2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-7281-9630-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/SANER50967.2021.00084

# Characterizing Duplicate Bugs: An Empirical Analysis

Berfin Kucuk
Udemy
Istanbul, Turkey
berfin.kucuk@udemy.com

Eray Tuzun
Bilkent University
Ankara, Turkey
eraytuzun@cs.bilkent.edu.tr

***Abstract*—Bug handling is an essential part of the software development process. Ideally, in a bug tracking system, bugs are reported, fixed, verified, and closed. In some cases, bugs have to be reopened mostly due to an incorrect fix. However, instead of reopening the existing bug report, users may submit a new report on a previously reported bug, which causes duplicate bug reports. Additionally, users might report duplicate bugs if they are unable to reopen the previously reported bugs due to the bug being unresolved (i.e., in progress) and when they miss previously reported bug reports. These duplicate bug reports may cost extra maintenance efforts in triaging and fixing bugs.**

**There have been several studies on characterizing reopened bugs and duplicate bug reports, however, to the best of our knowledge, there has been no prior work on understanding the dynamics of their intersection, which is *missed reopen* bugs. Our study is based on analyzing the differences between duplicate and non-duplicate bugs, and further categorizing the duplicated bugs. In this regard, we categorize duplicate bugs according to their creation time with respect to their master's resolution status as Master-Unresolved bugs and Master-Resolved (Missed Reopen bugs) to distinguish their properties. We compare these two different types of bugs in terms of various aspects such as their relationships to their master bugs, bug surface time, bug fix time, bug's severity, and the number of users involved. We perform case studies using the Eclipse and Mozilla projects' bug repositories that include more than 165,500 and 394,000 bug reports respectively.**

***Index Terms*—duplicate bug reports; reopened bugs; characterization study; bug management**

## I. Introduction

Software bugs are inevitable due to the complexity of software systems. With a bug tracking system like Bugzilla[1], testers and end-users can report bugs while developers can track and triage bugs. Ideally, bugs are reported, fixed, verified, and closed. In some cases, bugs have to be reopened due to reasons such as unclear descriptions given by the bug reporter or an incorrect fix. In other cases, due to the uncoordinated process of bug reporting, the same bug might be reported more than once. The triager will detect if a bug report is a duplicate; if so, the triager will mark this report as a **duplicate** report and the first report as the **master** report [1]. The remainder of the bugs will be addressed as **unique** bugs. Duplication of bugs in software projects might cause a significant negative impact in terms of time spent on these bugs [2], [3].

[1]https://www.bugzilla.org/

There have been several studies that investigate duplicate bugs [3], [4], [5] and reopened bugs [6], [7] separately. However, to the best of our knowledge, there has been no prior work on understanding the dynamics of their intersection. In this study, we categorize duplicate bugs to show that not all duplicates are the same and they have distinguishable characteristics. We propose to categorize duplicate bugs according to their creation (reported) time with respect to their master's resolution status. The duplicates that have been submitted before the resolution of their master bugs are referred to as Master-Unresolved and those that have been submitted after the resolution of their master bugs are referred to as Master-Resolved (Missed Reopen Bug). The expected behaviour for handling the duplicates of Master-Resolved bugs is reopening the master of the corresponding bug instead of opening a new one. On the other hand, the expected behaviour for the duplicates of Master-Unresolved bugs is not being opened at all since the master bug is still active. Throughout the study, we will refer to the Master-Unresolved bugs as Category 1, and to Master-Resolved (Missed Reopen bugs) as Category 2 interchangeably. To quantitatively characterize the different categories of bugs, we define a new metric, Duplicate Bug Surface Time (DBST), that represents the time difference between the reported time of a duplicate bug and the reported time of its master. We also define two more measures; severity that indicates the degree of impact that a bug has on the system, whereas the number of users involved indicates the perceived importance of a bug.

In this study, we will be quantitatively analyzing non-duplicated (unique) bugs and duplicate bugs (Category 1 and Category 2) in light of the following research questions (RQ):

**RQ1**: How do Category 1 and Category 2 bugs differ?

- **RQ1.1** How do Category 1 and Category 2 differ in terms of **the bugs' severity**?
- **RQ1.2** How do Category 1 and Category 2 differ in terms of **the number of users involved**?
- **RQ1.3** How do Category 1 and Category 2 differ in terms of **duplicate bug surface time**?
- **RQ1.4** How do Category 1 and Category 2 differ in terms of **their time distribution over their life cycle**?

**RQ2**: How do Duplicate and Non-duplicate bugs differ?

978-1-7281-9630-5/21/$31.00 ©2021 IEEE
DOI 10.1109/SANER50967.2021.00084

**RQ2.1** How do Duplicate and Non-duplicate bugs differ in terms of **the bugs' severity**?

**RQ2.2** How do Duplicate and Non-duplicate bugs differ in terms of **the number of users involved**?

The rest of the paper is organized as follows. Section II presents some background information on the typical life cycle of a bug, the characterization of bugs in general, reopened bug reports and duplicate bug reports. Section III presents our methodology for analyzing our research questions. Section IV describes the evaluation setup and Section V presents our results on the Eclipse and Mozilla datasets. Section VI presents threats to validity and finally, Section VII concludes the paper.

## II. Background and Related Work

In this section, we first analyze the typical life cycle of a bug, and then we discuss bug characterization studies. Since we analyze the dynamics of duplicate bug reports which should get reopened instead, we also provide background information on duplicate bug reports and reopened bugs as well. In this regard, we summarize the related work in different subsections.

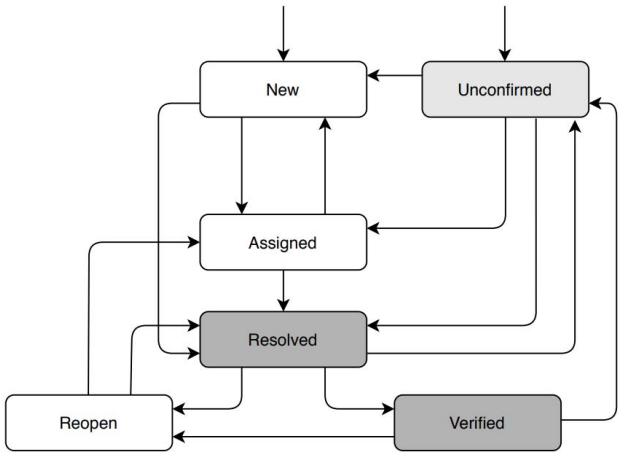



Fig. 1: Typical bug life cycle.

### A. Bug Life Cycle

A bug life cycle describes the workflow of a bug from its creation to its resolution [8]. Figure 1 shows the default workflow of Bugzilla[2] that both Eclipse and Mozilla use as a bug tracking system. The vertices represent the states and the edges represent the transitions between each state. When a bug is submitted, its state is either *New* or *Unconfirmed*. A new bug from a user who has permission to create a bug directly to the system or a product without an *Unconfirmed* state is registered as *New* while others are registered as *Unconfirmed*. The bug status changes from *Unconfirmed* to *New* whenever the bug is confirmed or receives enough votes. If a developer takes possession of a bug that is *New* or *Unconfirmed*, its status changes to *Assigned*. When the bug is successfully fixed by a developer, then its status changes to *Resolved*. When the developer stops working on the bug before reaching a solution, its status becomes *New*. Unconfirmed and new bugs can also move to the *Resolved* state directly when the bug is not valid or when the bug is corrected voluntarily before it is assigned to someone. Finally, when the resolved bug is verified by the quality assurance (QA) team, its status changes to *Verified* but if the QA team is not satisfied with the solution or the bug reoccurs, its status changes to *Reopen*.

[2]https://www.bugzilla.org/docs/3.6/en/html/lifecycle.html

### B. Characterization of Bugs

There are studies that characterize bugs from several perspectives. Tan et al. [9] manually investigate bugs in three dimensions: root causes, impacts, and components. Their findings show that semantic bugs are the dominant root cause that occur due to the inconsistencies with the requirements. Cotroneo et al. [10] investigate the characteristics of bugs from the bug manifestation perspective. They report the most relevant trigger types, their occurrence times, and their relation with complexity and fixing time. Results of Destefanis et al. [11] show that the more polite developers in the communication process are, the less time it takes to fix an issue. Khattar et al. [12] conduct a characterization study on regression bugs, i.e., bugs in a feature that was working earlier. Their findings include that more than half of the regression bugs are assigned a high priority and 50% of the regression bugs are closed within 8 days.

### C. Reopened Bug Reports

The work by Zimmermann et al. [7] characterizes reopened bugs in the Microsoft Windows operating system project. They focus on factors related to bug report edits and relationships between people involved in handling the bug. Shibab et al. [13] predict which bugs will be reopened in Eclipse using metrics related to work habits, bug reports, bug fix, and the team as input to a prediction model. Souza et al. [14] show that the bug reopening rate of versions developed in rapid cycles was about 7% higher. Mi et al. [6] show that over 93% of reopened bugs have severity level more than normal (including normal, major, critical, and blocker levels). They also conclude that the resolution duration for a reopened bug runs up to 2.14 times than normal ones.

### D. Duplicate Bug Reports

There have been few prior works on analyzing characteristics of duplicate bugs. Cavalcanti et al. [4] conclude that features like staff size, project lifetime, and the number of bug reports are not significant factors for duplication while features such as the submitter's profile and the number of submitters influence duplication. Li et al. [5] show that not all duplicate bug reports are value-neutral. They claim that different types of duplicates need different costs to utilize and play different roles in issue-resolving. Davidson et al. [3] investigate duplicate bug reports in free/open software projects. Most open source projects allow users to participate

by reporting bugs that lead to more duplicate bug reports. They claim that duplicate bug reports are a problem for especially medium-sized projects, which struggle with a large number of submissions without the resources of larger projects. In our study, we categorize duplicates and analyze if they are different in terms of their severity, the number of users involved, surface time, and their time distributions. In this regard, our study is different from the previous characterization studies because we categorize duplicates according to the resolution of their master bug and analyze the relationship with their master bug along with their quantifiable features to see if categories of duplicate bugs are different.

Several approaches have been introduced to detect duplicate bug reports automatically in order to reduce the negative impact of duplicate bugs and triaging efforts. An earlier approach uses information retrieval (IR) systems to retrieve information from document repositories and to process the text in the reports with natural language processing (NLP) techniques to detect duplicate bug reports [15]. Their evaluation shows that about 2/3 of the duplicates can be detected using the NLP techniques. Sun et al. [1] propose a retrieval function (REP) to measure the similarity between the two reports. REP utilizes not only the information available in a bug report including summary and description fields but also non-textual fields such as product, component, version, etc. Hindle et al. [16] propose the Continuously Querying approach which helps users to find duplicate bug reports as they type in their bug report. It has the potential to prevent duplicate before they occur in 42% of cases by creating a simple information retrieval model. The work by Nguyen et al. [17] introduces a duplicate bug report detection approach that takes advantage of both an IR approach and topic-based features. He et al. [18] propose an approach based on Dual-Channel Convolutional Neural Networks (DC-CNN). In this approach, bug report pairs are fed to a CNN model to capture the correlated semantic relationships between bug reports. They use the association features to classify whether a pair of bug reports are duplicate or not. The work by Wang et al. [19] presents an approach that further involves execution information along with natural language information. However, generally, execution traces might not be available in bug reports since only a small percentage of bug reports (0.83%) contain this information [20]. As stated in Bettenburg et al.'s work [21], additional information provided by duplicates helps to resolve bugs quicker. Thus, many duplicate bug report detection algorithms add links to duplicates [20].

In our study, we do not propose any technique to detect or prevent duplicate bug reports. Instead, we analyze their characteristics along with the comparison of different duplicate bug categories in the context of software projects. In this regard, our empirical study is important to understand the issue and define new perspectives for solutions. For example, duplicate bug detection algorithms might prioritize features that we found to be statistically significant or algorithms might be improved by calibrating their duplicate detection by considering different categories.

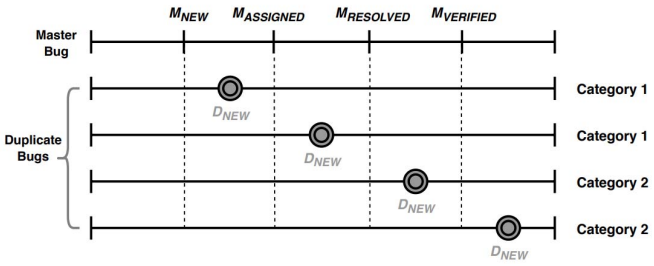



Fig. 2: Timeline of duplicate bug categories regarding their master bugs.

## III. Methodology

In this section, we describe our methodology to analyze duplicated bugs. In Section III-A, we categorize duplicate bugs according to the resolution time of their master bugs. Then, we present our methodology to analyze our research questions in separate sections.

### A. Categorizing Duplicate Bugs

In our study, there are two different categories regarding duplicate bug reports. Figure 2 shows the categories of duplicate bug reports according to the timeline of their master bugs. The top line indicates the timeline of a master bug. For simplicity, only the first resolution cycle of the master bug is taken into account. The time stamps on that line: $M_{NEW}$, $M_{ASSIGNED}$, $M_{RESOLVED}$, and $M_{VERIFIED}$ indicates the timestamps of the states of the potential life cycle of a master bug from left to right. For the sake of simplicity, states of the bug other than *New*, *Assigned*, *Resolved*, and *Verified* are not shown in the figure. The lines below the master bug line are used to indicate when a duplicate bug is created as $D_{NEW}$ relative to the state of its master bug. In this regard, we categorize the duplicate bugs as follows:

***Category 1 (Master-Unresolved): Duplicates are submitted when masters are not resolved yet.*** As seen in Figure 2, the first two lines below the master bug line show duplicate bugs which belong to Category 1 duplicates, in which a duplicate bug is created as *New* **before** its master bug is resolved. These duplicate bugs might get created before or after their master bugs are assigned which does not affect the category they belong to. Since these Category 1 duplicates are submitted when masters are not resolved yet, we name them Master-Unresolved duplicates.

***Category 2 (Master-Resolved or Missed Reopen): Duplicates are submitted when masters are resolved.*** As seen in Figure 2, the last two lines in the figure show duplicate bugs which belong to Category 2, in which a duplicate bug is created as *New* **after** its master bug has been resolved. In this regard, these duplicate bugs might get created before or after the master bug has been verified which does not affect the category. Since these Category 2 duplicates are submitted when masters are resolved, we name them Master-Resolved duplicates. However, these Category 2 duplicate bugs should

get reopened rather than creating a duplicate so throughout the paper, we refer to these duplicates as Missed Reopen duplicates.

To summarize, the duplicate bugs which are created **before** resolution of their master bugs belong to Master-Unresolved *Category 1* duplicates while others which are created **after** resolution of their master bugs belong to Missed Reopen *Category 2* duplicates.

### B. Severity of Different Bug Types

In this section, we analyze the severity of bugs and how it differs among duplicate bugs and between master and unique bugs. The documentation of Eclipse Bugzilla[3] describes the severity types as follows, which is shown from highest to lowest.

- ***Blocker:*** The bug that blocks development and/or testing work.
- ***Critical:*** Crashes, severe memory leak, loss of data due to the bug.
- ***Major:*** Major loss of functionality due to the bug.
- ***Normal:*** Regular bug and loss of functionality under specific circumstances.
- ***Minor:*** Minor loss of functionality due to the bug.
- ***Trivial:*** The bug does not cause any loss of functionality, problems such as misspelled words.
- ***Enhancement:*** The request for enhancement.

In this context, bugs with severity higher than *normal* can be considered as bugs that prevent the system from working properly. For simplicity, we refer to these bugs (i.e., *blocker*, *critical* and *major*) as **severe** bugs. We calculate the ratio of severe bugs in the master of duplicate bugs and compare with the ratio in unique bugs to understand if the severity of a bug affects the bug to be duplicated. Also, we calculate this ratio in the context of two different categories of duplicate bugs. For statistical testing, starting from highest to lowest, we scale the severity from 6 to 0. These ordinal data are used in the Mann-Whitney non-parametric test [22].

The severity of bugs might be changed during the bug life cycle as a result of further evaluation of the bug. In this regard, we take the re-evaluated last status as the current severity of the bug.

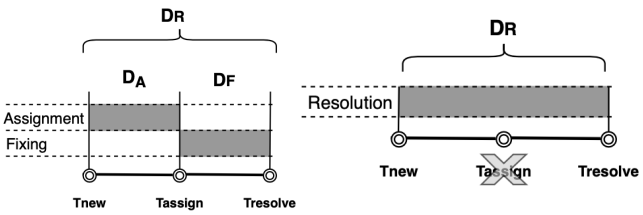
### C. Number of Users Involved in Different Bug Types

As in most of the bug tracking tools, in Bugzilla, a Carbon Copy (CC) list is used to add people to a mailing list that notify them when the bug has an update.[4] The reporter can add users to the CC list and also other users can add themselves to the CC list if they wish to be notified when there is a change in the bug status. Also, it is possible to customize what they want to be notified about. For example, rather than being notified for every change in the bug, they can be notified when the bug is resolved or verified, etc.

In this context, we believe that the CC list size can be used to understand the number of users involved in a bug. The number of users should indicate the perceived importance of a bug. The more users involved in a bug, the more important that bug is for the project. Therefore, we calculate the average number of unique users in the CC list to understand if there is a relationship between the number of users involved and a bug being duplicated. Besides, we also compare the average number of CC list users in Master-Unresolved duplicate bugs (Category 1) and Missed Reopen duplicate bugs (Category 2).

[3]https://wiki.eclipse.org/Eclipse/Bug_Tracking

[4]https://www.bugzilla.org/docs/3.0/html/userpreferences.html




(a) Assigned duplicate timeline. (b) Unassigned duplicate timeline.

Fig. 3: Timelines of duplicate bugs

### D. Duplicate Bug Surface Time (DBST)

*DBST* represents the time difference between the reported time of a duplicate bug and the reported time of its master bug. Mathematically, it refers to notations in Figure 2 and depicted by Equation 1.

$$DBST = D_{NEW} - M_{NEW} \tag{1}$$

With this metric, we can understand how far the surfacing of a duplicate bug is from the creation of its master. We measure the average duplicate bug surface time (*DBST*) of each category.

### E. The Time Distribution of Duplicate Bugs

Duplicate bugs put extra overhead on software projects, as the effort and the cost of managing duplicate bugs can be time-consuming. Thus, with the measurement of time distribution, we analyze where most of the time is spent.

We split the life cycle of a duplicate bug into intervals as demonstrated in Figure 3a and Figure 3b. The time points and durations in Figure 3 are defined similar to in the work of Mi et al. [6] which were used for timeline of reopened bugs.

A bug might be reassigned more than once, before it gets to the right developer who resolves the bug as a duplicate. In this case, we choose the time of the last assignment as $T_{assign}$. In addition, as seen in Figure 3b, a duplicate bug might get resolved as a duplicate without any assignment. In this case, we assume all of the time is spent in the resolution stage.

We use the following metrics to measure the time distribution of duplicate bugs:

- **Duration of Assignment in Days ($D_A$):**
  The interval between a bug being opened ($T_{new}$) and its assignment ($T_{assign}$).
- **Duration of Fixing in Days ($D_F$):**
  The interval between a bug's assignment ($T_{assign}$) and it being resolved as duplicate ($T_{resolve}$).

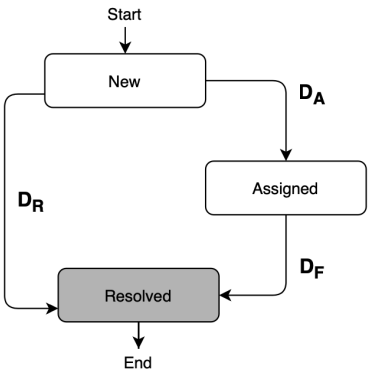



Fig. 4: Simplified typical workflow of a duplicate bug.

- **Duration of Resolution as Duplicate in Days ($D_R$):** The interval between a bug being opened ($T_{new}$) and it being resolved as duplicate ($T_{resolve}$).

## IV. Evaluation Setup

In this section, we first describe the datasets that we have analyzed, then present the results of our study regarding our research questions.

### A. About Dataset

We investigate our approach on two different open source projects: Eclipse and Mozilla. Eclipse[5] is an open source project which is an extensible multi-language software development environment. Mozilla Firefox[6], or simply Mozilla, is not only a free and open source web browser but also a framework for building cross-platform applications. Eclipse and Mozilla use Bugzilla[7] as a bug tracking software. Both Eclipse and Mozilla have cross-platform (i.e., Mac OS X, Linux, Windows, Solaris) usage and they are widely used. Thus, both Eclipse and Mozilla have sufficient bug report history to enable in-depth analysis.

In this study, we used Lamkanfi et al.'s [23] bug tracking dataset, filtered to contain only genuine bugs (i.e., no feature requests and only resolved bugs) within the whole bug-triage life cycle. Eclipse Bugzilla[8] dataset includes 165,547 bugs reported and Mozilla Bugzilla[9] dataset includes 394,878 bugs reported. However, these datasets do not keep the relationship between duplicate bugs and their master bugs. Thus, we extract the master bug of each duplicate bugs using HTML parsing from their websites. Eclipse and Mozilla datasets include 26816 (16.2%) and 91133 (23.1%) duplicates respectively. The resulting new dataset with duplicate information along with the implementation of the tools in this study is available online[10].

[5]https://www.eclipse.org
[6]https://www.mozilla.org/
[7]https://www.bugzilla.org
[8]https://bugs.eclipse.org/bugs/
[9]https://bugzilla.mozilla.org
[10]https://zenodo.org/record/4114096

### B. Data Preprocessing

Before running a quantitative analysis on questions being studied, we first processed our data to prevent potentially misleading results. We detected and removed noisy and unreliable data as follows. First of all, unconfirmed duplicates are removed from the datasets. These bug reports are submitted by the users of Eclipse and Mozilla and are never confirmed before being resolved as duplicate. Since these duplicates are detected at the very first stage of their life cycle and reports submitted by users are not controllable nor reliable, we believe that we reach more genuine results by removing them. In this context, the number of discarded duplicates is 52 (0.2%) and 56304 (62%) in Eclipse and Mozilla respectively. Most of the duplicates of Mozilla are discarded in this process. The possible reasons for this might be a better duplicate bug detection of Mozilla in earlier stages or its users' inattention on previously reported bugs. Additionally, in our study, we consider duplicate bugs that experience the typical workflow in Figure 4. In other words, we discarded duplicates whose time distributions cannot be extracted as in Figure 4 such as reopened duplicates and duplicates that have been assigned directly before reaching *New* state. In Eclipse and Mozilla, corresponding discarded bugs are 3226 (12.03%) and 2050 (2.3%) respectively. Then, we discarded 505 (1.9%) duplicate bugs in Eclipse, 3909 (4.3%) duplicate bugs in Mozilla whose master bugs are not in the dataset. Considering master bugs, we discarded 1629 (6.1%) duplicate bugs in Eclipse, 3618 (4.0%) duplicate bugs in Mozilla whose master bugs are not confirmed nor resolved yet because it is key to categorize our duplicate bugs. We also discarded duplicate bugs that have been reported before their master bugs have been reported which conflict with our master bug definition 4671 (17.4%) bugs in Eclipse, 7078 (7.8%) bugs in Mozilla. In the end, we analyzed our research questions with 16733 (62.3% of all duplicates) duplicate bugs of Eclipse and 18174 (20.0% of all duplicates) duplicate bugs of Mozilla.

## V. Results

In this section, we analyze the results of the proposed research questions. Both RQ1.1 and RQ2.1 analyze the severity of different bug types. Thus, we present them in the same section. Likewise, we present the results of RQ1.2 and RQ2.2 that analyze the number of users involved in the same section. The percentage of duplicate bug categories in Eclipse is 57% Master-Unresolved (Category 1) duplicates and 43% Missed Reopen (Category 2) duplicates. In Mozilla, 79% of them are Master-Unresolved duplicates and 21% of them are Missed Reopen duplicates. In the following, we present the results per each research question.

**RQ1.1 - RQ2.1: Severity of Different Bug Types**

We believe that bugs with severity higher than normal prevent the system from working properly. Thus, we calculate and compare the ratio of these severe bugs for different bug types. Both in Eclipse and Mozilla, the difference of severity between two duplicate categories is found to be statistically insignificant at the 5% level ($p$ = .161 in Eclipse, $p$ = .107

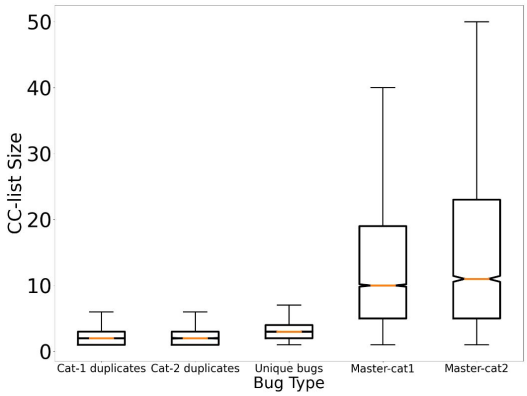



Fig. 5: Distribution of CC list size for each bug type in Mozilla.

TABLE I: Average number of CC list users in each bug type.

| | Median of CC List Size | |
|---|---|---|
| **Bug Type** | **Eclipse** | **Mozilla** |
| Category 1 Duplicates | 1 | 2 |
| Category 2 Duplicates | 1 | 2 |
| Masters of Category 1 Duplicates | 4 | 10 |
| Masters of Category 2 Duplicates | 5 | 11 |
| Unique Bugs | 1 | 3 |

in Mozilla). The reason might be that severity of a duplicate bug does not mean anything for the development team once they realize it is the duplicate of another one so it is not re-evaluated.

Besides, the ratios of severe bugs in unique bugs are only 11.2% and 19.9% in Eclipse and Mozilla respectively. Considering master bugs, these ratios are 20.5% in Eclipse and 31.6% in Mozilla. These differences are found to be statistically significant ($p < .001$) for both datasets. In this regard, the distinction between unique and duplicated bugs is clear when the severity is taken into account, so severity can be one of the metrics to decide if the bug will be duplicated.

**RQ1.2 - RQ2.2: The Number of Users Involved using CC list**

To measure how many users are involved in different bug categories, we use the number of users in the CC list as our metric. Figure 5 shows the CC list size as a notched box plot. The notches indicate a confidence interval around the median, therefore when two notches do not overlap, there is strong evidence (95% confidence) that their medians differ [24]. For better data visualization, outliers have been removed from all notched box plots. As seen in the plot, the numbers of users in the CC list have skewed distribution so we use the median to show the central tendency as in Table I. Medians of CC list size are low in duplicates compared to the master bugs as expected because duplicates are resolved relatively faster. Theoretically, master bugs refer to actual bugs since they are not closed as duplicates and continue their life cycles. In this regard, we see that more users are involved in masters of Missed Reopen (Category 2) duplicates than masters of Master-Unresolved (Category 1) duplicates. This difference is found to be statistically significant at the 5% level ($p < .001$) for both datasets. Also, we see a drastic difference between duplicated bugs (i.e., masters) and non-duplicated (i.e., unique) bugs. In Eclipse and Mozilla, medians of unique bugs have 1 and 3 users in their CC list respectively. In other words, the number of users in unique bugs are significantly less than master bugs ($p < .001$) for both datasets. Thus, we can conclude that bugs that have duplicates tend to have more users in their CC list. This analysis might be a guide to develop new duplicate bug report detection algorithms.

TABLE II: Median of DBST (in days) in each duplicate category.

| | DBST Median | |
|---|---|---|
| **Bug Type** | **Eclipse** | **Mozilla** |
| Category 1 Duplicates | 24.00 | 33.35 |
| Category 2 Duplicates | 90.52 | 102.98 |

**RQ1.3 Duplicate Bug Surface Time (DBST)**

We also investigate Duplicate Bug Surface Time (DBST) which indicates the time difference between the surfacing of a duplicate bug and the creation of its master. DBSTs of bugs have skewed distribution so we use the median to show the central tendency. As seen in Table II, considering DBST, the median of Missed Reopen (Category 2) duplicates are significantly higher than Master-Unresolved duplicates. This difference is found to be statistically significant at the 5% level ($p < .001$) for both datasets. Median DBSTs have approximately 1 to 4 ratio in Eclipse and 1 to 3 ratio in Mozilla which correspond to Master-Unresolved and Missed Reopen duplicates respectively. Since duplicate bugs in Missed Reopen (Category 2) duplicates surface in the later part of the life cycle of its master bug, this ratio is as expected.

**RQ1.4 Time Distributions**

With the measurement of time distribution, we analyze where most of the time is spent along with their comparison within different categories of duplicate bugs. We show results of Eclipse and Mozilla in different diagrams as in Figure 6 and Figure 7 respectively. As seen in the figures, the average direct resolution duration of Master-Unresolved (Category 1) duplicates are more than the average direct resolution duration of Missed Reopen (Category 2) duplicates. For example, in Eclipse, the average direct resolution duration $D_R$ is 29.30 days in Category 1 and 18.18 days in Category 2. However, since the time distribution of duplicates is highly skewed, we also measure the median of resolution in days. Corresponding medians of category 1 duplicates and category 2 duplicates are 0.7 days and 0.4 days respectively in Eclipse and 0.6 days and 0.4 days respectively in Mozilla. Although medians seem close to each other, both differences are found to be statistically significant ($p < .001$). More people are involved in masters of Missed Reopen (Category 2) duplicates when CC list sizes are taken into account. Thus, more people are aware of these

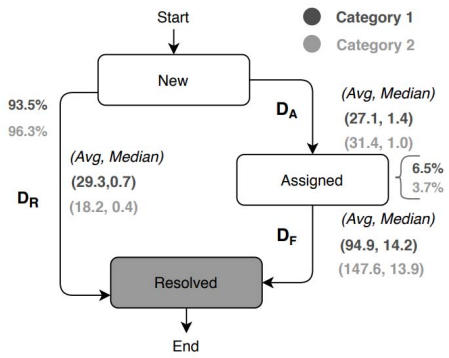



Fig. 6: Time distribution (in days) of duplicate bugs per each category in Eclipse.

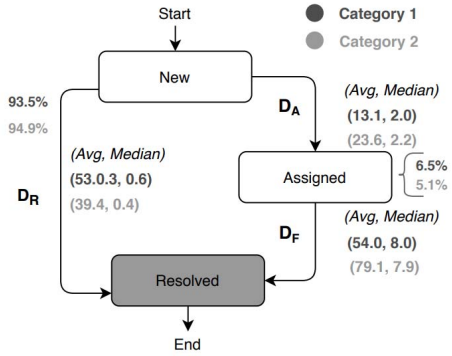



Fig. 7: Time distribution (in days) of duplicate bugs per each category in Mozilla.

bugs so they realize earlier that it is a duplicate.

In Eclipse, 6.5% of Master-Unresolved (Category 1) duplicates and 3.7% of Missed Reopen (Category 2) duplicates are assigned. In Mozilla, corresponding assigned percentages are 6.5% and 5.1% respectively. Among these assigned bugs, the difference of assignment duration between two categories is found to be statistically insignificant at the 5% level ($p$ = .304 in Eclipse, $p$ = .097 in Mozilla ). Likewise, considering fixing duration, the difference between the two categories is found to be statistically insignificant at the 5% level ($p$ = .237 in Eclipse, $p$ = .309 in Mozilla ).

## VI. Threats to Validity

In this section, we address the internal and external threats to the validity of our study.

**Internal Validity:** The duration of fixing ($D_F$) is measured from the timestamps in the bug tracking tool. It cannot be guaranteed that this is the actual duration of the developers working on the bug. Bug fixing might overlap with other activities of developers. However, we still believe that the effort developers spent on the bug can be represented with this metric. Actual duplicate bugs may be different from what was extracted because some duplicate bugs might not get marked as duplicate or non-duplicate bugs might mistakenly get marked as duplicate, by triagers. We assume their assignments are correct. We also discard duplicate bugs which do not have the typical workflow as in Figure 4 or if their master bugs do not have desired features such as not resolved yet which is key to categorize our duplicate bugs. Although this circumstance might lead to not analyzing all duplicate bugs in the datasets, we believe that discarding these bugs helps us to reach more genuine results with the remaining 34907 duplicate bugs in total.

To increase the replicability of the study, we shared the datasets that we used along with the implementation of the tools in Zenodo[11].

**External Validity:** We argue that Eclipse and Mozilla projects are sufficiently representative for our study since they are long-lasting projects. However, our findings cannot be generalized for other systems; and more empirical studies are needed to explore the subject further.

## VII. Conclusion and Future Work

Bug reporting is an essential part of the software development and maintenance process. However, it is still a challenge to report the bug only once even with the assistance of the bug tracking tools. These practically unavoidable duplicated bug reports cost extra maintenance effort in triaging and fixing bugs. To obtain a comprehensive understanding of duplicated bugs, we analyzed different properties of Category 1 (Master-Unresolved) and Category 2 (Missed Reopen) duplicated bugs. We also analyzed how duplicate and unique (i.e., non-duplicate) bugs differ. To the best of our knowledge, this is the first study to categorize duplicate bugs by their relative discovery time.

According to our empirical analysis on 165,547 bugs of the Eclipse project and 394,878 bugs of the Mozilla project, the summary of the results are as follows:

- Approximately 16% and 23% of bugs are duplicate in Eclipse and Mozilla respectively, indicating that the duplicated bugs play a significant role in software projects.
- When duplicated (i.e., master) bugs and unique bugs are compared with each other, it is seen that duplicated bugs are up to 2 times more severe than unique bugs. Additionally, duplicated bugs have up to 3.5 times more users in their CC list than unique bugs. Since duplicated bugs are more severe, it might be the reason that more people are involved. In this regard, the distinction between duplicated bugs and unique bugs is clear when the severity and CC list size are analyzed.
- The difference between the two duplicate categories in terms of severity is statistically insignificant. However, this difference between master bugs and unique bugs is significant because when master bugs are duplicated, they are re-evaluated carefully. On the other hand, the severity of a duplicate bug is not meaningful to the development team once they realize it is the duplicate of another one, so it is not re-evaluated.

[11]https://zenodo.org/record/4114096

- The median of Duplicate Bug Surface Time of Missed Reopen (Category 2) is approximately 4 times the median of the Master-Unresolved (Category 1) duplicates. This indicates that Category 2 bugs are more likely to be surfaced later compared to Category 1 bugs.
- Directly resolved (i.e., without any assignment) duplicates cost extra maintenance in bug triaging. Missed Reopen (Category 2) duplicates are resolved significantly faster than Master-Unresolved (Category 1) duplicates.

We believe that our findings will provide useful insights for both researchers and practitioners to better understand the typical features of duplicated bugs by comparing them with unique bugs. Furthermore, we empirically demonstrate that the two subcategories (Master-Unresolved and Missed Reopen Bugs) have distinguishable characteristics from many aspects.

We summarize the potential implications of our study as follows:

- Practitioners would have a better understanding of the different bug categories supported by empirical analysis.
- Practitioners can use the proposed categorization to potentially avoid (by introducing guidelines  rules to their bug management workflow) duplicated bugs.
- Researchers can use the enriched (Category 1 and Category 2 duplicates) data to improve their duplicate detection algorithms. The findings of this study provide potential insights into how a duplicate bug report detection algorithm should be designed. Algorithms might prioritize features that we found to be statistically significant.
- The current duplicate avoidance tools that search for possible duplicates such as Find Duplicates [25], might be improved by calibrating their duplicate detection by considering different categories.

As future work, we would like to analyze subcategories in depth by analyzing many more aspects such as why some duplicates are resolved earlier and the relationship between who resolves or introduces the duplicate bugs and characteristics of these bugs. In addition to mining Bugzilla projects, we are planning to mine projects that use Jira as well. We also plan to use more open source datasets and industrial data to increase the generalizability of our study.

## REFERENCES


[1] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 253–262.

[2] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 52–61.

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

[4] Y. C. Cavalcanti, P. A. d. M. S. Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, "The bug report duplication problem: an exploratory study," *Software Quality Journal*, vol. 21, no. 1, pp. 39–66, 2013.

[5] M. Li, L. Shi, and Q. Wang, "Are all duplicates value-neutral? an empirical analysis of duplicate issue reports," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 272–279.

[6] Q. Mi and J. Keung, "An empirical analysis of reopened bugs based on open source projects," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–10.

[7] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1074–1083.

[8] H. Rocha, G. de Oliveira, M. T. Valente, and H. Marques-Neto, "Characterizing bug workflows in mozilla firefox," in *Proceedings of the 30th Brazilian Symposium on Software Engineering*, 2016, pp. 43–52.

[9] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical software engineering*, vol. 19, no. 6, pp. 1665–1705, 2014.

[10] D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, "How do bugs surface? a comprehensive study on the characteristics of software bugs manifestation," *Journal of Systems and Software*, vol. 113, pp. 27–43, 2016.

[11] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli, "Software development: do good manners matter?" *PeerJ Computer Science*, vol. 2, p. e73, 2016.

[12] M. Khattar, Y. Lamba, and A. Sureka, "Sarathi: Characterization study on regression bugs and identification of regression bug inducing changes: A case-study on google chromium project," in *Proceedings of the 8th India Software Engineering Conference*, 2015, pp. 50–59.

[13] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 1005–1042, 2013.

[14] R. Souza, C. Chavez, and R. A. Bittencourt, "Do rapid releases affect bug reopening? a case study of firefox," in *2014 Brazilian Symposium on Software Engineering*. IEEE, 2014, pp. 31–40.

[15] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 499–510.

[16] A. Hindle and C. Onuczko, "Preventing duplicate bug reports by continuously querying bug reports," *Empirical Software Engineering*, vol. 24, no. 2, pp. 902–936, 2019.

[17] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2012, pp. 70–79.

[18] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 117–127.

[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 *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.

[20] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 45–54.

[21] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful... really?" in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 337–345.

[22] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[23] A. Lamkanfi, J. Perez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information," in *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories, May 18–19, 2013. San Francisco, California, USA*, 2013.

[24] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[25] Reliex. (2014, Oct) Find duplicates app for jira. [Online]. Available: https://marketplace.atlassian.com/apps/1212706/find-duplicates