Discussion and Implications

We discussed the most important implications in the paper. Here, we elaborated on other implications of the findings discussed in the paper:

Assessment of Firefox's Resolution Process. Our methodology, including the derivation and analysis of the sequences, process, and patterns, allows us to identify potential anomalies in Firefox's issue resolution process. For example, through the analysis of RQ1, we identified a lack of QA activities, especially VERIFICATION. While CODE REVIEW is consistently documented in recent years, we found it was lacking before 2019. Since Firefox's code review tools have been integrated with the issue tracker, we can confidently say that CODE REVIEW was missing in past issue reports. Given the importance of these activities to ensure software quality, Mozilla should monitor these activities closely. One way to do so is by promoting the record of QA activities in the issues, *e.g.,* via extra tool integration with the issue tracker.

The identified resolution patterns and sequences can be useful in identifying anomalous instances of Firefox's resolution process. For example, **`A,R,(SD,I,(CR|V))+`** is a complex pattern that indicates repetitive sequences of SOLUTION DESIGN, IMPLEMENTATION, CODE REVIEW, and VERIFICATION. By using this pattern and inspecting the stage sequences, we identified issue #593387 [151], which includes 12 commentators and a sequence of 15 stages with 5 repetitive combinations of the stages above. By inspecting the discussion of the issue, we found that after ANALYSIS and REPRODUCTION, the assigned developer directly implemented a solution (**R**) without any design discussion (**SD**). However, this solution was deemed incorrect by other developers via CODE REVIEW. This prompted a repetitive discussion of a better solution to the problem (**SD**), including iterative implementation (**I**), code review (**CR**), and verification of the solution (**V**). We argue that engaging in SOLUTION DESIGN before IMPLEMENTATION would have led to a more optimized process and potentially less effort involved. We found at least 6 more issues similar to this scenario (*e.g.,* issues #667586 [152], #947523 [154], and #687754 [153]).

Enhanced Process Recording via Interactive Tooling. We found that issue reports did not contain traces of all the issue resolution stages. We argue that these traces are needed to support developers and other stakeholders perform a variety of tasks, such as solving new issues that may require solutions employed in past issues, investigating reopened issues and regressions, documenting the rationale of code changes, and assessing the resolution process. For example, providing information about reproduction attempts in the issues, including configuration environment, used inputs and outputs, and other data, can help developers create test cases and better diagnose the issues.

However, providing more information on the issues may become a burden for developers as their primary goal is to solve issues. To alleviate such a burden, advanced tooling is needed to record useful information while developers are going through the resolution process. We envision tools running in the IDE, developer communication tools, and other systems, collecting useful information about the resolution process, interacting with the developers when needed without causing much disruption, and registering this information in the issues. Issue reports are a good place to record this information as it is the primary artifact for tracking and coordinating the issue resolution process.

Issue Resolution Effort Estimation. Our study found that certain issues and problem types tend to require high or low effort to solve, as indicated by the complexity of the derived patterns. For example, code design, defective functionality, feature development, UI issues, and crashes tend to have more issues solved with complex patterns. This information can be useful for Firefox developers to estimate the effort it may take them to address assigned issues as well as prioritize their issue backlog.

Since the derived patterns are proxies for resolution effort, we could leverage them as extra information to estimate the effort the team may take to solve an incoming issue. For example, if a new issue is similar to the previous one, a model could retrieve the pattern of the old issues, and based on the stages in the pattern and problem category/class (plus the information existing models use [155, 156]), it could suggest a potential effort level (*e.g.,* low/medium/high effort).

Bibliography

[151] Firefox bug #593387. https://bugzilla.mozilla.org/show_bug.cgi?id=593387, 2024.

[152] Firefox bug #667586. https://bugzilla.mozilla.org/show_bug.cgi?id=667586, 2024.

[153] Firefox bug #687754. https://bugzilla.mozilla.org/show_bug.cgi?id=687754, 2024.

[154] Firefox bug #947523. https://bugzilla.mozilla.org/show_bug.cgi?id=947523, 2024.

[155] M. Hamill and K. Goseva-Popstojanova. Analyzing and predicting effort associated with finding and fixing software faults. *Information and Software Technology*, 87:1–18, 2017.

[156] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1042–1051. IEEE, 2013.