Appendix of the paper: Assessing bug prioritization strategies using Bug Dependency Graphs for an open-source project

Unknown Author(s)

July 2020

1 Bug dependency graph

There could be $d \in \{1, 2, \ldots, \Delta\}$ many developers working on the bug dependency graph.

In a bug dependency graph, bug i has the fixing time of t_i as demonstrated in Figure 1.

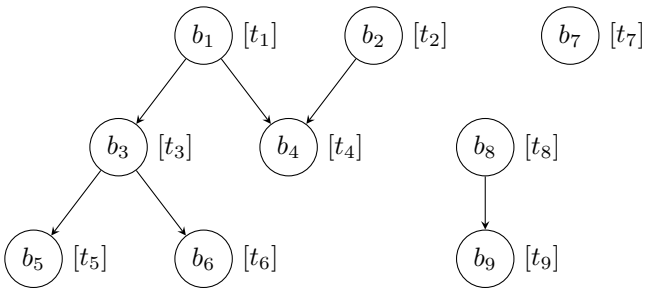

Figure 1: A typical BDG

According to Figure 1, we define bugs that do not have any dependency, e.g., b_7, as *solo* bugs. For the other bugs, the parent bug is called blocking bug and its children are called blocked bugs. A blocked bug can be solved if and only if the blocking bug is solved first. In our example, b_1 is a parent node, and b_4 can not be solved unless b_1 and b_2 are solved. Therefore, the bug dependency plays a crucial rule in bug prioritization.

When the bug b_1 is reported to the project at time t, its dependencies can be diagnosed at time $t + \epsilon$. Therefore, in the network, which we define, the dependency time is independent from reporting time. The evolutionary graph may have few dependencies (few arcs), but it grows as the developers pinpoint bug dependencies.

2 Descriptive analysis

Table 1 describes different available statuses in the Bugzilla environment. A bug at the beginning is "unconfirmed". When a developer confirms it or customers give enough votes to it, then its status will change to "new". A "new" bug will be "assigned" to a developer (or a developer will take the possession), and the resolution process will start. Typically, committers assume ownership of a bug if it is related to the domain they feel confident about or related to other bugs on which they are working. After it is assigned, the assignee will resolve the bug (with the possible resolutions of "fixed", "duplicate", "won't fix", "works for me", or "invalid"), and either is verified and closed or is reopened since the originator is not satisfied by the solution [1].

Figure 2 shows the frequency of each status in the system. Most of the bugs are labeled as resolved (76,638 bugs); however, relatively few bugs are verified (9,455 bugs). The system currently has 10,599 new bugs, and the developers are formally working on 223 of them.

Table 1: Different Status in Bugzilla and its definition

	Status	Description
Resolved / Verified	Duplicate	The bug is a duplicate of another bug already in the system
	Incomplete	Further information is needed.
	Works For Me	Someone in QA has tested this bug and is unable to reproduce it using the provided steps and/or test documents.
	Invalid	The bug report is invalid or does not related.
	Fixed	The bug has been fixed by a particular commit.
	Won't fix	This may be a bug, but it is not planned to fix it for some reasons, i.e. it is not urgent or it doesn't affect many users.
	Inactive	Resolved bugs in Firefox-related components that have no activity in at least one year.
	Moved	Used, when Bugzilla is not the proper place for the report.
Unconfirmed		Nobody has confirmed that the bug is real.
Others	New	Newly introduced bugs
	Assigned	A specific developer is working on the reported bug or enhancement request.
	Reopened	The bug, which was supposed to be resolved in the current version, is creating problems, and needs to be reopened.

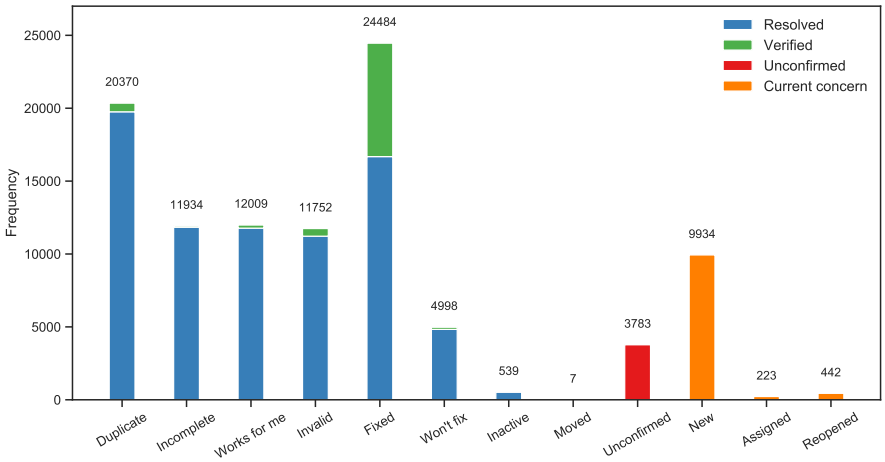

Figure 2: Frequency of different status

2.1 Analysis with three-month breakdown

Simulating short-term decision making of bug prioritization, we break down the whole dataset in three-month batches. The central assumption is that policymakers want to decide on bug prioritization for the upcoming quarter. Therefore, they narrow down their horizon to the next three months. In that case, at the beginning of every period, we reset the bug dependency graph and start from scratch. By applying this assumption, we can capture short-term decision-making in a real BDG, no matter how the system has evolved before the reset point.

Therefore, our experiment encompasses 40 different time horizons for the last decade, repeated five times to eliminate any bias in the output. The strategies is also repeated for different numbers of developers and the relative performance of the strategies remains the same.

RQ1: How do different bug prioritization strategies perform in terms of external indexes?

The performance of the strategies is examined based on the ground truth. Table 2 indicates that severity is an important factor if we aim to solve more bugs. In reality, bugs with higher severity are prioritized and have

shorter fixing time. *Children's severity* as a way of combining the depth and severity of the bugs eliminates more bugs in the long-run. *Max degree* strategy minimizes the maximum degree of the graph and, along with *Max depth* and *Children's severity*, has the smallest depth of the graph. Moreover, it serves as the best algorithm to reduce the depth of subgraphs and, consequently, to diminish the complexity of the BDG. Developers who triage bugs based on their severity, accomplish the task of having less severe remaining bugs in the product. Concerning short-term bug prioritization process, although there is no significant difference between strategies in terms of the number of solved bugs, graph-related approaches are able to reduce the number of dependencies and depth of the BDG by 47.3% and 34.1%, respectively, improving long-term sustainability of the issue tracking system.

Table 2: Short-term (three-month) validation of different bug prioritization strategies under external indexes. The numbers represents the average values after five iterations.

Strategy	The # of daily solved bugs	The # of daily arcs	Max degree	Max degree centrality	Max depth	Max depth centrality	Mean subgraph's depth	Mean severity
Max degree	0.581	0.590	**0.261**	**0.035**	**0.274**	0.036	**0.092**	3.154
Max depth	0.581	0.597	0.263	**0.035**	0.275	**0.035**	0.093	3.152
Max {degree + depth}	0.578	0.624	0.274	0.036	0.287	0.037	0.095	3.151
Max severity	**0.593**	1.019	0.380	0.045	0.399	0.046	0.140	**3.080**
Max {degree + severity}	0.590	0.626	0.272	0.036	0.290	0.036	0.096	3.082
Children's degree	0.581	0.609	0.266	0.037	**0.274**	0.037	0.095	3.155
Children's severity	0.590	**0.582**	0.277	0.039	0.286	0.039	0.096	**3.080**
Random	0.583	1.105	0.395	0.048	0.416	0.049	0.146	3.152

External indexes validate the reliability of the model since all outputs comply with our expectations of the dependency graph's evolution.

RQ2: How do different bug prioritization strategies perform in terms of internal indexes?

In order to evaluate the performance of the bug triaging strategies, we use intact metrics that the bug fixing process do not use. Table 3 summarizes the performance of different strategies based on internal indexes.

Table 3: Short-term (three-month) evaluation of different bug prioritization strategies under internal indexes. The numbers represents the average values after five iterations.

Strategy	Mean votes	Mean comments	Max authority	Mean authority	Max hub	Mean hub	Mean harmonic centrality (10^3)	Max harmonic centrality (10^3)
Max degree	1.173	13.266	0.222	0.173	**0.045**	**0.030**	9.842	**53.594**
Max depth	1.221	13.282	**0.220**	**0.171**	0.046	**0.030**	9.629	53.655
Max {degree + depth}	1.313	13.505	0.228	0.179	0.047	0.031	10.099	54.652
Max severity	1.201	13.185	0.284	0.212	0.061	0.039	12.068	70.759
Max {degree + severity}	**1.131**	13.125	0.231	0.180	0.046	**0.030**	**9.591**	54.176
Children's degree	1.227	13.557	0.229	0.178	0.047	**0.030**	10.083	55.985
Children's severity	1.180	**12.793**	0.240	0.190	0.048	0.032	10.568	56.955
Random	1.420	13.723	0.309	0.223	0.067	0.043	13.408	76.453

Children's severity strategy is the best to address the highly-debated bugs, i.e., more number of comments, whereas *Max {degree + severity}* strategy concerns the voice of users by solving bugs with the highest number of votes. This is equivalent to a 20.4% improvement in satisfying customer needs compared to the baseline. In terms of network-related indexes, *max depth* strategy has the best performance to lower the hub and authority of the BDG. *Max degree* and *Max {degree + severity}* strategies outperforms others regarding the maximum and average harmonic centrality of the BDG, respectively. Graph-based strategies reduce authority, hub, and centrality of the graph from 23.3% to 32.8% compared to the predefined baseline. Nonetheless, the results show that there is no single nostrum to cover all aspects of the problem, but graph-based approaches outperform the baseline in every internal aspect.

References

[1] M. P. Barnson, J. Steenhagen, and T. Weissman, "The bugzilla guide-2.17. 5 development release," *The Bugzilla Team*, vol. 88, 2003.