

Characterizing duplicate bugs: Perceptions of practitioners and an empirical analysis

Berfin Kucuk¹ | Idil Hanhan²  | Eray Tuzun² 

¹Udemy, Istanbul, Turkey

²Department of Computer Engineering,
Bilkent University, Ankara, Turkey

Correspondence

Idil Hanhan and Eray Tuzun, Department of
Computer Engineering, Bilkent University,
Ankara, Turkey.
Email: idilhanhan@gmail.com and 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 for a previously reported bug, which results in duplicate bug reports. Users might report duplicate bugs if they miss the previous bug report or if the previous bug is unresolved (i.e., in progress) and therefore cannot be reopened. These duplicate bug reports may cost extra maintenance efforts in triaging and bug fixing. 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 interaction, which is *missed reopen* bugs. Our study is based on analyzing the difference between duplicate and nonduplicate bugs and further categorizing the duplicate bugs. In this regard, we categorize duplicate bugs according to the original bug's resolution status at the duplicate's creation time as Master-Unresolved bugs and Master-Resolved (Missed Reopen) bugs to distinguish their properties. We compare these two types of bugs in terms of their relationship to their master bug, 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. Additionally, we investigate the perceived importance, impact, and causes of duplicate bugs, as well as the difference between nonduplicate and duplicate bugs and its categories for practitioners in the software industry by conducting a survey.

KEYWORDS

bug management, bug reopen, characterization study, duplicate bug reports, reopened bugs

1 | INTRODUCTION

Software bugs are inevitable due to the complexity of software systems. With a bug-tracking system like Bugzilla,* 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 description given by the bug reporter or an incorrect fix. In other cases, an uncoordinated process of bug reporting might cause the same bug to be reported more than once. The triager will inspect if a bug report is a duplicate; if so, the triager will

*<https://www.bugzilla.org/>.

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}

There have been several studies that investigate duplicate bugs^{3–5} 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 interaction. In this study, we categorize duplicate bugs to show that not all duplicates are the same and that they have distinguishable characteristics. We believe that such distinction will increase the understanding of duplicate bugs. 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 bug are referred to as Master-Unresolved and those that have been submitted after the resolution of their master bug are referred to as Master-Resolved (Missed Reopen bugs). The expected behavior for handling Master-Resolved duplicates is reopening the corresponding master bug report instead of opening a new one. On the other hand, the expected behavior for Master-Unresolved duplicates is not being opened at all because the master bug is still active. Throughout this study, we will refer to the Master-Unresolved bugs as Category 1 and 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 use two more measures; severity that indicates the degree of impact that a bug has on the system and the number of users involved that indicates the perceived importance of a bug.

Previously, we reported⁸ our quantitative analysis on nonduplicate (unique) bugs and duplicate bugs (Categories 1 and 2) in light of the following research questions (RQs):

RQ1: How do duplicate and nonduplicate bugs differ?

RQ1.1 How do duplicate and nonduplicate bugs differ in terms of *the bugs' severity*?

RQ1.2 How do duplicate and nonduplicate bugs differ in terms of *the number of users involved*?

RQ2: How do Categories 1 and 2 bugs differ?

RQ2.1 How do Categories 1 and 2 differ in terms of *the bugs' severity*?

RQ2.2 How do Categories 1 and 2 differ in terms of *the number of users involved*?

RQ2.3 How do Categories 1 and 2 differ in terms of *DBST*?

RQ2.4 How do Categories 1 and 2 differ in terms of *their time distribution over their life cycle*?

We investigated these RQs by analyzing the bug report history of two open-source projects, Eclipse and Mozilla. We investigated the severity of different bug types according to the severity type given in the bug reports and define a severe bug as a bug that prevents the system from working. We found that duplicate bugs are up to two times more severe than unique bugs and that the severity between Categories 1 and 2 bugs is statistically insignificant. In order to investigate the number of users involved in different bug types, we used the Carbon Copy (CC) list size of the bug report as an indicator for the number of users involved in the bug. We found that duplicate bugs have up to 3.5 times more users than unique bugs and that more users are involved in the master's of Category 2 bugs than master's of Category 1 bugs. We investigated the duplicate bug surface time of Categories 1 and 2 bugs by measuring the time elapsed between the report of the master and its duplicate. We found that Category 1 bugs are more likely to be surfaced later than Category 2 bugs. In order to investigate the time distribution of Categories 1 and 2 bugs, we defined time points in the lifetime of the bug and measured their duration of assignment, fixing and resolution. We found that Category 2 duplicates are resolved faster than Category 1 bugs.

In this study, we extend our previous work⁸ by investigating the perceived importance, impact, and cause of duplicate bugs, as well as the perceived difference between different types of bugs. We analyzed the responses to our survey that was conducted with 51 participants from the software industry, in the light of the following RQs:

RQ3: What is the perceived impact of duplicate bugs in the software industry?

RQ4: What are the potential reasons for duplicate bug occurrence in the software industry?

RQ5: What is the perceived difference between duplicate and nonduplicate bugs in the software industry?

RQ5.1 How does the perceived difference between duplicate and nonduplicate bugs differ in terms of *the bugs' severity*?

RQ5.2 How does the perceived difference between duplicate and nonduplicate bugs differ in terms of *the number of users involved*?

RQ6: What is the perceived difference between Categories 1 and 2 bugs in the software industry?

We investigate these RQs by analyzing the responses to our survey. The survey consists of multiple-choice, multiple-selection, and open-ended questions. The analysis of the responses to the first two question types consists of directly sharing the participant's responses and their

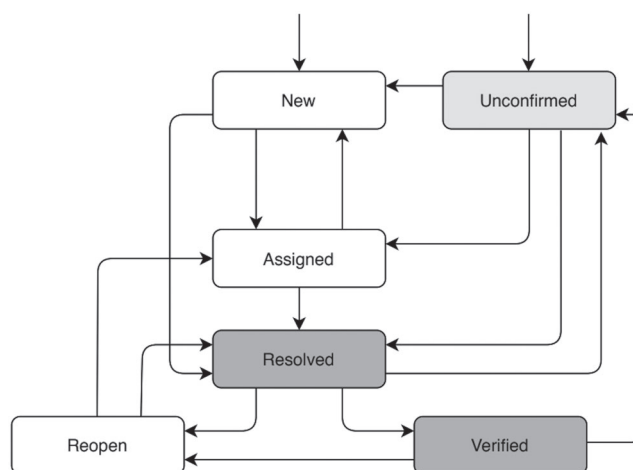


FIGURE 1 Bug life cycle of Bugzilla

percentage among all responses. For the responses to the open-ended questions, we categorize each response to a response category that we identify during our analysis. For the perceived impact, we found that 47.1% of the participants rate the impact of duplicate bugs as “Moderate” and the majority (75%) mention “Waste of Resources” as their impact. For the potential reasons of duplication, we found that 88.2% of the participants believe that Customers/Users are more likely to report duplicates and 32.6% argue the overall reason of duplication to be “Lack of Research.” For the perceived difference between duplicate and nonduplicate bugs, we found that more participants believe that duplicates have more users involved and 63.3% disagree that duplicates with high severity are more likely to be duplicated. Finally, our investigation of the perceived difference between Categories 1 and 2 bugs show that Category 1 bugs are perceived to be more important and the most common reason (43.2%) for this selection is the rework these bugs can cause. Additionally, we also found that majority of the participants (58.1%) believe that Category 2 bugs are closed earlier because they are “Easier to Detect.”

The rest of the paper is organized as follows. Section 2 presents background information on the typical life cycle of a bug, the characterization of bugs in general, reopened bug reports, and duplicate bug reports. Section 3 presents our methodology for analyzing our RQs. Section 4 describes the evaluation setup, and Section 5 describes the survey setup and shares the demographic characteristics of the participants. Section 6 presents our results on the Eclipse and Mozilla datasets and our analysis of the survey responses. Section 7 presents the insights of the results and compares the results of the previous work and survey results when applicable. Section 8 presents threats to validity, and finally, Section 9 concludes the paper.

2 | BACKGROUND AND RELATED WORK

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

2.1 | Bug life cycle

A bug life cycle describes the workflow of a bug from its creation to its resolution.⁹ Figure 1 shows the default workflow of Bugzilla[†] which is used as a bug-tracking system by both Eclipse and Mozilla. The vertices represent the states, and the edges represent the transition between states. 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 product is registered as *New* while others are registered as *Unconfirmed*.

The bug status changes from *Unconfirmed* to *New* whenever the bug is confirmed. 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, 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

[†]<https://www.bugzilla.org/docs/3.6/en/html/lifecycle.html>.

directly if the bug is not valid or 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*.

If the QA team is not satisfied with the solution or the bug reoccurs, its status changes to *Reopen*. A reopened bug can return to *Assigned* and *Resolved* states. A bug in the *Resolved* or *Verified* states can return to an *Unconfirmed* state if it is reopened and is never confirmed. A bug is closed when it is verified or resolved.

2.2 | Characterization of bugs

There are studies that characterize bugs from several perspectives. Tan et al¹⁰ manually investigate bugs in three dimensions: root causes, impacts, and components. Their findings show that as software evolves, semantic bugs increase, while memory-related bugs decrease, calling for more effort to address semantic bugs. Cotroneo et al¹¹ investigate the characteristics of bugs from the bug manifestation perspective. They report the most relevant trigger types, their occurrence time, and their relation to complexity and fixing time. Results of Destefanis et al¹² show that the more polite developers in the communication process are, the less time it takes to fix an issue. Khattar et al¹³ conduct a characterization study on regression bugs, that is, 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.

2.3 | Reopened bug reports

The work by Zimmermann et al⁷ 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. They conclude that the source of a bug influences the likelihood of bug reopens such that bugs found by code analysis or human reviews are less likely to be reopened whereas bugs found by customers and during system testing are more likely to be reopened. Shibab et al¹⁴ 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 al¹⁵ show that the bug reopening rate of versions developed in rapid cycles was about 7% higher. Mi and Keung⁶ 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 the normal ones. Caglayan et al¹⁶ build a univariate and best subset logistic regression model to find the most important factors in issue reopening. They find that issue complexity and developers' workload play an important role in triggering issue reopening. Xia et al¹⁷ evaluate the effectiveness of various supervised learning algorithms to predict if a bug report would be reopened. They claim that among the 10 algorithms, Decision Table and Bagging achieve the best performance with accuracy scores of 92.80% and 92.91%, respectively.

2.4 | Duplicate bug reports

There have been few prior works on analyzing characteristics of duplicate bugs. Cavalcanti et al⁴ 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 are influential. Li et al⁵ 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 al³ investigate duplicate bug reports in free/open software projects. Most open-source projects allow users to participate by reporting bugs which leads 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 whether they are different in terms of their severity, the number of users involved, surface time, and their time distributions. 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.¹⁸ Their evaluation shows that about 2/3 of the duplicates can be detected using the NLP techniques. Sun et al¹ 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, and version. Hindle and Onuczko¹⁹ 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 duplicates before they occur in 42% of cases by creating a simple IR model. The work by Nguyen et al²⁰ introduces a duplicate bug report detection approach that takes advantage of both an IR approach

and topic-based features. He et al²¹ 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 association features to classify whether a pair of bug reports are duplicate or not. The work by Wang et al²² present 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.²³ As stated in Bettenburg et al.'s work,²⁴ additional information provided by duplicates helps to resolve bugs quicker. Thus, many duplicate bug report detection algorithms add links to duplicates.²³

In our study, we do not propose any techniques 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.

3 | METHODOLOGY

In this section, we describe our methodology to analyze duplicate bugs. In Section 3.1, we categorize duplicate bugs according to the resolution time of their master bugs. Then, we present the details of our methodology to analyze our RQs in separate sections.

3.1 | Categorizing duplicate bugs

In our study, duplicate bugs are grouped into two categories. Figure 2 shows these categories of duplicate bug reports in relation to the timeline of their master bugs. The top line indicates the timeline of a master bug where we only take into account the first resolution cycle of the master bug. The time stamps on the top line: M_{NEW} , $M_{ASSIGNED}$, $M_{RESOLVED}$, and $M_{VERIFIED}$ indicate, from left to right, the time stamps of the possible states in a lifecycle of a master bug. For simplicity, bug states other than New, Assigned, Resolved, and Verified are not shown in the figure. Each line below the master bug line indicates a possible creation of a duplicate bug relative to the state of its master as D_{NEW} . In this regard, we categorize duplicate bugs as follows:

Category 1 (Master-Unresolved): Duplicate is submitted prior to master's resolution. As seen in Figure 2, the first two lines below the master bug line represent 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 bug is assigned which does not affect the category they belong to. Because Category 1 duplicates are submitted when their master is not resolved yet, we name them Master-Unresolved duplicates.

Category 2 (Master-Resolved or Missed Reopen): Duplicate is submitted after the master's resolution. As seen in Figure 2, the last two lines in the figure represent Category 2 duplicates, in which a duplicate bug is created as New after its master bug has been resolved. These duplicate bugs might get created before or after the master bug has been verified which does not affect the category. Because Category 2 duplicates are submitted when their master is resolved, we name them Master-Resolved duplicates. The expected behavior for resolved bugs, in case of a recurrence, is for them to be reopened instead of creating a new bug report which is a duplicate; therefore, we also refer to these duplicates as Missed Reopen duplicates.

To summarize, the duplicate bugs which are created *before* the resolution of their master bug belong to Master-Unresolved Category 1 duplicates while others, which are created *after* the resolution of their master bug, belong to Missed Reopen Category 2 duplicates. Category 1 consists of duplicate bug reports that are reported within the same time interval by different reporters. These bugs are reported such that the initial bug report is not resolved yet. Therefore, users who report the same bug afterwards cannot reopen the bug, because the master bug is not closed yet.

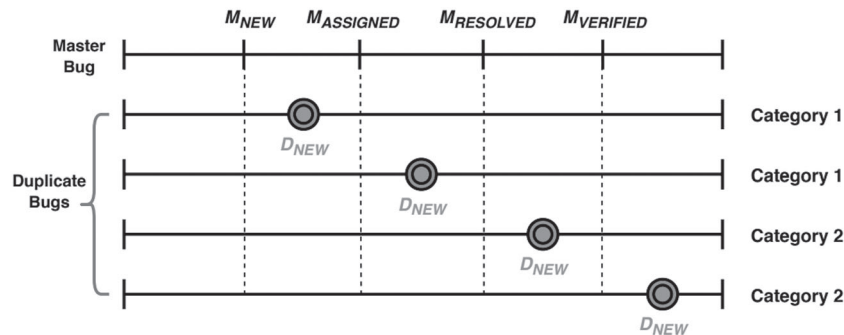


FIGURE 2 Timeline of duplicate bug categories regarding their master bugs

This circumstance results in duplicate bug reports because the reporters may not scan previously reported bugs, which are generally checked by the triagers. In contrast to the Master-Unresolved (Category 1) duplicates, in the Missed Reopen (Category 2) duplicates, reporters are able to reopen the bug reports of the same bug. However, reporters create a new bug report instead, which is a duplicate bug report. These duplicates may cost extra maintenance efforts in triaging and fixing bugs.

3.2 | 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[‡] describes the severity types, from highest to lowest, as follows:

- *Blocker*: The bug that blocks development and/or testing.
- *Critical*: Crashes, severe memory leak, loss of data might occur 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 and is caused by 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 master bugs to all masters 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.

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.

3.3 | Number of users involved in different bug types

In Bugzilla, as in most of the bug-tracking tools, a CC list is used to add people to a mailing list that notify them when the bug has an update.[§] The reporter can add users to the CC list, and other users can also add themselves to the CC list if they wish to be notified when there is a change in the bug status. It is also 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 and so on.

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 for each bug type to understand if there is a relationship between the number of users involved and a bug being duplicated. Additionally, we also compare the average number of CC list users in Master-Unresolved duplicate bugs (Category 1) and Missed Reopen duplicate bugs (Category 2).

3.4 | DBST

DBST represents the time elapsed between the report of the master and the report of its duplicate. Mathematically, it is depicted by Equation (1) with the notations from Figure 2.

$$DBST = D_{NEW} - M_{NEW}. \quad (1)$$

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 DBST of each category.

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

[§]<https://www.bugzilla.org/docs/3.0/html/userpreferences.html>.

3.5 | The time distribution of duplicate bugs

Duplicate bugs put extra overhead on software projects because the effort of managing duplicate bugs can be time consuming. In order to investigate this, we analyze where most of the time is spent and refer to this as the time distribution of the bug.

A duplicate bug might follow different paths in the bug life cycle before it is eventually marked as a duplicate. In one possible path, the bug is resolved as duplicate right after it is created as *New*. Another path might be the one in which the bug is resolved as duplicate after it passes *New* and *Assigned* states.

In this regard, we first define three different time points as shown in Table 1, and then, we split the life cycle of a duplicate bug into intervals demonstrated in Figure 3A,B. The time points in Table 1 are defined similar to the work of Mi and Keung,⁶ where they are used to define time points in the timeline of reopened bugs. These time points are T_{new} , T_{assign} , and $T_{resolve}$.

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 define the time point of the last assignment as T_{assign} . In another case, as seen in Figure 3B, a duplicate bug might get resolved as a duplicate without any assignment. In this case, we assume that the total time spent on the bug is for resolution.

TABLE 1 Definitions of time points in the timeline

Time point	Description
T_{new}	The time when a bug is confirmed and starts its life cycle
T_{assign}	The time when a bug is assigned to a developer
$T_{resolve}$	The time when a bug is resolved with resolution status duplicate

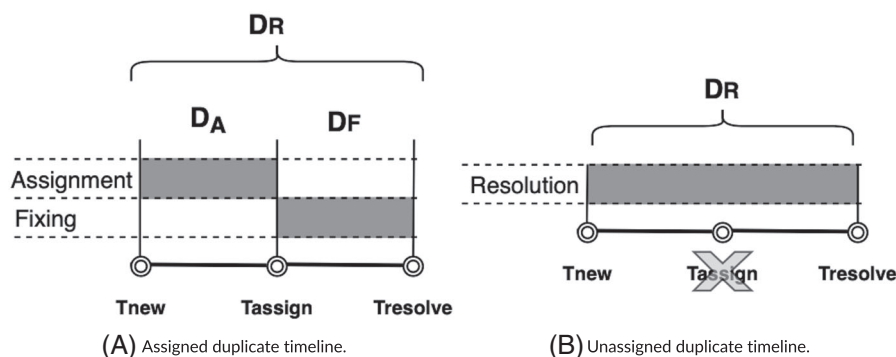


FIGURE 3 Timelines of duplicate bugs

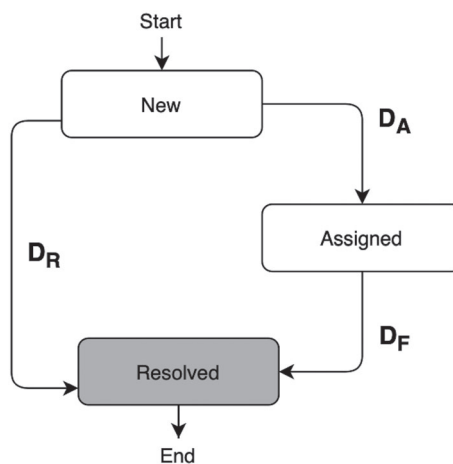


FIGURE 4 Simplified typical workflow of a duplicate bug

With the timelines of duplicate bugs given in Figure 3A and time point definitions given in Table 1, we use the following metrics, from the work of Mi and Keung,⁶ 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 ($T_{resolve}$).

Duration of resolution as duplicate in days (D_R): The interval between a bug being opened (T_{new}) and it being resolved as duplicate ($T_{resolve}$).

We are limiting our analysis to duplicate bugs that have the workflow in Figure 4 because duplicate bugs that pass through other states would prevent us from observing the genuine impact of duplication. For example, a bug that has been resolved as “later” and then is reopened and resolved as a duplicate does not represent a typical life cycle of a duplicate bug.

Time spent on fixing duplicate bugs is a duplication of effort because these bugs might get resolved as duplicate after developers start working on or fix these bugs. We use D_F to measure the negative impact of duplicate bugs on developers. Although D_F does not represent the actual time that a developer spends on fixing a bug, this attribute is still an important indicator in terms of quantifying the extra workload for developers.

We measure the aforementioned metrics for each category to compare them and investigate the differences among these categories. We use each bug's status change history to figure out $DBST$, D_A , D_F , D_R , severity, and number of involved users along with separate measurements for each category, because this information is kept by the bug-tracking tools.

In this regard, among two duplicate bug categories, we compare severity, number of users involved, $DBST$, and their time distributions to see whether they are statistically different. Additionally, master bugs and unique bugs are compared statistically in terms of the severity and the number of users involved.

4 | EVALUATION SETUP

In this section, we first describe the datasets that we have analyzed and then explain the data preprocessing steps.

4.1 | About dataset

We investigate our approach on two different open-source projects: Eclipse and Mozilla. Eclipse[¶] is an open-source project and an extensible multilanguage software development environment. Mozilla Firefox,[#] 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^{||} as their bug-tracking software. Both Eclipse and Mozilla are widely used and have cross-platform (i.e., Mac OS X, Linux, Windows, and Solaris) support. Thus, both Eclipse and Mozilla's bug report history is sufficient for an in-depth analysis.

In this study, we used Lamkanfi et al.'s²⁵ bug-tracking dataset which is filtered to contain only genuine bugs (i.e., no feature requests and only resolved bugs) within the whole bug-triage life cycle. Eclipse Bugzilla^{**} dataset includes 165,547 bug reports and Mozilla Bugzilla[†] dataset includes 394,878 bug reports. However, these datasets do not keep the relationship between duplicate bugs and their master bugs. Therefore, we extract the master bug of each duplicate bug using HTML parsing from their corresponding website. Eclipse and Mozilla datasets include 26,816 (16.2%) and 91,133 (23.1%) duplicates, respectively. The resulting datasets, which include duplicate information, along with the implementation of the tools in this study are available online.^{‡‡}

4.2 | Data preprocessing

Prior to the quantitative analysis on RQs, 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 were 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. Because 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 will reach more genuine results by removing them. In this context, the number of discarded duplicates is 52 (0.2%) and 56,304 (62.0%) in Eclipse and Mozilla, respectively. Most of the

[¶]<https://www.eclipse.org>.

[#]<https://www.mozilla.org/>.

^{||}<https://www.bugzilla.org>.

^{**}<https://bugs.eclipse.org/bugs/>.

[†]<https://bugzilla.mozilla.org>.

^{‡‡}<https://zenodo.org/record/4114096>.

duplicates of Mozilla were discarded in this process. This might be due to Mozilla having a better duplicate bug detection in the early stages or due to their users' inattention to previously reported bugs which could result in a significant number of duplicate bug reports.

Additionally, in our study, we consider duplicate bugs that experience the typical workflow given 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 the *New* state. In Eclipse and Mozilla, discarded bugs are 3226 (12.03%) and 2050 (2.3%), respectively. Then, we discarded 505 (1.9%) duplicate bugs in Eclipse and 3909 (4.3%) duplicate bugs in Mozilla whose master bugs were not in the dataset. With respect to master bugs, we discarded 1629 (6.1%) duplicate bugs in Eclipse and 3618 (4.0%) duplicate bugs in Mozilla whose master bugs were not confirmed nor resolved because the master bug's assignment and resolution is key to our categorization of duplicates. We also discarded duplicate bugs that have been reported before their master bugs have been reported which is in conflict with our master bug definition; as a result, 4671 (17.4%) bugs in Eclipse and 7078 (7.8%) bugs in Mozilla were discarded. In the end of preprocessing, we had 16,733 (62.3% of all duplicates) duplicate bugs in Eclipse and 18,174 (20.0% of all duplicates) duplicate bugs in Mozilla.

5 | SURVEY SETUP

In this section, we explain the purpose of our survey and the survey process and present the demographics of the participants. In order to further investigate the characteristics of duplicate bugs, we conducted an online survey from May to June 2021 with 51 participants. The survey includes questions related to the participant's experience with duplicate bugs, the possible reporter, and cause of such bugs and their impact. Our intent was to get in touch with people who work in software industry and are actively using a bug-tracking system. Therefore, we decided to contact our network, via email and phone, to ask them to participate, and they were also free to distribute the survey to their network. This type of sample is referred to as nonprobability or convenience sample.²⁶ We received 51 responses with 68% response rate, not including the cases where the survey was forwarded outside of our knowledge. Participants are sufficiently experienced in the software industry, with an average of 12.5 years of experience. Only 3.9% of the participants do not use a bug-tracking system, while 92.2% use bug-tracking systems for reporting bugs, 86.3% use them to close bugs, and 86.3% use them to browse bugs. It should be noted that the participants can choose multiple use cases for the way they use the bug-tracking system.

We allow participants to declare their job titles as they wish, and as a result, we have 40 different job titles. Some are the same title written in a different way, such as *Software Engineer* and *Software Engineer (backend)*. Others are different titles in text, but due to the responsibility, they indicate we group them under more general titles, such as *technical lead* and *lead project engineer*. This results in six job categories that are software engineer, software architect, QA engineer, technical management, upper management, and consultant. The majority of the participants are software engineers (52.9%) followed by technical management (17.6%), upper management (9.8%), QA engineers (7.8%), software architects (5.8%), and consultants (5.8%). The participants are from at least 27 different companies and one undefined company; 21.4% of our participants did not specify their company but instead provided a city name. We consider these unspecified companies as one and group them under an undefined company. We have a single participant from 71.4% of the companies, two from 14.3% of the companies. We have four, five, six, and eight participants from 3.6% of the companies.

Prior to sending the questionnaire to the participants, we conducted a pilot test questionnaire to test and improve the instrument. The pilot included six members from the target audience, and as a result, we decided to further clarify and give more information about some questions.

The final questionnaire consists of four parts. The first part is for understanding the demographic characteristic of our participants. We ask them their job title and experience in the software industry and in their current company. Additionally, we ask if they use a bug-tracking system and if so, how do they use it. The second part includes questions related to duplicate bugs. We give a description of duplicate bugs and ask, both in the form of multiple choice and open-ended questions, the implications of having them. We also ask what could be the possible reason of duplicate bugs. The aim of the third part is to understand the participant's, and their companies, bug-reporting process for both unique and duplicate bugs. The fourth and final part is related to categories of duplicate bugs. We first describe unique and duplicate bugs and ask about their difference in terms of severity and number of users involved. Then, we describe Master-Unresolved and Missed Reopen bugs and ask which category they perceive to be more important and why. The survey questions can be seen in AppendixS1, and the survey is also shared.^{§§}

In summary, with this survey, we aim to understand the following:

- How duplicate bugs are treated in the software industry
- Why duplicate bugs are reported and by whom
- Whether practitioners differentiate between unique and duplicate bugs
- Whether practitioners differentiate between Categories 1 and 2 duplicate bugs.

^{§§}<https://forms.gle/ef4tmBzFJtFar7R9>.

Similar to the work of Zimmerman et al,^{6,7} for the open-ended questions, two authors analyzed the responses independently, performed an open card sort, and then merged their results into a single taxonomy. Later, the third author reviewed these categories, and authors discussed the categories or responses that caused concern. In case of a disagreement, where some argued the response should be in Category A and another suggested Category B, we voted and settled on the decision of the majority.

Our initial aim, with each question, was to group one response into one category, meaning a response could only be a part of one of the categories we defined. However, as we continued our analysis, we realized that a significant number of responses actually contained multiple responses. In order to ensure we did not overlook the insights given by our survey participants, we decided to allow answers to be a part of multiple categories. We also had some sections where users were able to write in short responses. For example, when we asked participants to explain what they do before reporting a bug, we also allowed them to write their own explanation. As a result, we had multiple explanations for more or less the same behavior. In order to have a clear representation of the responses, we combined and clarified such answers.

6 | RESULTS

In this section, we present the results of the proposed RQs. First, we present the results of RQ1 and RQ2 which are based on the analysis of Eclipse and Mozilla's bug reports. In Eclipse, 57% of the duplicate bugs are Master-Unresolved (Category 1) and 43% of them are Missed Reopen (Category 2) duplicates. In Mozilla, 79% of them are Master-Unresolved duplicates and 21% of them are Missed Reopen duplicates. Then, we present results of RQ3, RQ4, RQ5, and RQ6 which are based on the analysis of the responses to our survey.

6.1 | Statistical testing

Prior to sharing the results, we first want to explain the statistical testing we use to investigate whether the difference between our findings for different bug categories are statistically significant. For statistical testing of the severity, whose results will be shared in RQ1.2 and R2.1, we scale the severity from 6 to 0 from highest to lowest. These ordinal data are used in the Mann–Whitney nonparametric test.²⁷ We use the same test for the statistical testing of the difference for number of users in different bug types. For the statistical testing of the time distribution, we again conduct the Mann–Whitney nonparametric test (using 5% level) to examine whether the average of each metric of specified two categories are statistically different.²⁷ We adopt Mann–Whitney test for significance test because our data are not normally distributed. They are measured on an ordinal or continuous scale, and all the observations from both groups are independent of each other which also make our data suitable for Mann–Whitney test.

6.2 | RQ1.1: Severity of duplicate and nonduplicate bugs

We believe that bugs with severity higher than normal prevent the system from working properly. Therefore, we calculate and compare the ratio of these severe bugs for different bug types. We present the severity of different bug types in Table 2. The table shows percentages of severe bugs in different bug types. The ratio 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 ($p < 0.001$) for both datasets. In this regard, the distinction between unique and duplicate bugs is clear when the severity is taken into account, and therefore, severity can be one of the metrics to decide if the bug will be duplicated.

TABLE 2 High severity percentages of different bug types

Bug type	Blocker		Critical		Major		Total severe bugs	
	Eclipse	Mozilla	Eclipse	Mozilla	Eclipse	Mozilla	Eclipse	Mozilla
Category 1 duplicates	1.9%	1.6%	3.9%	13.4%	9.9%	9.8%	15.7%	24.8%
Category 2 duplicates	2.7%	1.7%	5.7%	15.0%	10.3%	10.1%	18.7%	26.7%
Master bugs	1.7%	3.1%	5.4%	16.4%	13.4%	12.1%	20.5%	31.6%
Unique bugs	0.9%	0.8%	2.5%	9.5%	7.8%	9.6%	11.2%	19.9%

6.3 | RQ1.2: The number of users involved using the CC list of duplicate and nonduplicate bugs

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 for each bug type as notched box plots. 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.²⁸ 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 3. Medians of CC list size are low in duplicates compared with the master bugs as expected because duplicates are resolved relatively faster. Theoretically, master bugs represent actual bugs because they are not closed as duplicates and continue their life cycles.

Also, we see a drastic difference between duplicate bugs (i.e., masters) and nonduplicate (i.e., unique) bugs. In Eclipse and Mozilla, the medians of CC list size in unique bugs have one and three users in their CC list, respectively. In other words, the number of users in unique bugs are significantly less than master bugs ($p < 0.001$) for both datasets. Thus, we can conclude that bugs that have duplicates tend to have more users in their CC list.

6.4 | RQ2.1: Severity of Categories 1 and 2 bugs

As mentioned before, we present the severity of different bugs in Table 2. For example, in Eclipse, the percentage of high severity is 15.6 in Master-Unresolved (Category 1) duplicates while this number is 18.7 in Missed Reopen (Category 2) duplicates. However, both in Eclipse and Mozilla, the difference of severity between two duplicate categories is found to be statistically insignificant at the 5% level ($p = 0.161$ in Eclipse, $p = 0.107$ 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.

6.5 | RQ2.2: The number of users involved using the CC list of Categories 1 and 2 bugs

As explained under the results of RQ1.2, we use the CC list size for each bug type to measure the number of users involved. The CC list size as notched box plots can be seen in Figure 5, and again, we will be using the median to show the central tendency as in Table 3. In this regard, we

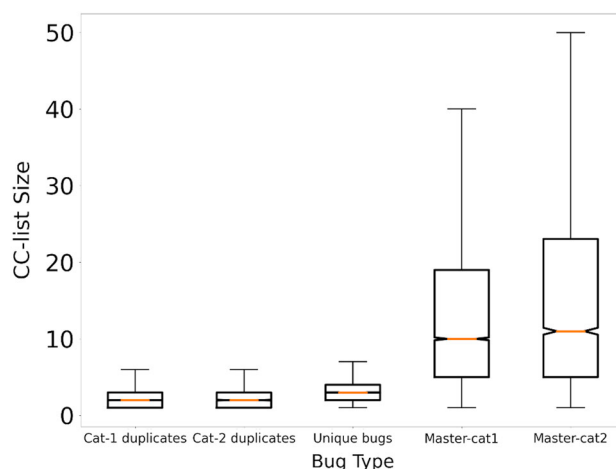


FIGURE 5 Distribution of CC list size for each bug type in Mozilla

TABLE 3 Median of CC list size in each bug type

Bug type	Median of CC list size	
	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

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 ($p < 0.001$) for both datasets. However, the median of the number of users for Categories 1 and 2 duplicate bugs are the same.

6.6 | RQ2.3: DBST

We also investigate 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. Considering DBST, the median of Missed Reopen (Category 2) duplicates are significantly higher than the median of Master-Unresolved duplicates, as seen in Table 4. For both datasets, this difference is found to be statistically significant at the 5% level ($p < 0.001$). Median DBSTs have approximately 1 to 4 ratio in Eclipse and 1 to 3 ratio in Mozilla which corresponds to Master-Unresolved and Missed Reopen duplicates, respectively. Because 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.

6.7 | RQ2.4: 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 the results of Eclipse in Figure 6 and Mozilla in Figure 7. 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, because 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 and 0.4 days, respectively, in Eclipse and 0.6 and 0.4 days, respectively, in Mozilla. Although medians seem close to each other, both differences are found to be statistically significant ($p < 0.001$). 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 bugs, so they realize earlier that it is a duplicate.

TABLE 4 Median of DBST (in days) in each duplicate category

Bug type	DBST median	
	Eclipse	Mozilla
Category 1 duplicates	24.00	33.35
Category 2 duplicates	90.52	102.98

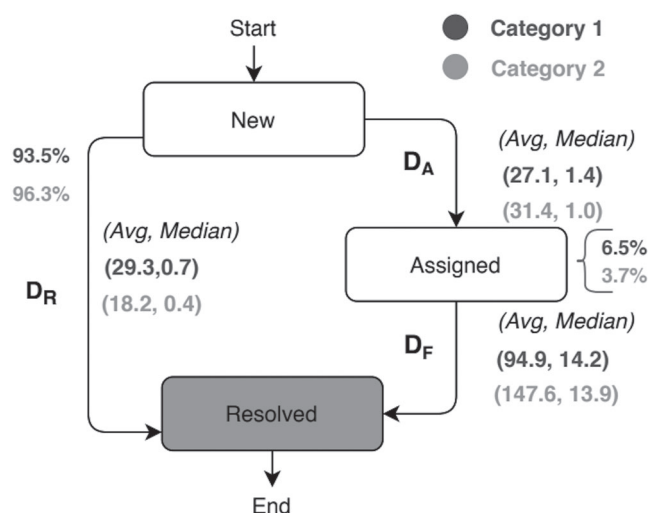


FIGURE 6 Time distribution (in days) of duplicate bugs per each category in Eclipse

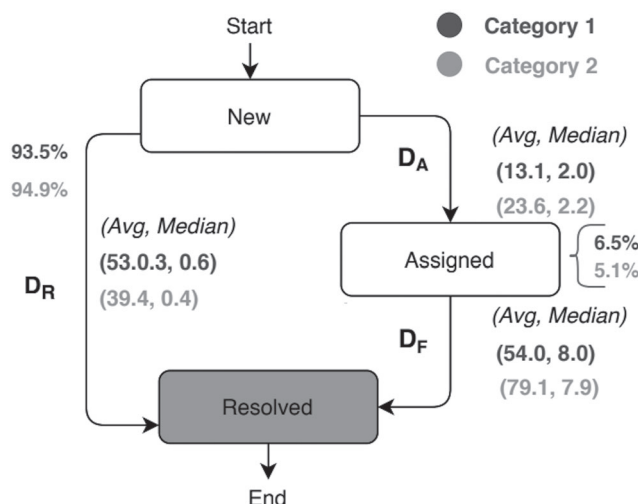


FIGURE 7 Time distribution (in days) of duplicate bugs per each category in Mozilla

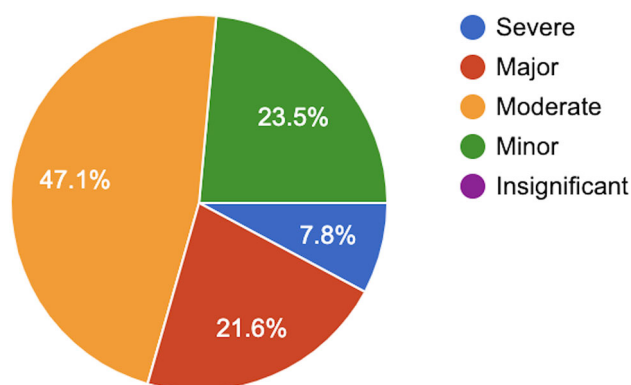


FIGURE 8 Responses to the question: How would you rate the negative effects of duplicate bugs?

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 ($p = 0.304$ in Eclipse, $p = 0.097$ in Mozilla). Likewise, considering fixing duration, the difference between the two categories is found to be statistically insignificant at the 5% level ($p = 0.237$ in Eclipse, $p = 0.309$ in Mozilla).

6.8 | RQ3: Perceived impact of duplicate bugs

To investigate the perceived impact of duplicate bugs in the software industry, we analyze the responses to our survey. We mainly focus on the respondent's rating of the impact of duplicate bugs and the detailed responses they gave related to this. We gather the respondent's rating with a multiple-choice question where they choose from Insignificant, Minor, Moderate, Major, and Severe. The results of the rating can be seen in Figure 8. For the detailed explanation of this impact, we analyze the responses to the following question: "What is the impact of duplicate bugs? (Why is it important to detect them?)." The analysis of the responses to this open-ended questions are first done individually by two of the author, then merged to form a single taxonomy and later reviewed by all authors. For the responses to this question, we define the following categories: "No Negative Impact," "Waste of Resources," "Noise in Project and Bug Metrics," "Signal for Concern," and "Loss of End User Trust." In order to ensure all of the insight given in a response is considered, we allow one response to be in multiple categories. An example response is: "Project metrics are affected. More resources are consumed to analyze them. Different resources may be assigned to solve the same problem." This participant first mentions the potential noise duplicates create in project metrics and later also states that duplicates waste resources. Therefore, this response is in both "Waste of Resources" and "Noise in Project and Bug Metrics" response category. The results can be seen in Table 5. Additionally, 5.9% (3 out of 51) of the responses are discarded because the response either had no or unclear or unrelated explanation.

TABLE 5 Response categories for question: What is the impact of duplicate bugs? (Why is it important to detect them?)

Category	Percentage	Example
Waste of Resources	75.0%	"Duplicates effort in tracking them, bug solving process run redundantly for each duplicate bug."
Noise in Project and Bug Metrics	25.0%	"It is important to mitigate duplicate bugs for better sprint/task planning."
No Negative Impact	12.5%	"It's not important actually. I combine it with the issue, which is a copy. This bugs sometimes has different detailed information. Actually it's good to read for every bug."
Signal for Concern	4.2%	"It shows that part is not written well according to the software principles, like SOLID etc."
Loss of End User Trust	4.2%	"losing customer/end user trust; reallocating time resource to fix the bug again (waste of resources)."

Note: Forty-eight out of 51 responses are considered, and responses can be in multiple categories.

As seen in Figure 8, most of the participants (47.1%) rate the impact of duplicate bugs as "Moderate"; 23.5% rate it as "Minor," 21.6% rate it as "Major," and only 7.8% rate it as "Severe." It is important to note that, none of the participants consider the impact of duplicates as "Insignificant." When it comes to the explanation of the impact, 75.0% of the responses mention waste of resources, namely, waste of time and effort. One participant lists the processes that are repeated for duplicates and states that "They are waste of time: they need be read, triaged, and reproduce again, mark as duplicate, write an explanation, they all take time." Another draws attention to duplication of effort and states that "It is inefficient because more than one team member works on the same bug unnecessarily." There is no pattern in the participant's impact rating when the impact that is explained is in the "Waste of Resources" response category, as this category includes response with all but "Insignificant" rating.

The second most common impact, with 25.0%, is the possible noise in project and bug metrics, meaning incorrect or misleading information, in project and bug metrics. These responses are in the "Noise in Project and Bug Metrics" category. One participant states that "Also, if the duplicate bugs are too many, it can misdirect managers when they look at the statistics." Here, the participant focuses on the duplicate's possible effect on the metrics which can cause the managers to rely on misleading information. Another participant presents the number of bugs in a project as a criteria in "project closure" and argues that the increase in number of bugs, due to duplicates, can risk the closure or acceptance of the project.

A total of 12.5% of the responses state that duplicates do not have any important or negative impact. In fact, one participant considers them as a positive because they could potentially have "detailed info" about the bug. Some participants credit the environment they work in for the absence of any negative impact. One participant states that "In a company that adopts agile software development techniques, I don't think there will be many duplicate bugs and if there is they will be easily spotted without much effect since the team that should be working on the bug is the same," where they relate the little or no impact of duplicates to the agile development techniques they use. Another states that they work in a "critical system" and therefore "duplicates do not occur." Eighty-three percent (5 out of 6) of the responses in this category are linked to the "Minor" impact rating. The remaining response is linked to "Major" impact rating. In their detailed response, this participant links the lack of impact to the environment they work in and later mentions open-source projects as an environment where duplicates would impact the project resources.

A total of 4.2% of the responses argue the presence of duplicate bugs are indicators of other, more serious, problems in the project. These responses are in the "Signal for Concern" category. One participant states that the presence of duplicate bugs could signal an issue with the codebase. Specifically, they say this indicates that the code, which caused the bug, was not written according to "software principles, like SOLID etc." Another respondent identifies duplicates as an issue that could impact the functioning of the system.

Finally, 4.2% of the responses argue that the presence of duplicate bugs can hinder the trust end users have in the system.

6.9 | RQ4: Reasons of duplicate bugs

We investigate the potential reasons for duplication of bugs by analyzing the responses to the survey questions related to the participant's bug-reporting behavior and their explanation for bug duplication.

6.9.1 | Bug-reporting behavior

To begin with, we investigate the types of bug reporters by analyzing the responses to the following question: "In general, who reports bugs in your company? (You can select more than one)." The results can be seen in Figure 9. The most common bug reporter is "Testers" with 45 out of

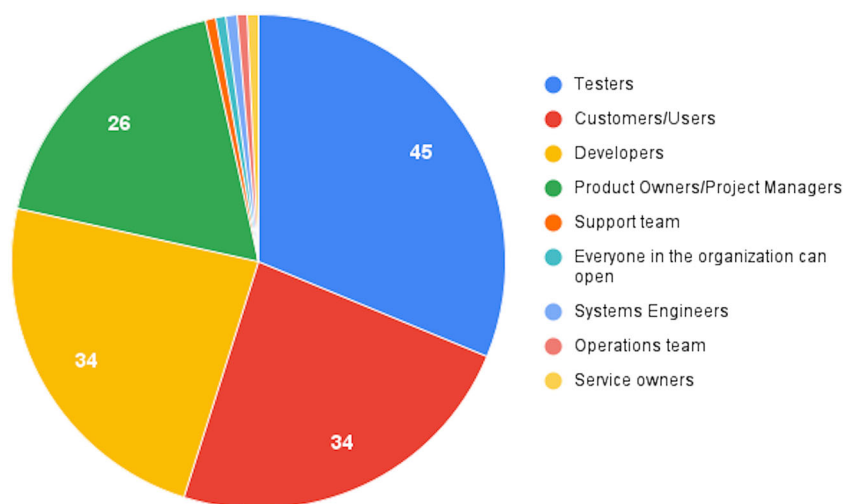


FIGURE 9 Responses to the question: In general, who reports bugs in your company? (You can select more than one). The numbers on the figure correspond to the number of respondents who selected the respective option

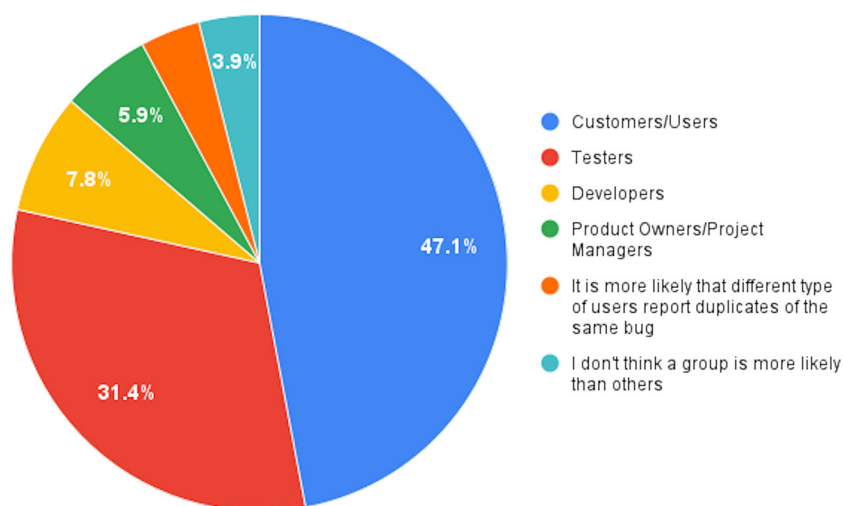


FIGURE 10 Responses to the question: In your experience, which kind of reporters (people who report bugs) are more likely to report a duplicate bug? (Which user category reports more duplicate bugs in your organization?)

51 (88.2%) responses, followed by “Customers/Users” and “Developers” with 34 out of 51 (66.7%) for both. Twenty-six out of 51 (51.0%) responses include “Product Owners/Project Managers” as bug reporters. In this question, participants are allowed to write in any other bug reporter. As a result, we also have the following reporters with one response (2.0%) for each: “Support Team,” “System Engineers,” “Operation Team,” and “Service Owners.” The inclusion of “Support Team” as a reporter is related to their connection with the users/customers. The participant states that when users/customers have a problem with the product, they first contact the support team who then creates a bug report. Additionally, one participant (2.0%) states that everybody in the company can report bugs.

In order to investigate which reporter is more likely to report duplicates, we analyze the responses to the question: “In your experience, which kind of reporters (people who report bugs) are more likely to report a duplicate bug? (Which user category reports more duplicate bugs in your organization?).” The results can be seen in Figure 10. 47.1% of the participants believe that customers or users are more likely to report duplicates. This is followed by testers with 31.5%, developers with 7.8%, and product owners or project managers with 5.8%. Only 3.9% state that one kind of reporter is not more likely than others. Again, 3.9% state that duplicates are generally reported by different kind of reporters.

We investigate the participant's actions prior to reporting a bug, the bugs they investigate, and whether they use an autodetection tool with their bug-tracking system. The questions we ask to investigate these and the results can be seen in Figures 11–13, respectively. Some of the response in Figure 11 are shortened or rephrased for clarity. Majority of the participants (72.8%) investigate the previously reported bugs; 37.3% of the participants investigate them in detail, and 27.5% only skim through the recent bugs; 4.0% searches for the bug report using keywords, and

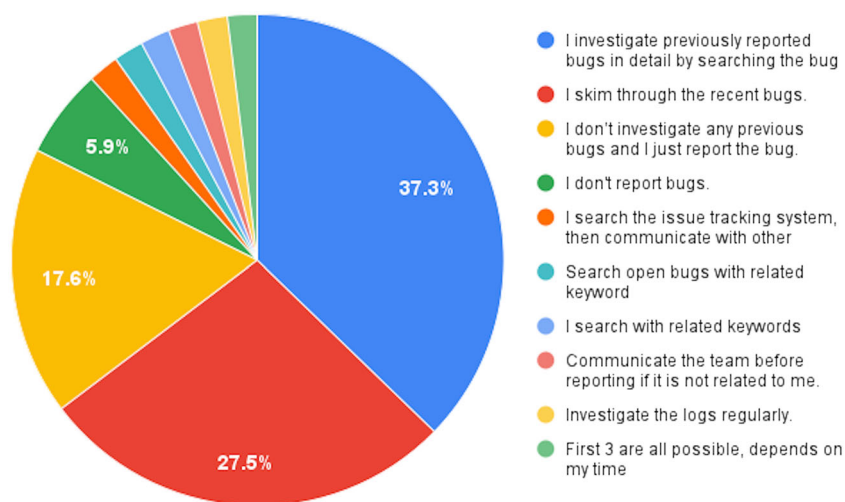


FIGURE 11 Responses to the question: What do you do before reporting a bug?

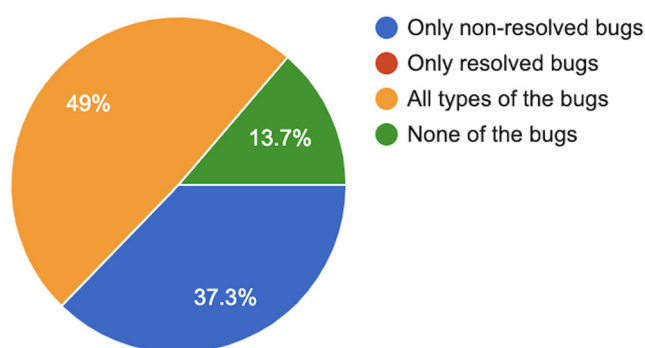


FIGURE 12 Response to the question: While investigating previous bugs, do you investigate all bugs, nonresolved bugs, or resolved (i.e., closed) bugs?

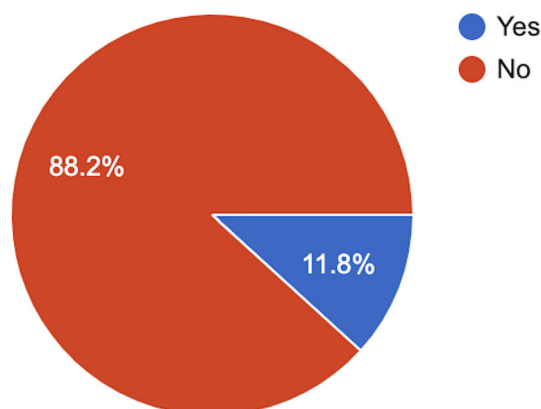


FIGURE 13 Responses to the question: In the bug-tracking system that you use, is there any tool that autodetects duplicate bugs? (e.g., while you type, it shows you potential previously reported bugs so it prevents you from reporting a duplicate bug.)

2.0% searches the bug-tracking system and communicates with their team; 2.0% of the participants state that they search bug reports regularly, and again, 2.0% of the participants state that their ability to investigate bugs or investigate them in detail depends on the time they have (this corresponds to the response labeled as "First 3 are all possible, depends on the time"); 5.9% of the participants do not report bugs.

For the type of bugs they investigate, 49.0% of the participants state that they investigate all types of bugs; 37.3% only investigate non-resolved bugs, and 13.7% do not investigate any. None of the participants only investigate the resolved bugs.

Lastly, we ask participants if the bug-tracking system they use has a tool that autodetects duplicates; 88.2% state that they do not have such tool, and 11.8% state that they do.

6.9.2 | Reasons of duplication

In order to investigate possible reasons of duplication, we focus on the responses to the following question: “What are some reasons for bugs being reported more than once? (Why are duplicate bugs being reported?).” We define the following response categories for reasons of duplication: “Lack of Research,” “Lack of Communication,” “Lack of Testing and Bug-Reporting Process,” “Insufficient Bug Tool Support,” “Bug Exists in Multiple Use Cases,” “Wording Difference in Bug Reports,” “Inexperience of Bug Reporters.” The results can be seen in Table 6 with examples. For this question, we allow one response to be in multiple categories if multiple reasons were given in the response. We discard 3.9% (2 out of 51) of responses because they do not have enough or clear explanation.

The most common reason, with 32.6%, is “Lack of Research.” This category includes the cases where the reporter reports a bug without searching for the bug's existence or without going through the bug-tracking system. An example response is, “When a person (dev/user/etc.) reports a bug that they randomly discovered, they don't browse/search the current bug reports to see if there's already a duplicate. They just submit an issue.” One participant pays special attention to lack of research being an issue with open-source projects, and another mentions “wide user base” as a possible cause. Some responses draw attention to the scope as the “lacking” part of the research by mentioning that reporters do not search through “recent,” “existing,” and/or “open” bugs. In fact, 18.7% of the responses mention open bugs.

A total of 28.6% of the responses are in the category “Bug Exists in Multiple Use Cases.” These responses relate duplication to the bug appearing in different parts or use cases of the system or product. One participant states “But more commonly, the root cause of more than one defect may be the same however the tester may come across this defect in different panels/pages/states/environments. Although from the developer point of view they are all the same defect, from tester perspective they are all different failures.” Another category that is as common is the “Lack of Testing and Bug-Reporting Process.” This category includes responses that relate duplication to mistakes or oversights either in the testing or the reporting process. This is different than “Lack of Research” because the attention is not on the reporter for not searching the backlog but on the process itself not including the necessary steps. For example, one participant states, “Sometimes the codes are tested by more than one test engineer and each of them reports the same bugs.” Here, the attention is not on the test engineer not looking for an existing bug but on the fact that multiple test engineers are working on the same task. Another participant states “not classifying bugs deeply” as a possible reason for duplication. This participant is not linking the cause of duplication to the reporter not searching the bug-tracking system but to the lack of classification of the bug during the reporting process.

The next response category is “Lack of Communication” with 18.7% of the responses relating duplication to the lack of communication between team members. One participant states “Not announcing the opening of a bug report” as their explanation for duplication. In this statement, we can see a clear lack of communication between reporters or team members after a bug is reported.

A total of 10.2% of the responses are in the response category “Wording Difference in Bug Reports.” In these responses, participants argue that duplication occurs due to differing descriptions of the same bug in master and duplicate bug reports. This difference means that reporters are

TABLE 6 Response categories for question: What are some reasons for bugs being reported more than once? (Why are duplicate bugs being reported?)

Category	Percentage	Example
Lack of Research	32.6%	“Not browsing the existing bugs before opening a new one”
Bug Exists in Multiple Use Cases	28.6%	“Different testers encountering the same problem or the same bug appearing in different scenarios.”
Lack of Testing and Bug-Reporting Process	28.6%	“Sometimes the codes are tested by more than one test engineer and each of them reports the same bugs.”
Lack of Communication	18.7%	“When more than one product groups work on same product integration such cases may occur.”
Wording Difference in Bug Reports	10.2%	“Phenomenon in bigger teams or across interacting teams, unclear root cause, huge number of bugs, bad bug description content, thus hardly searchable/findable.”
Insufficient Bug Tool Support	8.2%	“Different test teams/testers, customer, etc. can report bugs more than once, but bug tracking system should help to minimize the duplicates.”
Inexperience of Bug Reporter	6.1%	“Lack of knowledge.”

Note: Forty-nine out of 51 responses are considered, and responses can be in multiple categories.

not able to identify that a report is created for the bug they are going to report. For example, one participant states: "... two different people can describe the same defect differently and can not understand that the other one is talking about the same defect."

A total of 8.2% of the responses relate duplication to the insufficient bug tool support of the bug-tracking system. These responses are in the "Insufficient Bug Tool Support" response category. For example, one participant states that "Bug report pages not being easily navigable/searchable." In this response, the focus is not on whether the research for existing bugs is done but on the inability to do so because of the tool. One participant argues that even though reporters might report duplicates, the bug-tracking tool "should help to minimize the duplicates." Finally, 6.1% of the responses connect the occurrence of duplicates to the overall inexperience of the reporter, and these responses are in the "Inexperience of Bug Reporter" category; an example can be seen in Table 6.

6.10 | RQ5.1: Perceived difference of duplicate and unique bugs according to severity

We investigate the perceived difference between duplicate and unique bugs in terms of bug severity. In order to accomplish this, we analyze the responses to the following question: "In your experience, do you think a bug with a higher severity is more likely to be duplicated? Why?" The results can be seen in Table 7. We discard 3.9% of the responses (2 out of 51) because the response is not understood or it is in conflict with itself. We define the following response categories: "No Idea," "Yes (No Explanation)," "Yes: High Severity Bugs Are More Noticeable," "No (No Explanation)," "No: Duplication Not Related to Severity," "No: High Severity Bugs Are Known Issues," and "No: High Severity Bugs Have Shorter Life Cycle."

We categorize the responses that voice agreement with high severity bugs being more likely to be duplicated under "Yes (No Explanation)" and "Yes: High Severity Bugs Are More Noticeable"; 2.0% of the responses agree with this statement but do not offer further explanation; 30.6% of the responses argue that high severity bugs are more likely to be duplicated because such bugs are more noticeable. Because these bugs are more noticeable, more people could potentially "discover" them and report it independently from each other. For example, one participant states: "Yes because it is more important to solve so it is probably more obvious. Then more people can notice and create an issue about it." Another mentions that developers will be occupied with fixing the high severity bug instead of searching for it in the bug-tracking system. Overall, 32.6% of the responses agree that high severity bugs are more likely to be reported.

We categorize the responses that voice disagreement with high severity bugs being more likely to be duplicated under "No (No Explanation)," "No: Duplication Not Related to Severity," "No: High Severity Bugs Are Known Issues," and "No: High Severity Bugs Have Shorter Life Cycle." Only one response is in both "No: High Severity Bugs Are Known Issues" and "No: High Severity Bugs Have Shorter Life Cycle" because it mentions both as the reason of their disagreement. The "No (No Explanation)" category covers 10.2% of the responses and includes responses that disagree with the statement without an explanation. The most common category here is "No: High Severity Bugs Are Known Issues" with 26.5%. The responses in this category argue that bugs with high severity are the ones that a lot of people know or are notified about, and therefore, the duplication of these bugs is less likely; 16.3% of the responses are in the "No: Duplication Not Related to Severity" category. These responses relate the duplication to factors other than severity or state that it is not related to severity. One participant relates the presence of duplicates to somebody not doing their job correctly and not to severity. Another states that duplication of bugs with high severity is possible but are less likely to happen; 12.2% of the responses are in the "No: High Severity Bugs Have Shorter Life Cycle" response

TABLE 7 Response categories for question: In your experience, do you think a bug with a higher severity is more likely to be duplicated? Why?

Category	Percentage	Example
No Idea	4.1%	"No Idea."
Yes (No Explanation)	2.0%	"Yes."
Yes: High Severity Bugs Are More Noticeable	30.6%	"Yes because it is more important to solve so it is probably more obvious. Then more people can notice and create an issue about it."
No (No Explanation)	10.2%	"No."
No: High Severity Bugs Are Known Issues	26.5%	"No. The team is aware of severe bugs and it is unlikely to be reported again."
No: Duplication Not Related to Severity	16.3%	"No, A badly sentenced and explained bugs are much more likely to be duplicate bugs." "No, duplication can be made for bugs of any severity."
No: High Severity Bugs Have Shorter Life Cycle	12.2%	"Bugs with higher severity are solved quicker than others, to my experience duplicate bugs more frequently occurs in lower severity reports."

Note: Forty-nine out of 51 responses are considered, and responses can be in multiple categories.

category. The argument in these responses is that high severity bugs are prioritized and solved quickly and therefore duplication is less likely. Overall, 63.3% of the respondents disagree that high severity bugs are more likely to be duplicated.

6.11 | RQ5.2: Perceived difference of duplicate and unique bugs according to number of users

We investigate the perceived difference between duplicate and unique bugs in terms of number of users by analyzing the responses to the following question: “In your experience, do you think master bugs (duplicate bugs that is initially reported) have more users involved compared with non-duplicated bugs? (users involved means: being added to CC or becoming watchers of a bug.) Please explain why?” The results can be seen in Table 8. We discard 39.2% of the responses (20 out of 51) because these responses left the impression that the participant does not understand the question.

We define the following response categories: “No Idea,” “Yes (No Explanation),” “Yes: More Users due to Duplication,” “No (No Explanation),” “No: Bugs With More Users Will Not Be Duplicated,” and “Not Related to Duplication.” In this analysis, each response is in a single category; 19.4% of the participants state that they do not have any idea. The responses in this category are different than the discarded responses because it is possible that the participant understands the question and concludes that they do not know the answer; 12.9% of the responses relate the number of users to other factors or states that they do not see a correlation between duplication and number of users. These responses are in the “Not Related to Duplication” category. The other factors in these responses include “priority,” “severity,” and “criticality.”

The responses that state duplicate bugs have more users compared with the unique bugs are split into two categories: “Yes (No Reason)” and “Yes: More Users due to Duplication.” The first category covers 9.7% of the responses. In these responses, the participant agrees with the statement but does not provide an explanation. The second category covers 25.8% of the responses. These responses relate the more number of users in the duplicate bug to the act of duplication. For example, one participant states “yes, reporters of the duplicate bug are added to master.” Another mentions that more developers will be affected and therefore involved in the duplicate bug, especially if it was duplicated because the bug appears in multiple use cases. Overall, 35.5% of the participants agree that duplicate bugs have more users than unique ones.

The responses that state unique bugs have more users than duplicate bugs are split into two categories: “No (No Explanation)” and “No: Bugs With More Users Will Not Be Duplicated.” The first category covers 22.6% of the responses and the second category covers 9.7% of them. The responses in the second category argue that if the duplicates had more users, then they would not be duplicated in the first place. For example, one participant states: “I think no. If more users involved then it is less likely for opening a duplicate.” Overall, 32.3% of the participants believe that unique bugs have more users than duplicates.

To sum up, more participants (35.5%) perceive the difference between duplicates and unique bugs in terms of number of users as duplicates having more users than unique bugs.

6.12 | RQ6: Perceived difference of Categories 1 and 2 bugs

To further investigate the difference between Categories 1 and 2 duplicates, we analyze which category participants perceive to be more important and why. Additionally, we also investigate participant's explanation for our finding related to the time distribution of Categories 1 and 2 duplicates. In order to get informed responses from the participants, we first inform them about these bug categories via descriptions in the survey.

TABLE 8 Response categories for question: In your experience, do you think master bugs (duplicate bugs that is initially reported) have more users involved compared with nonduplicated bugs? (users involved means: being added to CC or becoming watchers of a bug.) Please explain why?

Category	Percentage	Example
No Idea	19.4%	“Not sure.”
Yes (No Explanation)	9.7%	“Yes.”
Yes: More Users due to Duplication	25.8%	“Yes. When a bug is duplicated, it results in more people being involved and loss of productivity.”
No (No Explanation)	22.6%	“No.”
No: Bugs With More Users Will Not be Duplicated	9.7%	“No, I think that if the number of users involved increases, awareness of the bug also increases, in this situation, its duplication potential decreases.”
Not Related to Duplication	12.9%	“Severity level is important in involving users not its being duplicate.” “I don't think there is a strong correlation.”

Note: Thirty-one out of 51 responses are considered, and each response is in one category.

To begin with, we analyze which category participants perceive to be more important. The results can be seen in Figure 14. Then, we investigate their explanation for choosing that category as more important. We define the following response categories: “No Idea,” “Both,” “Category 1: Because They Include Rework,” “Category 1: Because They Are Harder to Detect,” “Category 2: Because This Means the Master Was Not Fixed,” “Category 2: Because They Are Harder to Detect,” and “Category 2: Because They Waste Resources.” We discard 27.4% (14 out of 51) questions because they left the impression that the participant misunderstood the question. The result can be seen in Table 9 with examples.

As seen in Figure 14, the majority of the participants (60.8%) think that Category 1 is more important, and 31.4% think Category 2 is more important; 7.8% of the participants state that they do not have any idea. The participant's explanation for choosing one category over the other is initially categorized into “No Idea,” “Both,” “Category 1,” and “Category 2.” Overall, 56.7% of the participants explain why Category 1 is more important, and 24.3% does the same for Category 2; 5.4% of the participants argue that both are important. The “Both” response category only exists in the response to the open-ended question because participants are not allowed to choose multiple categories in the initial question (whose responses can be seen in Figure 14). Finally, 18.9% state that they do not have any idea.

We further categorize the response category “Category 1” into “Category 1: Because They Include Rework” and “Category 1: Because This Means the Master Was Not Fixed.” The former is more common with 40.5% of the responses relating the importance to the potential rework or extra effort Category 1 duplicates can cause. One participant argues that investigating a bug requires significant effort in bug fixing and the duplication of the unresolved master will also “duplicate the effort”; 16.2% of the responses argue that Category 1 is more important because these bugs are harder to detect than Category 2 bugs. For example, one participant argues that because Category 2 bugs were previously resolved they would be “eliminated more easily” than Category 1 bugs.

We also further categorize the response category “Category 2” into “Category 2: Because This Means the Master Was Not Fixed,” “Category 2: Because They Are Harder to Detect,” and “Category 2: Because They Waste Resources.” The first category is more common with 18.9% of the responses arguing that Category 2 is more important because their presence suggests that the master bug was not fixed properly. Another participant stated that such circumstances indicate that “there is a problem with QA or testing.” The second category covers 2.7% (1 out of 37) of the

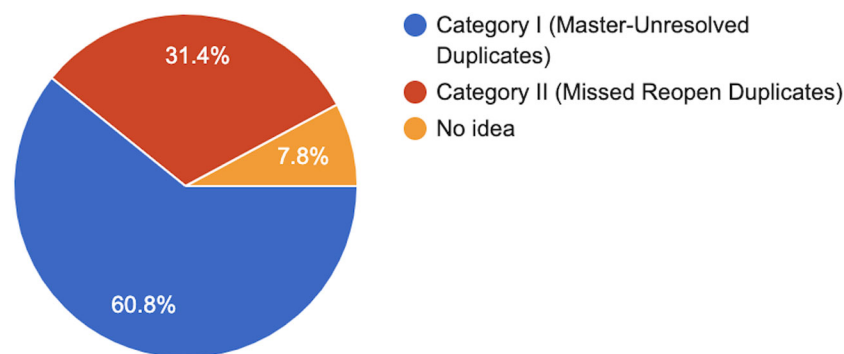


FIGURE 14 Responses to the question: Which category do you think is more important to detect?

TABLE 9 Response categories for question: Why is that category more important?

Category	Percentage	Example
No Idea	18.9%	“No Idea.”
Both	5.4%	“Both of them are very important.”
Category 1: Because They Include Rework	40.5%	“To avoid rework. Worst case is assigning duplicate pairs to different developers and solving the same issue twice.”
Category 1: Because They are Harder to Detect	16.2%	“To reduce analysis and triage actions for duplicate bugs. Category 2 is relatively easy to spot, as another similar bug has been worked on.”
Category 2: Because This Means the Master was not Fixed	18.9%	“It shows that the actual bug haven't fixed correctly which also implies that the work wasn't done.”
Category 2: Because They are Harder to Detect	2.7%	“It is easier for staff to identify duplicates if they are reported in a shorter time interval (in Cat 1). Any duplicates reported in Cat II will take more time to identify.”
Category 2: Because They Waste Resources	2.7%	“Resulted in more waste of man power.”

Note: Thirty-seven out of 51 responses are considered, and response can be in multiple categories.

responses, and this participant suggests that Category 2 bugs are harder to detect than Category 1 bugs. The final category also covers 2.7% of the responses, and this participant states that Category 2 bugs waste man power.

As mentioned in the RQ2.4 subsection, we found that Category 2 bugs are closed earlier than Category 1 bugs. In order to investigate the participant's perception of this, we investigate the responses to the following question: "According to our study, Category 2 (Missed Reopen) duplicates are closed (as duplicates) earlier compared with Category 1 (Master-Unresolved) duplicates. What could be the potential reason for Category 2 (Missed Reopen) duplicates to be detected earlier?" We define the following response categories: "No Idea," "Easier to Detect," "Prioritized," and "Existing Test Cases." The results can be seen in Table 10. We discard 15.7% (8 out of 51) of the responses because these participants either do not explain their response enough and/or the response does not relate back to the question. Additionally, 25.6% of the participants state that they do not have any idea.

The most common response category is "Easier to Detect" with 58.1% of the responses relating Category 2 bugs' earlier closure time to these bugs being easier to detect. Many participants suggest they will be easier to detect because their master bug is already resolved and more people are aware of them. An example can be seen in Table 10. Next, 11.6% of the responses relate the earlier detection to the prioritization of Category 2 bugs. One participant states that when a Category 2 duplicate is assigned to someone, the aim would be to solve the bug "ASAP." The final response category is "Existing Test Cases" with 9.3%. These responses relate the earlier closure to existing test cases or tool support that exist due to their master's resolution. For example, one participant states "the tools can easily detect a duplicate bug after a master bug is resolved."

7 | DISCUSSION

In this section, we discuss the motivation of this research and share what we can learn from our results. Additionally, we also compare the results we have obtained from the analysis of open-source projects and the survey responses. Our RQs focus on the differences between duplicate and nonduplicate and Categories 1 and 2 bugs, the perceived impact, the perceived reason of duplicates, and again, the perceived difference of different bug types.

In the first step of our research, which corresponds to our previous work, we analyze the bug report history of Eclipse and Mozilla to investigate the characteristics of different bug categories in terms of severity, numbers of users involved, bug surface time, and time distribution. The results are presented in the previous section under RQ1 and RQ2. We believe the characteristics we identified for these bug types can be used by practitioners and researchers to improve bug detection. For example, we have found that duplicate bug reports tend to have more users in their CC list than nonduplicate bugs. This analysis could be a guide to develop duplicate bug report detection algorithms. Our further categorization of duplicate bugs into Categories 1 and 2 according to their master's resolution time provides a further understanding of duplicates and can be used to "fine-tune" duplicate detection algorithms in order to detect certain bug types.

In the second step of our research, which corresponds to the survey analysis, we investigate the perception of different bug categories and the impact and reason of duplicate bugs. The results are presented in the previous section under RQ3, RQ4, RQ5, and RQ6. Considering that duplicates are a part of the practitioner's life in the software industry, we believe that investigating their perception of duplicates and the duplication process will increase our understanding of them. Additionally, this also offers us a chance to compare the quantitative results of the differences in bug categories with the practitioners' perception of them.

Our results for the perceived impact of duplicates (RQ3) can be used to understand and justify the necessity of bug detection tools. Considering that the majority of the survey participant's responses is in the "Waste of Resources" response category, the benefit of implementing a bug

TABLE 10 Response categories for question: According to our study, Category 2 (Missed Reopen) duplicates are closed (as duplicates) earlier compared with Category 1 (Master-Unresolved) duplicates. What could be the potential reason for Category 2 (Missed Reopen) duplicates to be detected earlier?

Category	Percentage	Example
No Idea	25.6%	"No Idea."
Easier to Detect	58.1%	"Since the bug is already resolved, the team is familiar with the problem. Detecting unresolved duplicates requires triage of experienced staff."
Prioritized	11.6%	"Category 2 bugs may be prioritized by developers since customers may be disappointed with facing the same issue."
Existing Test Cases	9.3%	"Because, the tools can easily detect a duplicate bug after a master bug is resolved. Also, in Category 1, the people/team can create a bug without knowing each other situation and detecting these kind of bugs is more difficult."

Note: Forty-three out of 51 responses are considered, and responses can be in multiple categories.

detection algorithm or tool is clear. Our results for the potential reasons for duplication (RQ4) also includes insights into the bug-reporting process. These results can increase our understanding of the bug-reporting behavior and the duplication process as we directly share the actions of the survey participants. For example, they can give insights on what aspects the reporting process can be improved to avoid duplication. We found the most common reason for duplication, according to survey participants, to be “Lack of Research” prior to reporting. This analysis can be a guide to develop autodetection tools that notify the reporter about possible duplication while reporting and therefore decrease duplication.

We investigate the difference between duplicate and nonduplicate bugs according to severity and numbers of users both with the analysis of the Eclipse and Mozilla bug-reporting data (RQ1) and the survey responses (RQ5). When severity is considered, the analysis of the bug-reporting data show that duplicate bugs are up to two times more severe than nonduplicate bugs (RQ1.1), while the majority of the survey participants (63.3%) disagree that high severity bugs are more likely to be reported (RQ5.1). Some even state that they do not see any relation between severity and duplication. The possible reason for this difference is the different environment these results stem from. Our quantitative analysis uses the bug reports of two open-source projects while the survey are developers or technical members in the software industry whose experience is based on a smaller scale and possibly more agile working environment. When number of users is considered, results from both the empirical analysis and the survey show that duplicated bugs involve more users (RQ1.2 and RQ5.2, respectively). Our quantitative results show that duplicate bugs have up to 3.5 times more users in their CC list than unique bugs and more participants perceive the duplicate bugs as having more users than unique ones. Having the same result from both analysis supports our previous suggestion of using the CC list size of bugs to detect duplicates.

We also investigate the difference between Categories 1 and 2 bugs with the analysis of the Eclipse and Mozilla bug-reporting data and the survey responses. As mentioned before, duplicate bugs that are created before the resolution of their master bug are Category 1 or Master-Unresolved duplicates, while the ones that are created after the resolution of their master bug are Category 2 or Missed Reopen duplicates. In the analysis with bug-reporting data, we investigate their difference in terms of severity, number of users, surface time, and time distribution. These results are given in RQ2 in the previous section, and they can be used to identify Categories 1 and 2 bugs among duplicates. In the analysis with survey responses, we investigate which category is perceived to be more important and the practitioner's opinion on the time distribution of Categories 1 and 2 bugs. The majority of the practitioners (56.7%) perceive Category 1 bugs to be more important, and the most common reason (40.5%) is the possible rework that these bugs can cause. This can be a guide to “fine-tune” duplicate detection tools to especially detect Category 1 bugs as they are perceived to be more important. We also investigate the survey participant's opinion on Category 2 bugs being closed earlier as duplicate than Category 1 duplicates. When sharing this result in RQ2.4 in Section 6, we suggest that they would be easier to detect because they also have a high CC list size compared with Category 1 bugs. The majority of the participants (58.1%) also state that Category 2 bugs are “Easier to Detect.” This analysis supports our explanation for the early detection time for Category 2 bugs.

8 | THREATS TO VALIDITY

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

8.1 | Internal validity

The duration of fixing (D_F) is measured from the time stamps in the bug-tracking tool. It cannot be guaranteed that this is the actual duration of the developers work 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. However, we believe that deviance is not large enough to affect our findings. We also use UNIX time stamps of the datasets, in which seconds are not included in the datasets. Thus, some duplicate bugs seem to be created at the same time with their master bug which is, indeed, not the case. We also discard duplicate bugs that 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.

Another potential threat is the fact that the severity of bugs stored in the bug trackers might be unreliable. According to Tian et al,²⁹ around 51% of the duplicate bug reports have inconsistent human-assigned severity labels even though they refer to the same software problem with their master bugs. This could be especially true for duplicate bugs; the severity of a duplicate bug may not mean anything for the development team once they realize it is the duplicate of another one. However, master bugs are potentially re-evaluated, so we believe the severity values are more reliable.

To increase the replicability of the study, we shared the datasets that we used along with the implementation of the tools in Zenodo.^{††}

^{††}<https://zenodo.org/record/4114096>.

We also found a number of threats related to the survey of our study. Prior to distributing the survey, we conducted a pilot test questionnaire and clarified the questions that could potentially cause an issue. However, we still found some responses that suggested the participant did not understand the question. Such responses were discarded, and this was explicitly mentioned in Section 6. Additionally, some respondents answered the open-ended questions with “No Idea” or left them unanswered. This could be because, these respondents work in small and/or agile systems rather than open-source projects and therefore might not have much experience with duplicate bugs. However, we argue that the questionnaire is important to understand the perception of duplicate bugs from all type of practitioners.

In the open-ended questions, many participants gave multiple responses in one response. For example, if they were asked about the possible reason of bug duplication, they mentioned multiple reasons that could be the cause. In order to stay true to the insights given by the participants, all of the reasons they mentioned were taken into consideration during the analysis of these questions. An example for such response analysis can be seen in Table 5. Because we allowed multiple “responses” to be present in one, in some cases, the percentages in these tables do not add up to 100. This is the case because we use the number of respondents with valid responses as the baseline while calculating percentages. It is important to note that we did not allow one response to be in multiple categories for the questions where there is a clear distinction between different responses (Yes/No), such as the results in Table 8.

8.2 | Construct validity

This is the extent to which our measure “behaves” in a way that is consistent with theoretical hypotheses and that represents how well the results on the instrument are indicative of the theoretical construct. In our study, we defined two measures; severity that indicates the degree of impact that a bug has on the system and the number of users involved that indicates the perceived importance of a bug. For the first measure, the definition of a bug's severity closely aligns with the degree of impact over the system, and therefore, the measure is consistent. However, we rely on the assumption that the severity field is entered properly and can be trusted. Because there are many fields to be filled in a bug report, the severity field might not be entered with proper care. With the second measure, we intend to measure the perceived importance of the bug. Because there is no direct measure of this attribute in a bug report, we used the closest measure, the number of users involved. Bug reports do not include a field that is directly related to the number of users; therefore, we use the CC list size of the bug reports for this purpose. Here, we make two assumptions: The CC list size represents the number of users involved, and the number of users is a metric related to the perceived importance of the bug. Although these are reasonable assumptions, still, this measure may not accurately represent the intended measure.

8.3 | External validity

We argue that Eclipse and Mozilla projects are sufficiently representative for our study because 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.

The target audience of the survey is people who actively use or have experience with bug-tracking systems, and it was distributed to the authors' network. Even though the survey participants are from at least 27 different companies and have 40 different titles, it is still not possible to draw general conclusions. Hence, we can not assume that results will be generalizable to all software product systems which use bug-tracking systems.

The sample of the survey is a convenience sample where, as mentioned before, it was distributed to the authors' network. While a probability sample could have been a better representation of the population,²⁶ we believe our choice is justified when the number of responses that had to be discarded or did not voice an opinion is taken into consideration. The survey included six open-ended question that means we had 306 such responses. During analysis, 16.0% (49 out of 306) of the responses were discarded, and 8.5% (26 out of 306) voiced lack of knowledge and/or idea. Even though the respondents had an average of 12.5 years of experience, almost a quarter of the responses did not offer an insight, and we believe that this percentage would have been higher with a probability sample.

9 | 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 duplicate bugs, we analyzed different properties of Categories 1 (Master-Unresolved) and 2 (Missed Reopen) duplicate bugs. We also analyzed how duplicate and unique (i.e., nonduplicate) 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 duplicate 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 duplicate bugs are up to two times more severe than unique bugs. Additionally, duplicate bugs have up to 3.5 times more users in their CC list than unique bugs. Because duplicate bugs are more severe, it might be the reason that more people are involved. In this regard, the distinction between duplicate bugs and unique bugs is clear when the severity and CC list size are analyzed.
- The difference between the Categories 1 and 2 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.
- The median of DBST of Missed Reopen (Category 2) is approximately four times the median of the Master-Unresolved (Category 1) duplicates. This indicates that Category 2 bugs are more likely to be surfaced later compared with 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.

According to our survey with 51 participants with an average of 12.5 years of experience in the software industry, the summary of the results are as follows:

- A total of 47.1% of the participants rate the impact of duplicate bugs as “Moderate” and 75.0% explain the impact as waste of resources and effort.
- A total of 47.1% of the participants state that Customers/Users are more likely to report duplicate bugs. The most common reason for bug duplication, with 32.6%, is lack of research prior to reporting a bug.
- A total of 63.3% of the participants disagree that high severity bugs are more likely to be reported.
- More participants (35.5% compared with 32.3%) perceive the difference between unique and duplicate bugs as duplicates having more users than unique bugs.
- A total of 60.8% of the respondents perceive Category 1 duplicates as more important and the most common reason (40.5%) for this is the possible rework such bugs cause.
- A total of 58.1% of the participants believe Category 2 duplicates are closed earlier because they are easier to detect.

We believe that our findings will provide useful insights for both researchers and practitioners to better understand the typical features of duplicate 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) duplicate bugs.
- Researchers can use the enriched (Categories 1 and 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 Duplicates³⁰ 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.

DATA AVAILABILITY STATEMENT

The data that support the empirical findings of this study are openly available in Zenodo at <https://doi.org/10.5281/zenodo.4114096>.

ORCID

Idil Hanhan  <https://orcid.org/0000-0002-5636-5637>

Eray Tuzun  <https://orcid.org/0000-0002-5550-7816>

REFERENCES

1. Sun C, Lo D, Khoo S-C, Jiang J. Towards more accurate retrieval of duplicate bug reports. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011) IEEE; 2011:253-262.
2. Jalbert N, Weimer W. Automated duplicate detection for bug tracking systems. In: 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN) IEEE; 2008:52-61.
3. Davidson JL, Mohan N, Jensen C. 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; 2011:101-108.
4. Cavalcanti YC, Neto PAMS, Lucr dio D, Vale T, de Almeida ES, de Lemos Meira SR. The bug report duplication problem: an exploratory study. *Softw Qual J*. 2013;21(1):39-66.
5. Li M, Shi L, Wang Q. 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; 2019:272-279.
6. Mi Q, Keung J. 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; 2016:1-10.
7. Zimmermann T, Nagappan N, Guo PJ, Murphy B. Characterizing and predicting which bugs get reopened. In: 2012 34th International Conference on Software Engineering (ICSE) IEEE; 2012:1074-1083.
8. Kucuk B, Tuzun E. Characterizing duplicate bugs: an empirical analysis. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2021:661-668.
9. Rocha H, de Oliveira G, Valente MT, Marques-Neto H. Characterizing bug workflows in Mozilla Firefox. In: Proceedings of the 30th Brazilian Symposium on Software Engineering; 2016:43-52.
10. Tan L, Liu C, Li Z, Wang X, Zhou Y, Zhai C. Bug characteristics in open source software. *Empir Softw Eng*. 2014;19(6):1665-1705.
11. Cotroneo D, Pietrantuono R, Russo S, Trivedi K. How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. *J Syst Softw*. 2016;113:27-43.
12. Destefanis G, Ortu M, Counsell S, Swift S, Marchesi M, Tonelli R. Software development: do good manners matter? *PeerJ Comput Sci*. 2016;2:e73.
13. Khattar M, Lamba Y, Sureka A, 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; 2015:50-59.
14. Shihab E, Ihara A, Kamei Y, et al. Studying re-opened bugs in open source software. *Empir Softw Eng*. 2013;18(5):1005-1042.
15. Souza R, Chavez C, Bittencourt RA. Do rapid releases affect bug reopening? A case study of Firefox. In: 2014 Brazilian Symposium on Software Engineering IEEE; 2014:31-40.
16. Caglayan B, Misirli AT, Miranskyy A, Turhan B, Bener A. Factors characterizing reopened issues: a case study. In: Proceedings of the 8th International Conference on Predictive Models in Software Engineering; 2012:1-10.
17. Xia X, Lo D, Wang X, Yang X, Li S, Sun J. A comparative study of supervised learning algorithms for re-opened bug prediction. In: 2013 17th European Conference on Software Maintenance and Reengineering IEEE; 2013:331-334.
18. Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. In: 29th International Conference on Software Engineering (ICSE'07) IEEE; 2007:499-510.
19. Hindle A, Onuczko C. Preventing duplicate bug reports by continuously querying bug reports. *Empir Softw Eng*. 2019;24(2):902-936.
20. Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun C. 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; 2012:70-79.
21. He J, Xu L, Yan M, Xia X, Lei Y. Duplicate bug report detection using dual-channel convolutional neural networks. In: Proceedings of the 28th International Conference on Program Comprehension; 2020:117-127.
22. Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th international conference on Software engineering; 2008:461-470.
23. Sun C, Lo D, Wang X, Jiang J, Khoo S-C. A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Vol. 1; 2010:45-54.
24. Bettenburg N, Premraj R, Zimmermann T, Kim S. Duplicate bug reports considered harmful ... really? In: 2008 IEEE International Conference on Software Maintenance IEEE; 2008:337-345.
25. Lamkanfi A, Perez J, Demeyer S. 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.
26. Kasunic M. Designing an effective survey; 2018.
27. Mann HB, Whitney DR. On a test of whether one of two random variables is stochastically larger than the other. *Ann Math Stat*. 1947;18(1):50-60.
28. Chambers JM, Cleveland WS, Kleiner B, Tukey PA. *Graphical Methods for Data Analysis*: Wadsworth; 1983.
29. Tian Y, Ali N, Lo D, Hassan AE. On the unreliability of bug severity data. *Empir Softw Eng*. 2016;21(6):2298-2323.
30. Reliex. Find duplicates app for Jira; 2014.
31. Kucuk B, Hanhan I, Tuzun E. The implementation of characterization of duplicate bugs and datasets. <https://doi.org/10.5281/zenodo.4114096>; 2021.

SUPPORTING INFORMATION

Additional supporting information may be found in the online version of the article at the publisher's website.

How to cite this article: Kucuk B, Hanhan I, Tuzun E. Characterizing duplicate bugs: Perceptions of practitioners and an empirical analysis. *J Softw Evol Proc*. 2022;e2446. doi:10.1002/smr.2446