systems.

Based on the findings in this paper, our recommendations for better bug tracking systems are as follows:

- Provide a feature to *merge bug reports*, so that all information is readily available to developers in one bug report and not spread across many.

- Check for *resubmission of identical bug reports*. These duplicates are easy to catch and could be easily avoided by the bug tracking system.

- Allow users to *renew long-living bug reports* that are still not fixed. Often the only way to remind developers of these bugs is to resubmit them (and thus creating a duplicate report). Providing an easy way to renew, i.e., show that a bug is still in the latest version, also shows what bugs are important to users. In addition one could assign bug reports with an *expiry date*, which is automatically prolonged as long as there is activity for a bug report or it is renewed by users.

- If someone finds a problem already reported in the bug database, bug tracking systems should *encourage users to provide more information*. Often bug writing guidelines such as the one for ECLIPSE [9] stress the importance of searching for duplicates, but do not solicit more information for filed bug reports.

- Improve *search for bug reports*. Most users are willing to spend some time to search for duplicates, but not a lot. Here approaches for duplicate detection will be a valuable addition to bug tracking systems.

While we showed in this paper that bug duplicates contain additional information, it is unfortunate that the data is in separate reports. Merging duplicates into the master report can help developers to find all relevant information in one place. Ideally, users would search for a master report and if they find one, add more information. However, in practice the search feature of bug tracking systems is only

of limited use and does not help users to find master reports [11], which is why building better search functionality is the recommendation that we consider most important. Further, users should be encouraged to add more information to existing bug reports.

To learn more about our work on bug tracking systems and mining software archives, visit

`http://www.softevo.org/`

Acknowledgments. Many thanks to Sascha Just and the anonymous ICSM reviewers for valuable and helpful suggestions on earlier revisions of this paper.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy. Coping with an open bug repository. In *eclipse '05: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39. ACM, 2005.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 361–370. ACM, 2006.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. Quality of bug reports in Eclipse. In *eclipse '07: Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, pages 21–25. ACM, 2007.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *FSE '08: Proceedings of the 16th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. ACM Press, November 2008. To appear.

[5] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting structural information from bug reports. In *MSR '08: Proceedings of the 5th Working Conference on Mining Software Repositories*. ACM, 2008.

[6] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. In *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 105–111. ACM, 2006.

[7] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1767–1772. ACM, 2006.

[8] D. Cubranic and G. C. Murphy. Automatic bug triage using text categorization. In *SEKE '04: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, pages 92–97, 2004.

[9] E. Goldberg. Bug writing guidelines. https://bugs.eclipse.org/bugs/bugwritinghelp.html. Last accessed March 2008.

[10] L. Hiew. Assisted detection of duplicate bug reports. Master's thesis, The University of British Columbia, Vancouver, Canada, May 2006.

[11] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Computer Society, September 2008. To appear.

[12] S. Kim. MSR Mining challenge 2008. http://msr.uwaterloo.ca/msr2008/challenge/, 2008. Last visited March 2008.

[13] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *VL/HCC '06: Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 127–134. IEEE Computer Society, 2006.

[14] A. Page. Duplicate bugs. http://blogs.msdn.com/alanpa/archive/2007/08/01/duplicate-bugs.aspx, 2007. Last visited April 2008.

[15] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *ICML '03: Proceedings of the Twentieth International Conference on Machine Learning*, pages 616–623. AAAI Press, 2003.

[16] S. Richardson. Screening duplicate bugs. http://www.mozilla.org/quality/help/screening-duplicates.html, 2008. Last visited March 2008.

[17] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 499–510. IEEE Computer Society, 2007.

[18] S. Siegel and N. J. Castellan, Jr. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, second edition, 1988.

[19] 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 '08: Proceedings of the 30th International Conference on Software Engineering*. ACM, 2008.

[20] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, second edition, 2004.

[21] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007.

[22] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, second edition, 2005.

[23] A. Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, October 2005.