Characterizing Bug Workflows in Mozilla Firefox

Henrique Rocha
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte - Brazil
henrique.rocha@dcc.ufmg.br

Guilherme de Oliveira
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte - Brazil
guilhermesfdeoliveira@gmail.com

Marco Tulio Valente
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte - Brazil
mtov@dcc.ufmg.br

Humberto Marques-Neto
Department of Computer Science
Pontifical Catholic University of Minas Gerais
Belo Horizonte - Brazil
humberto@pucminas.br

ABSTRACT

Bug handling represents a major effort in most software projects. To improve this relevant task, software organizations must first understand the current status of their bug resolution process. Although there are plentiful research in bug reports, few of them address the bug handling workflow to better understand and reason about the maintenance process. To this purpose, we report a characterization study focused on the typical workflow followed by Mozilla Firefox developers when resolving bugs. We propose the concept of Bug Flow Graphs (BFG) to help understand the characterization. We analyze 13,564 bugs reported for Firefox in 2015 and we discovered some interesting characteristics of Firefox's bug workflow: (a) when a bug is not formally assigned to a developer it requires ten more days to be resolved; (b) approximately 94% of duplicate bugs are closed within two days or less after they appear in the tracking system (which reveals the efficiency of Firefox's duplicate bug detection procedures); (c) incomplete bugs, which are never assigned to developers, usually require 70 days to be closed; (d) more skilled developers show a faster resolution time than less skilled ones; (e) for less skilled developers a bug usually spends more time waiting to be assigned than being fixed.

CCS Concepts

•**Software and its engineering** → **Maintaining software;** Software post-development issues; Software evolution;

Keywords

Characterization Study; Bug Workflow; Life Cycle of a Bug; Bug Flow Graph (BFG); Mozilla Firefox

SBES '16 Sep 19–23, 2016, Maringá, PR, Brazil

DOI: xx.xxx/xxx_x

1. INTRODUCTION

Typically, open source systems handle maintenance requests under a continuous policy [17]. Maintenance requests or bugs[1] are registered in an issue tracking system (e.g., Bugzilla). Then, developers and volunteers choose their own work [10, 14]. This is a fundamental difference from commercial systems, where bugs are usually assigned by a project manager [9]. For example, the Mozilla Foundation mostly follows a continuous maintenance process to address maintenance requests. The maintenance workflow is uncoordinated, which may result in redundant work and waste of effort [10]. Finally, popular open source systems have an overwhelming number of bugs. For example, over 100K bugs were reported for Mozilla systems in 2015.

In this paper, we provide an empirical study on Firefox[2] bug reports, characterizing their resolution status, paths and time. Basically, our characterization relies on graphs—called Bug Flow Graphs (BFG)—that describe the workflow followed when resolving bugs. BFGs show all stages the bugs go through the maintenance process, and their resolution time information. We claim BFGs provide a lightweighted and visual representation of the general bug fixing process. Our overall motivation is to help open source developers to understand and to resolve bottlenecks and therefore to improve the maintenance process of their systems. Although, there is a plethora of research in bug reports, there are very few studies aimed to characterize bug's workflows.

By computing BFGs for 13,564 recent Firefox bugs, we reveal for example that their resolution time is at least ten days longer when they are not officially assigned to a developer. This information cannot be easily found in the tracking system, while the proposed BFGs help to directly infer it. In summary, our main contribution is to provide a characterization study showing many aspects of the bug handling workflow process followed by a major open source system. Moreover, as a second contribution, our characterization methodology supported by bug flow graphs can also be applied to other studies.

The remainder of this paper is organized as follows. Section 2 describes the bug life cycle workflow followed by Mo-

[1] In this paper, we use the term bug to describe any maintenance request registered in tracking systems.
[2] https://www.mozilla.org/firefox, verified 2016-04-20.

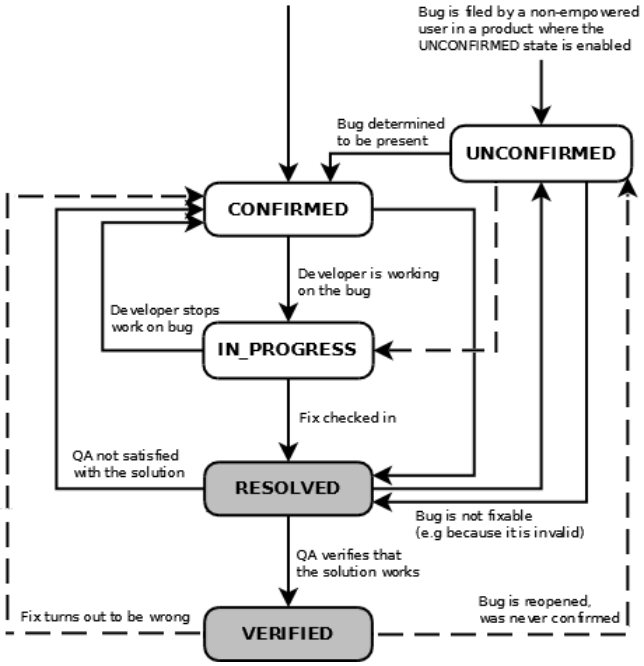


Figure 1: Standard Bugzilla (version 5.x) Workflow [2].

zilla systems. Section 3 describes the concept of Bug Flow Graphs. Section 4 presents and analyze the Firefox bug dataset used in this paper. Section 5 presents the characterization study using BFGs to analyze the maintenance workflow of Firefox bugs. Section 6 reviews related work. Finally, Section 7 concludes and shows possible lines for future work.

2. THE LIFE CYCLE OF MOZILLA BUGS

In this section, we explain the workflow Mozilla developers adopt for handling bugs, also known as life cycle of bugs. We begin by describing the Mozilla customized maintenance process and its workflow (Section 2.1). Then, we define the workflow states, most common transitions, and the possible resolution for bugs (Section 2.2).

2.1 Mozilla Workflow Process

A bug life cycle describes the workflow followed by a bug until its resolution [2]. The Mozilla Foundation employs Bugzilla as the official issue tracking system for all its projects, including Firefox. Bugzilla provides a standard bug workflow (Figure 1) which is composed of the following states: *Unconfirmed*, *Confirmed*, *In_Progress*, *Resolved*, and *Verified* [2].

However, Firefox workflow shows different states than those presented by the standard Bugzilla workflow. We questioned a BMO[3] maintainer, who sent to us the following answer:

"This is because default workflow of Bugzilla changed a few years ago, and Mozilla still uses the old one. (...) However, if it's different to the default for earlier Bugzilla, those will be customizations." – BMO's Maintainer (2015-12-29)

[3]BMO is Mozilla's customized version of Bugzilla. The main page for BMO is at https://bugzilla.mozilla.org (verified 2016-04-20).

Therefore, Mozilla systems are using a customized older version of the Bugzilla workflow. The states found for Mozilla bugs in 2015 are: *Unconfirmed*, *New*, *Assigned*, *Resolved*, *Verified*, and *Reopen*. From these states, three have the same names and semantics as the standard Bugzilla (*Unconfirmed*, *Resolved*, and *Verified*), and two states show different names while representing equivalent information (*Confirmed* ≡ *New*, and *In_Progress* ≡ *Assigned*). Only *Reopen* does not have an equivalent state in the current Bugzilla workflow. Figure 2 shows the states for BMO and the valid transitions between them. {*Start*} represents the states a new bug report can start in the workflow, i.e., it denotes which states a newly created bug can be registered in the tracking system.

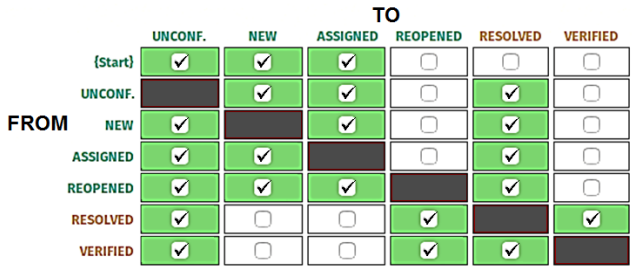


Figure 2: Workflow states for Mozilla's customized version of Bugzilla. The green cells with a check mark represent valid transitions, white cells indicate that there is no transition between the states, and dark grey cells are used when the beginning state is the same as the end state. States with red colored names (Resolved and Verified) indicate that a bug is closed. The information for this figure was provided by a BMO Maintainer in 2015-12-29.

Figure 3 shows the BMO workflow where the vertices represent the states and the edges represent the transitions between each state.

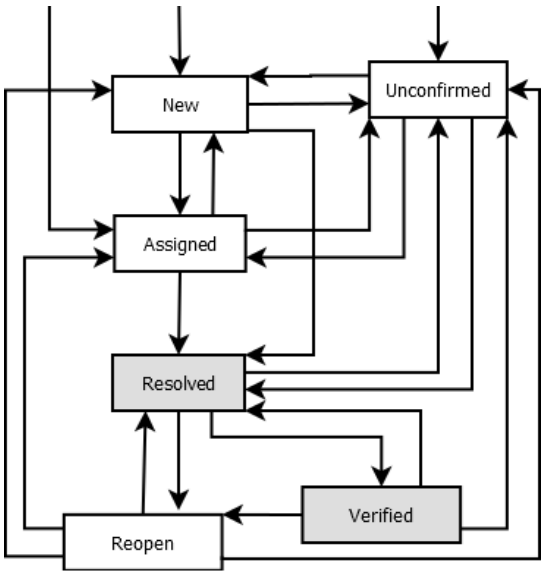


Figure 3: Bug Life Cycle Workflow for Mozilla's customized version (BMO).

2.2 Understanding the Life Cycle Workflow

In this characterization study, we focus on the workflow followed by Firefox bugs, which is the BMO version. Figure 3 shows the possible paths followed by a Firefox bug during its life cycle.

When a bug is registered, it is set as *Unconfirmed*, *New*, or *Assigned*. In some systems, common users can only register unconfirmed bugs. On the other hand, super-users may register unconfirmed, new, or assigned bugs. They may also promote unconfirmed bugs to a new or assigned status.

The bug status changes to *Assigned* whenever a developer is assigned to work on it. If such developer finishes his work on the bug, then its status changes to *Resolved*. Unconfirmed and new bugs may also move directly to *Resolved* without passing through *Assigned*. There are two main reasons for this event: (i) the bug is not valid, or (ii) a volunteer posted a correction before someone was officially assigned to work on the bug. Finally, the quality assurance (QA) team verifies the bug resolution. If the bug passes this verification, its status changes to *Verified*.

If a resolved or verified bug proves to be incorrect or flawed, its status changes to *Reopen* to developers to start working on it again. A reopened bug can return to the earlier states of the maintenance process (*New*, *Assigned*) or be concluded as resolved. A reopen, resolved, or verified bug can also return to *Unconfirmed* if the bug is never confirmed.

A bug is closed when it is either resolved or verified, otherwise it is considered open. A closed bug has one of the following resolution status: *Fixed*, *Duplicate*, *WontFix*, *WorksForMe*, *Invalid*, or *Incomplete*. A successfully fixed bug is marked as *Fixed*. If there is in the tracking system a report describing the same bug, then the bug is marked as *Duplicate*. A bug that will not be fixed is marked as *WontFix*. When a developer can not reproduce the bug, it is marked as *WorksForMe*. If the bug is not valid, then it is marked as *Invalid*. A vague report description that developers cannot understand or a support request are marked as *Incomplete*. These resolution status establish the type of bugs investigated in this characterization study.

3. BUG FLOW GRAPHS

As proposed in this paper, a Bug Flow Graph (BFG) is a directed graph that summarizes a bug life cycle workflow. It provides a visual representation that shows the bugs going through the maintenance process. BFGs are computed directly from the bug tracking system database, without cleaning and preprocessing steps. In a BFG, the nodes represent the status a bug may take throughout the maintenance workflow. The edges represent the transitions between status. Self loops indicate bugs that stay in the same status, i.e., bugs that do not change status and continue in that particular state. Since self loops are not represented in the original workflow, we use dotted edges to draw them. An edges' weight is a multi-valued information $P(Md)$, where P is the percentage of bugs that went through the transition and Md is the median time to make the transition. We use the median because it is more robust to skewed distributions, as it is the case of most state transition times. For better visualization of the BFG, we remove loops with no bugs. However, we claim that removing other edges with no representative value would hinder the understanding of the process. For this reason, these edges are colored gray.

Example: Figure 4 shows a fragment of a BFG computed for Firefox bugs. The BFG reveals that, during the time frame used to collect the bugs, 90% of the bugs that reached a state called *Assigned* advanced to a *Resolved* state. On the median, three days are needed to perform this transition. It also shows that 6% of the bugs returned to the *New* state, but only after spending 17 days in *Assigned*, on the median. Finally, 4% of the bugs that reach *Assigned* do not leave this state. When considered together, their median time in the *Assigned* state is 79 days.

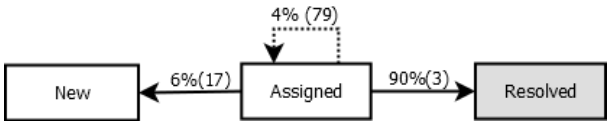


Figure 4: Fragment of a BFG for Firefox

The basic steps to create a BFG are: (i) acquire the bugs status changes; (ii) calculate the edges information (percentage and median time); and (iii) draw the workflow.

BFGs are an improvement over directly acquiring workflow process data from the bug tracking system. Although the information showed by BFGs are indeed present in tracking systems, it is not easily accessible [7, 11]. For example, it is not possible to acquire this data with simple queries, requiring more complex process to extract and to mine historical data. Due to this difficulty, it is reasonable to assume that developers and managers might overlook such information. By contrast, the overall visualization provided by BFGs provides an easier way to verify bug handling workflows.

Although the BFGs presented in this paper are specific to Firefox (and Mozilla) bug workflow, the methodology can be applied to other workflows.

4. DATASET OVERVIEW

Firefox is the second most popular web browser[4], and it is also a very popular member of the Mozilla ecosystem. For this characterization study, we consider all bugs created in 2015 for Firefox, which is composed of 13,564 reports in total. In 2015, Firefox was the third BMO product in number of reported bugs.

4.1 Resolution Status

Table 1 presents the bugs studied in this paper according to their resolution status. *Open* indicates bugs still not resolved when the data was acquired. Table 1 also shows the resolution time (in days) for each status (average, standard deviation, and median). As we can see, more than one third of the bugs are open and waiting resolution (34.95%). The most common resolution status is fixed (26.89%), followed by duplicate (14.97%). The less common resolution is *WontFix* (2.68%).

When we analyze the resolution times in Table 1, duplicate and invalid bugs show the lowest resolution time (average and median). Indeed, Bugzilla provides features to aid in the detection of duplicate bugs, consequently, we expect this kind of bugs to be resolved more quickly. Invalid bugs are also more easily detectable since they describe situations that work on that way on purpose (i.e., it works

[4]According to http://www.w3schools.com/browsers/browsers_stats.asp (verified 2016-04-20).

Table 1: Firefox bugs classified by their resolution status

Status	Bugs		Resolution Time		
	Number	%	Avg	Dev	Med
Open	4,740	34.95%	–	–	–
Fixed	3,647	26.89%	34	53	11
Duplicate	2,030	14.97%	28	63	1
WorksForMe	1,319	9.72%	105	109	64
Invalid	800	5.90%	33	68	2
Incomplete	664	4.90%	171	116	161
WontFix	364	2.68%	86	106	31
Total	13,564	100.00%	55	87	12

like that by design) or they are not a Mozilla bug (e.g., a bug in a third-party library). For this reason, the time to resolve invalid bugs is also low. The highest resolution time is recorded for incomplete bugs (average and median). An incomplete bug includes a vague description to reproduce it, or it is a support request. The high resolution time to close this kind of bug may indicate that developers need better ways to identify incomplete bugs. *WorksForMe* also shows a high average and median values. This is expected because developers may waste a long time trying to reproduce such bugs [8]. *WontFix* bugs present a high average but a more reasonable median. Since only module owners can resolve bugs with *WontFix* status, it is also expected the longer resolution time because few developers have the permission to mark a bug as wont fix. Fixed bugs show a higher average and median resolution time than Duplicates and Invalids. On the other hand, they show much lower average values when compared to *Incomplete*, *WorksForMe*, or *WontFix* resolutions.

The average, standard deviation, and median resolution times presented in Table 1 suggest that the bugs' resolution time do not fit into a normal distribution. We expected the bugs to follow a skewed distribution based on previous research and experience [11]. To verify our expectation, we plotted the resolution times for each status (and considering all bugs together) as a violin chart (Figure 5). This chart shows a very high density of bugs resolved within a few days. Moreover, the plots indeed indicate that the resolution times follow a skewed distribution (except for *Incomplete*). For this reason, measurements like median better represent the resolution data than average and standard deviation.

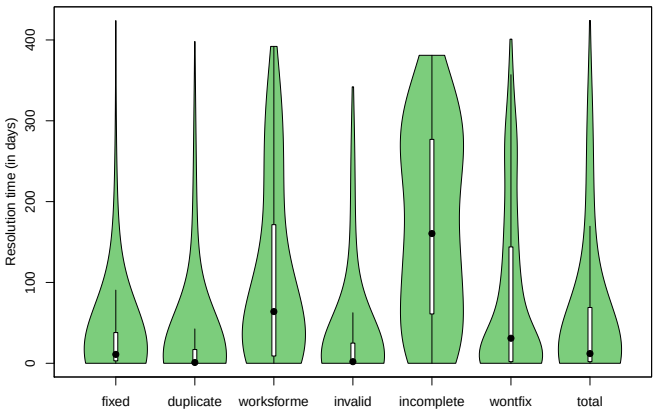


Figure 5: Violin plots showing the distribution of resolution times (in days) per bug status.

Table 2: Types of Firefox users registered in Bugzilla

Type	Number	%
Reporter	4,078	76.14%
Commenter	2,051	38.29%
Patcher	396	7.39%
Developer	410	7.65%
User (total)	5,356	100.00%

We also applied the Kruskal-Wallis test to compare the bugs status resolution times. The test shows that the distributions are different (p-value < 0.001). Therefore, the bug status may indeed impact on the bug resolution time.

4.2 Firefox Users

When we analyze the bugs and their resolution time, we have a general overview of the bugs. However, there is another key aspect to bug handling, the users interacting with the bug tracking system. We therefore consider the distinction among different types of users according to their interaction with bugs. We classify users into four types: reporter, commenter, patcher, and developer (Table 2). A reporter is someone who reports a bug. A commenter is an user who posts comments on bugs. A patcher is an user who posted a patch file for a bug. And a developer is someone who was assigned to handle a bug. For us, an user is someone who interacts in any way with a bug, i.e., user is the union among the four types. However, these types are not mutually exclusive, and it is possible for a single user to belong to more than one user type.

In total, there were 5,336 users interacting with Firefox bugs in 2015. As we expected, the number of reporters and commenters is greater than those of patchers and developers. Patchers and developers are very related groups and share many common users, almost 83% of patchers also fall into the developer category. For this characterization study, we focus on analyzing the developers because they are the users officially assigned to resolve bugs.

4.3 Developers Profile

We classify developers into specific categories according to their bug handling skills. For this study, we exclude the developer identified as "nobody"[5] because such user is not a person. We consider the number of bugs assigned to the developer according to the skill based classification. In our dataset, developers are assigned nine bugs on average, with 23 bugs as standard deviation. The average and deviation indicate that bugs assigned to developers follow a skewed distribution. For this reason, we divide the developers into quartiles (Table 3). We call *Newbies* the developers from the first and second quartiles, i.e., developers who are assigned to work on two bugs or less. The third quartile are *Junior* developers who worked on five bugs or less. *Senior* developers belong to the fourth quartile and they worked on more than five bugs.

We compute the resolution time (in days) for every bug worked by each developer, grouped by the skill category. We add a new category called *Experts*, which are the top-10 de-

[5]"nobody" is used in Bugzilla as a placeholder, until the bug is properly assigned to a real developer. However, a bug can be resolved and its assigned field not updated to the responsible developer, and as such, "nobody" takes credit for the resolution.

Table 3: Developers classification by skill

Quartile	Assigned Bugs	Skill Level
1st	1	Newbie
2nd	2	
3rd	5	Junior
4th	>5	Senior

velopers in number of worked bugs. The *Expert* category is a subset of the *Senior*. Figure 6 shows the resolution times as a notched boxplot. The notches indicate a confidence interval around the median, therefore when two notches do not overlap there is strong evidence that their medians differ [3]. Based on this information we can observe a trend on skill vs resolution time, i.e., the more skilled the developer, the faster he/she is in resolving bugs. For instance, *Newbies* showed an average, standard deviation, and median of 42.3, 56.8, and 18.5 days, respectively. On the other hand, *Senior* developers showed 29.5, 48.9, and 10 days, respectively.

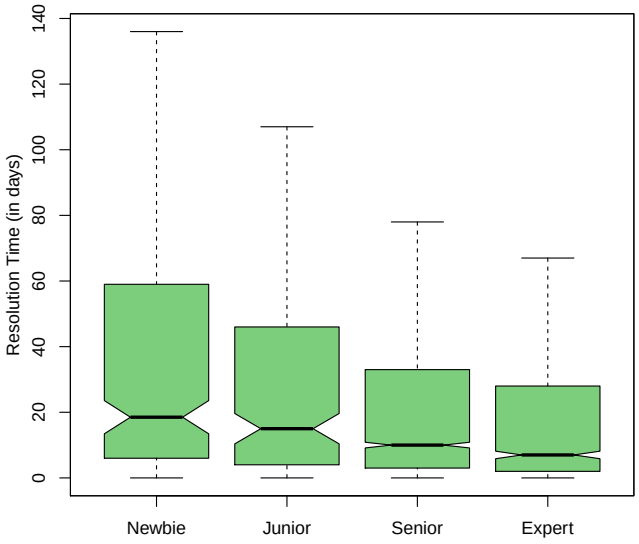


Figure 6: Distribution of resolution times (in days) for each developer skill category.

We also applied the Kruskal-Wallis test to compare resolution times according to developers category. We found that the distributions are different (p-value < 0.001). Therefore, the developers skill indeed impact on the bug resolution time.

5. CHARACTERIZATION STUDY

In this section we present and analyze the results of our characterization study. Our motivation is to perform an exploratory research and to learn lessons from the results we obtain. To guide our research, we elaborate two major research questions:

- **RQ#1:** What are the distinguishing characteristics of Firefox bug resolution workflow?
- **RQ#2:** Does the workflow followed by developers differ according to their skill?

5.1 BFG Workflow Analysis

The previous discussion in the previous section shows a general overview of the bugs registered for Firefox. However, we need to analyze the workflow path followed by these bugs to perform a more thorough characterization. Therefore, to address RQ#1 we create BFGs to analyze such bugs.

5.1.1 Overall Workflow

Figure 7 shows the BFG for all Firefox bugs considered in this study.

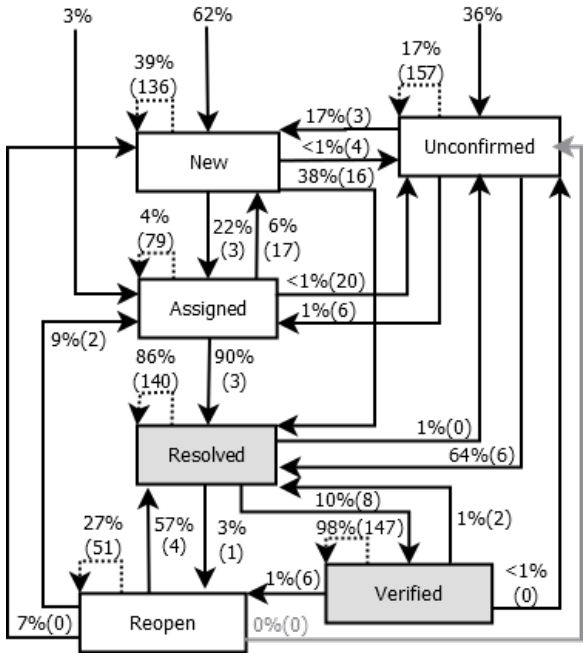


Figure 7: BFG for all Firefox bugs.

By analyzing this BFG, we can make at least the following observations about the workflow followed when resolving these bugs:

- Most bugs start as *New* ($\approx 62\%$). However, there is a large portion of bugs starting the workflow as *Unconfirmed* ($\approx 36\%$). As expected, it is a rare condition for a bug to start as *Assigned* ($\approx 3\%$).
- For the unconfirmed bugs, few of them stay in the unconfirmed status ($\approx 17\%$) and few are confirmed as a new bug within three days (transition *Unconfirmed* to *New*, $\approx 17\%$). Almost two thirds of unconfirmed bugs are directly resolved in six days (transition *Unconfirmed* to *Resolved*, $\approx 64\%$).
- A small proportion of new bugs wait three days in the tracking system until they are officially assigned to a developer (transition *New* to *Assigned*, $\approx 22\%$). On the other hand, a greater proportion of new bugs are directly resolved without being officially designated to a developer after waiting 16 days (transition *New* to *Resolved*, $\approx 38\%$). There are also many bugs that remain open in the *New* status ($\approx 39\%$).
- There is an interesting finding when we observe the resolution times, since new bugs are resolved faster

when properly assigned to a developer. It usually takes six days (three days from *New* to *Assigned* plus three days from *Assigned* to *Resolved*) for a new bug to be resolved if it is assigned to a developer. By contrast, it takes 16 days for a new bug to be closed without being formally designated to a developer (transition *New* to *Resolved*).

- Most resolved bugs stay closed, as either in the *Resolved* ($\approx$ 86%) or *Verified* ($\approx$ 98%) state. Few bugs are verified by the quality control team which usually takes eight days (transition *Resolved* to *Verified*, $\approx$ 10%). Only a small portion of bugs gets reopen (transition *Resolved* to *Reopen*, $\approx$ 3%) and even fewer return to the beginning of the workflow as unconfirmed bugs (transition *Resolved* to *Unconfirmed*, $\approx$ 1%).

Positive Points: Most resolved (86%) and verified (98%) bugs stay closed, which appoints to the efficacy of Firefox developers resolving bugs. Another positive finding indicates that developers are generally fast in detecting incorrect solutions, as most bugs are reopened within one day after being closed (*Resolved* to *Reopen*).

Negative Points: Many open bugs remain in the same state. The time these bugs stay in the beginning states of the workflow are very high (*Unconfirmed* 157 days; and *New* 136 days), which could indicate that they are being ignored or neglected.

Opportunities for Improvements: We found that bugs are resolved ten days faster when properly assigned to a developer. Therefore, tools to assist developers and volunteers on finding bugs of their interest can contribute to reduce the resolution time of Firefox bugs. As examples, we can mention tools to allocate developers to bugs [1] or to recommend similar bugs that can be fixed after a given bug [15, 16]. Another effort can be directed in helping developers finding more open bugs, especially those being neglected for a long time in the *New* and *Unconfirmed* states.

Although this BFG provides a starting point to understand the Firefox workflow, grouping all types of bugs may hide peculiarities of the maintenance process. For example, the BFG in Figure 7 does not discriminate fixed bugs from other types. For this reason, we decide to create BFGs to analyze specific bugs, according to their resolution status as discussed in the following subsections.

5.1.2 *Fixed Bugs*

Figure 8 shows a BFG that considers only fixed Firefox bugs. By analyzing this BFG, we can make at least the following observations:

- Most fixed bugs bugs start as new ($\approx$ 85%), while unconfirmed ($\approx$ 7%) and assigned ($\approx$ 8%) bugs are less frequent. This contrasts with the BFG for all bugs where new bugs show a lower percentage ($\approx$ 62%) as the starting state for bugs.
- The BFG also shows that $\approx$ 56% of the confirmed bugs are formally assigned to a developer within three days (transition *New* to *Assigned*). The remaining 44% are resolved directly after 13 days (transition *New* to *Resolved*).

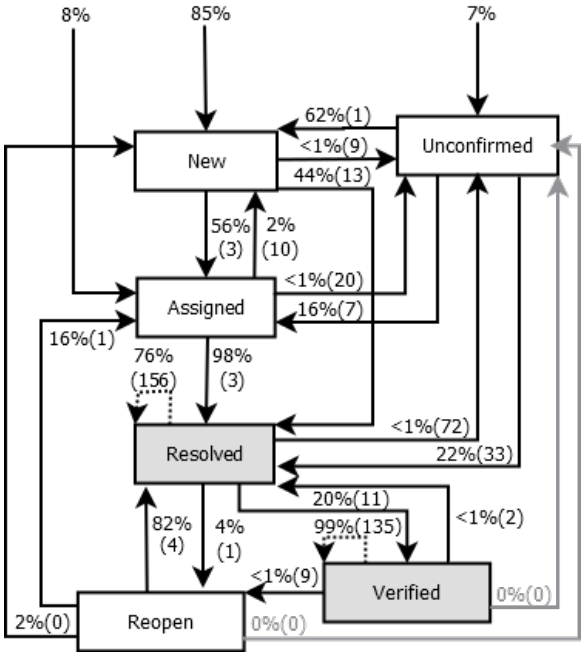


Figure 8: BFG for Fixed Firefox bugs.

- The BFG also confirms the findings we discovered analyzing the BFG for all bugs, which is that bugs not formally assigned to a developer take a longer time to be resolved.
- Only one out of five resolved bugs are verified by the quality control team within 11 days after they are fixed (transition *Resolved* to *Verified*).

Positive Points: Unconfirmed bugs are quickly confirmed within one day as they arrive in the workflow (*Unconfirmed* to *New*). This is important because new bugs show faster paths towards resolution than unconfirmed ones. Moreover, we can see an increase in the percentage of assigned bugs, which takes less time to fix. Even though there is a low percentage of resolved bugs that are verified (20%), even fewer resolved bugs are reopened by an incorrect fix (4%).

Negative Points: Although the percentage of verified bugs is higher than the workflow for all bugs (20% fixed versus 10% all bugs), we claim it is still lower than expected for a mature process. Another negative aspect is the time spent for unconfirmed bugs to be directly resolved, which is much greater than the one showed for all bugs (33 days for fixed, and six days for all bugs).

Opportunities for Improvements: The verification of bugs may be a bottleneck in the workflow, mainly because of the time it takes. The time in days required for a bug to be verified is almost four times the number of days needed for a developer to fix a bug. (three days to fix versus 11 days to verify). We could infer that the quality control team may need more people or tools to improve this verification. We also found that new bugs are fixed faster when properly assigned to a developer. An unassigned bug is fixed seven days later than an assigned one. Therefore, this finding reinforces that techniques to assist developers on finding appropriate bugs can also contribute to reduce the time to fix bugs.

5.1.3 Duplicate Bugs

Figure 9 presents the BFG extracted for duplicate bugs. We analyze duplicate bugs because they are the second most common resolution status for Firefox bugs (Table 1).

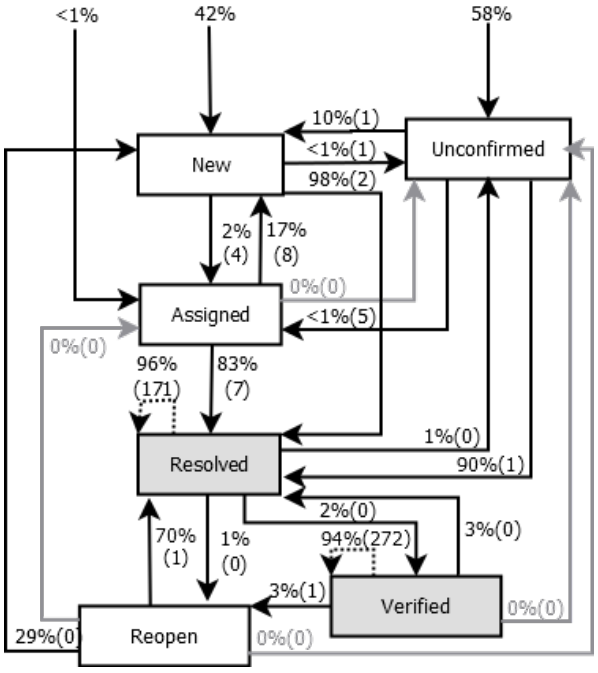


Figure 9: BFG for Duplicate Firefox bugs.

We can make the following observations about the BFG in Figure 9:

- Most duplicate bugs reported as *Unconfirmed* are detected and resolved in one day after their arrival in the tracking system (transition *Unconfirmed* to *Resolved*, $\approx 90\%$). Moreover, 98% of the duplicate bugs reported as *New* take two days to be detected and marked as *Resolved* (transition *New* to *Resolved*).

- The remaining 1% of the duplicated bugs that are not detected in the *New* and *Unconfirmed* states generally take four to five days to be assigned to a developer (transitions *New* to *Assigned*, and *Unconfirmed* to *Assigned*). Moreover, the assigned developer takes seven extra days to discover that the bug is duplicate and resolve it (transition *Assigned* to *Resolved*).

Positive Points: Approximately 94% of duplicated bugs (starting to *New* to *Resolved*: $42\% \times 98\% = 41\%$ plus starting to *Unconfirmed* to *Resolved*: $59\% \times 90\% = 53\%$) are detected and resolved in two days or less, on the median. This shows the efficiency of Firefox duplicate bug detection policies.

Negative Points: Even though there are few duplicate bugs being assigned to developers, these bugs are still wasting valuable effort time from developers. First, the bugs wait four or five days to be assigned (*New* to *Assigned* or *Unconfirmed* to *Assigned*). Then, after wasting seven days of a developer's time, the bug is closed as duplicated.

Opportunities for Improvements: Since duplicate bugs are the second most common resolution, any approach to identify duplicates before wasting developers time is useful. There is also the possibility to improve the current duplicate detection to find them even faster.

5.1.4 Incomplete Bugs

Figure 10 presents the BFG extracted for incomplete bugs. We analyze incomplete bugs because they show the highest median resolution time.

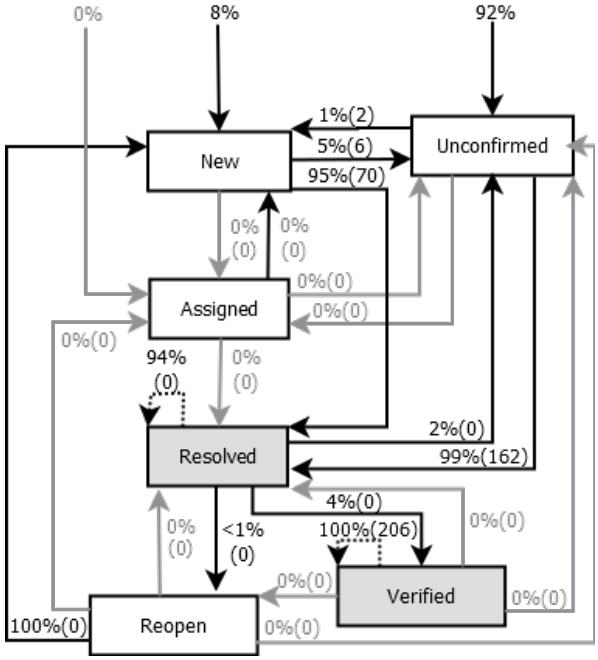


Figure 10: BFG for Incomplete bugs.

We can make the following observations about the BFG in Figure 10:

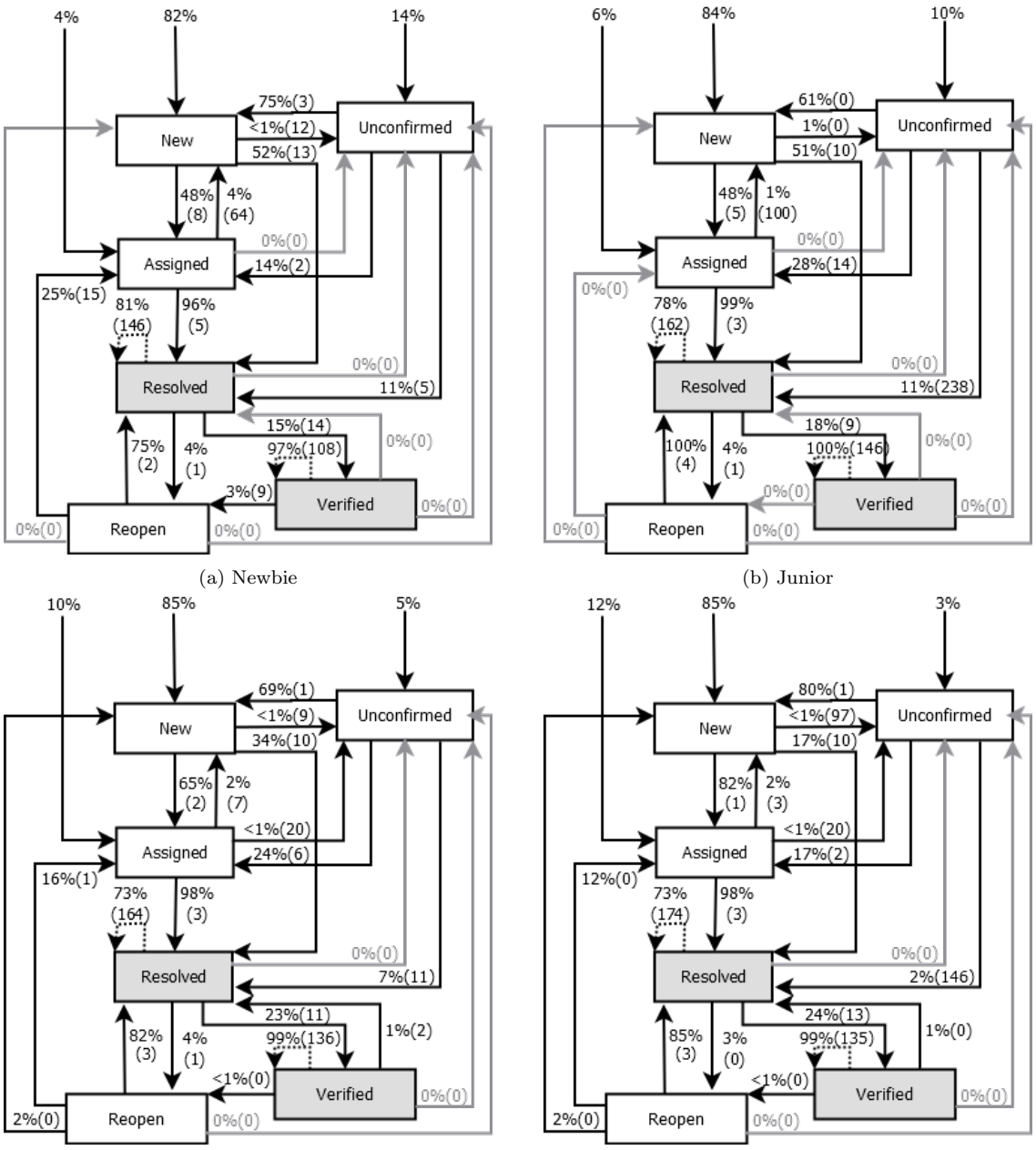
- Bugs marked as unconfirmed usually require 162 days to be resolved (transition *Unconfirmed* to *Resolved*). By contrast, new bugs may take up to 70 days (transition *New* to *Resolved*).

- Incomplete bugs are not assigned to developers. The BFG shows that every in or out transition of the *Assigned* state has zero bugs. This may indicate that Firefox developers are aware when a bug is incomplete and do not get assigned to it.

Positive Points: Incomplete bugs are not assigned to developers, which indicate that Firefox developers are aware when a bug is incomplete and do not allocate their time working on it.

Negative Points: It takes a long time for incomplete bugs to be discovered and closed. The best path takes 70 days on the median.

Opportunities for Improvements: Tools and techniques could be investigated to identify incomplete bugs (just like the ones used for duplicate detection) and reduce the number of days to close them.



(a) Newbie (b) Junior

(c) Senior (d) Expert

Figure 11: BFGs for Developers' Skill Profiles: (a) *Newbie*, (b) *Junior*, (c) *Senior*, and (d) *Expert*.

5.2 Developers Workflow

In this section, we investigate the developers handling bugs even further to answer RQ#2. Figure 11 show BFGs considering only fixed bugs for each developer skill level (Section 4.3). As we can see in the figure, the workflow from each skill level shows different results. We can make the following observations when comparing these BFGs:

- First, we begin by analyzing the assignment of bugs. As we previously discussed (Section 5.1), a bug takes less time to be fixed when it is assigned to a developer. The BFGs show that more skilled developers have a higher percentage of bugs being assigned to them (transition *New* to *Assigned*). Moreover, the time in days for a new bug to be assigned to a developer decreases as the skill level increases, i.e., skilled developers are assigned to a bug report in less days. We can also see an increase on the number of bugs that start the workflow as already assigned in accordance to the developer skill level.

- Analyzing the resolution for assigned bugs (transition *Assigned* to *Resolved*), we can see that the percentage of bugs following the transition are very similar (96% and higher). *Newbies* showed a greater resolution time than the others (five days). *Junior*, *Seniors* and *Ex-*

perts usually take three days to fix an assigned bug. Moreover, if we sum up the time when the bug is assigned (*New* to *Assigned*) and the fixing time (*Assigned* to *Resolved*) the results are eight days for *Juniors*, five days for *Seniors*, and four days for *Experts*.

- An interesting finding is that for *Newbies* and *Juniors* the bugs spend more time waiting to be assigned than being worked on by developers. For *Newbies*, a bug waits eight days in the *New* status until its assigned to a developer, but only five days to be fixed after its assignment. *Juniors* wait five days to be assigned to a bug, while fixing the bug takes three days.

- When we analyze bugs that are fixed directly following the transition from *New* to *Resolved*, developers skill seem to affect the percentage of bugs and resolution times. The percentage shows a clear trend, since the transition *New* to *Assigned* is more common than transition *New* to *Resolved*, in the case of skilled developers

- The more skilled the developer, the less likely is he/she to work on unconfirmed bugs. We can see a decrease in the number of bugs that start the workflow as unconfirmed, as the developer becomes more experienced. Moreover, the percentage of unconfirmed bugs that are directly resolved (transition *Unconfirmed* to *Resolved*) also decreases as the developer is more skilled.

- We can also see a relation between the developers experience and the percentage of bugs that are verified by the quality control team. Skilled developers have a higher percentage of verified bugs.

Opportunities for Improvements: When *Newbies* and *Juniors* are involved, a bug spends more time waiting to be assigned than actually being fixed. Therefore, tools to assign bugs to developers are much more valuable to less skilled developers, and could reduce the fixing time of bugs handled by *Newbies* and *Juniors*.

5.3 Threats to Validity

Construct Validity: The transition times in our results are extracted from the bug tracking system. We cannot guarantee that this is the time the developer is actually working on the bug. Certainly, developers overlap bug fixing with other activities, specially in the case of volunteers.

Internal Validity: We investigated Firefox bugs reported for one year (2015). For this reason, our findings may change if different time intervals are investigated.

External Validity: We analyze bugs reported for a single open source system, Firefox. Therefore, our findings may not be valid to other other open source or closed source systems.

6. RELATED WORK

Joorabchi et al. [8] characterize *Works For Me* bugs from six systems. Their characterization classify *Works For Me* bugs into six categories, and show workflow patterns for these bugs. Guo et al. [6] analyze factors that affect the bug resolution in Windows 7 and Vista. They also propose a prediction model to identify the probability of a new bug to be fixed. Zimmermann et al. [18] characterize which bugs are reopened for Windows 7 and Vista. They also try to understand the main causes to bug reopening by using a survey and a quantitative study. Finally, they show the impact of different bug report features on the probability of bug reopening. The aforementioned characterization studies analyze a specific type of bug report (fixed, reopen, or works for me). Our characterization resulted in findings for several types of bugs by analysing their workflow information.

D'Ambros et al. [5] propose an approach to follow the evolution and maintenance of systems by looking at the bug life cycle. They present two types of visualization techniques: *system radiography* and *bug watch*. Sassc et al. [4] propose a web visual analytics platform, called in*Bug, to visualize bug reports. These works on visualizations can be useful, however, they provide a different view of the bug maintenance process. For instance, neither tools show the transitions and time information we use in our study to report our workflow findings.

Minelli et al. [13] developed a tool, called DFlow, to visualize the workflow of developers. DFlow is implemented in Pharo Smalltalk, and it analyses the developers tasks in an IDE when they are working. This work contrasts with our developer characterization mainly because we focus on the maintenance workflow (i.e., the bug life cycle) and how developers handle the bugs. On the other hand, Minelli et al. focus only on IDE interactions.

Ihara et al. [7] also propose a graph based visualization technique to analyze bug workflow. This work shares similarities with our paper, mainly because both works employ graph and transitions to better understand and visualize the bug handling process. On the other hand, there is also many differences to between the works. Ihara et al. propose a general model in which the workflow is adapted to fit such model. The adaption has the disadvantage to not accurately represent all the states and transitions of the actual workflow followed by developers. Our work represents the actual workflow followed by developers showing all its states and transitions, and our methodology can be applied to any bug tracking system. Another difference is that our paper analyzes different resolution status and their workflow, while Ihara et al. analyse all bugs from the studied system together. Another important difference is that Ihara et al. use the average of days as its main time measure, while we use the median. As we argued in this paper, bug resolution times usually follow a skewed distribution and for this reason median is a more appropriate measure than average when analyzing this data.

Tan and Mookerjee [17] compare the benefits of periodic and continuous maintenance policies. Mockus et al. [14] analyze maintenance and development practices for Mozilla and Apache projects. Marques-Neto et al. [12] propose an approach to evaluate maintenance services, using Maintenance Model Graphs (MMG). As the Bug Flow Graphs proposed in this paper, MMGs also model the workflow followed when processing software maintenance requests. However, there are at least three differences between these models: (a) MMGs are defined by developers and project managers, while BFGs are extracted automatically from tracking systems; (b) MMGs are summarized using K-means clustering algorithm, which requires a non-trivial step to define the number of clusters; (c) MMGs show only the percentage of time spent during workflow steps, while BFGs provide more data specific to visualize bug workflows.

There are several approaches designed to help bug handling and triage. Among them we can list techniques to: (i) predict duplicated bug reports (e.g., Liu et al. [10]); (ii) assign the most appropriate developer to handle a bug (e.g., Anvik et al. [1]), (iii) find similar bugs and help developers to handle more bugs (e.g., Rocha et al. [15, 16]).

7. CONCLUSION

Popular open source systems may face a high number of reported issues. In this paper, we conducted a characterization study focusing on the workflow followed by developers to handle bug reports. To help this characterization, we propose the concept of Bug Flow Graph (BFG) to provide a visual representation of the overall workflow process. BFGs can be used by team leaders to identify critical points in the maintenance workflow and, consequently, address such points to optimize the maintenance process.

We analyzed 13,564 bugs reported for the Firefox web browser in 2015. The BFG for fixed bugs showed that a bug takes at least seven days longer to be resolved when it is not properly assigned to a developer. For all bugs, the BFG shows a greater resolution time when a new bug is not properly assigned. Therefore, tools and techniques to help developers find more bugs can contribute to improve the number of bugs being handled. Moreover, the BFG for duplicated bugs reinforce how techniques and policies for duplicate detection are well used in Firefox bugs, as most of them are detected within a few days.

We also investigated the developers handling the bugs. Our study indicates that more skilled developers usually fix bugs much faster than less skilled ones. We also showed that more experienced developers prefer to be properly assigned to bugs, which have a faster resolution time, than to resolve then directly. We also found for less skilled developers that bugs spend more time waiting to be assigned than such developers spend in actually fixing it.

As future work, we plan to extend our study to consider other types of bugs and systems. We also plan to verify the developers' impressions about our results and the BFGs. Another extension is to investigate how the bug handling workflow has evolved over the years, by characterizing and comparing different time periods. Another line of future work is to implement a web based tool, to automatically generate BFGs for different systems.

Acknowledgments

Our research is supported by CNPq, CAPES, and FAPEMIG. We would also like to thank the Mozilla Foundation for providing us with the bug reports' data from Firefox.

8. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *28th International Conference on Software engineering (ICSE)*, pages 361–370, 2006.

[2] Bugzilla Team. *Bugzilla Documentation - 5.0.2+ Release*. Mozilla Foundation, 2015.

[3] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall Boston, 1983.

[4] T. Dal Sassc and M. Lanza. A closer look at bugs. In *1st IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4, Sept 2013.

[5] M. D'Ambros, M. Lanza, and M. Pinzger. A Bug's life Visualizing a bug database. In *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, pages 113–120, 2007.

[6] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows. In *32th International Conference on Software Engineering (ICSE)*, pages 495–504, 2010.

[7] A. Ihara, M. Ohira, and K. Matsumoto. An analysis method for improving a bug modification process in open source software development. In *7th Joint International Workshop Principles of Software Evolution and Software Evolution (IWPSE-Evol)*, pages 135–144, 2009.

[8] M. E. Joorabchi, M. Mirzaaghaei, and A. Mesbah. Works for me! characterizing non-reproducible bug reports. In *11th Working Conference on Mining Software Repositories (MSR)*, pages 62–71, 2014.

[9] G. Junio, M. Malta, H. de Almeida Mossri, H. Marques-Neto, and M. Valente. On the benefits of planning and grouping software maintenance requests. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 55–64, 2011.

[10] K. Liu, H. B. K. Tan, and M. Chandramohan. Has this bug been reported? In *20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 82–91, 2012.

[11] B. Luijten, J. Visser, and A. Zaidman. Assessment of issue handling efficiency. In *7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 94–97, 2010.

[12] H. Marques-Neto, G. J. Aparecido, and M. T. Valente. A quantitative approach for evaluating software maintenance services. In *28th ACM Symposium on Applied Computing (SAC)*, pages 1068–1073, 2013.

[13] R. Minelli and M. Lanza. Visualizing the workflow of developers. In *1st IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4, 2013.

[14] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[15] H. Rocha, G. Oliveira, H. Marques-Neto, and M. Valente. NextBug: a Bugzilla extension for recommending similar bugs. *Journal of Software Engineering Research and Development (JSERD)*, 3(1):1–14, 2015.

[16] H. Rocha, M. T. Valente, H. Maques-Neto, and G. C. Murphy. An empirical study on recommendations of similar bugs. In *23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 46–56, 2016.

[17] Y. Tan and V. Mookerjee. Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, 31(3):238–255, 2005.

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