Interview Methodology and Results

Goal

To conduct semi-structured online interviews with Mozilla developers to gather detailed feedback on potential use cases of the identified issue resolution patterns.

Research Question

The interview aims to answer the following research question:

1. What are the potential use cases of the patterns?

Interview Protocol

The interviews were conducted over Zoom and were structured into four sections:

1. Participant's Background: Participants were asked to share their background and experience in software development and issue resolution at Mozilla and other companies.
2. Mozilla's Issue Resolution Process: Participants were asked to describe Mozilla's issue resolution process (both prescribed by Mozilla and implemented by developers) as well as the specific approaches they follow.
3. Research Presentation: The research team presented the study's goals, methodology, and findings, including the identified patterns. Participants were encouraged to ask questions about the patterns and findings.
4. Question-Answering: Participants were asked 11 open-ended questions to provide feedback on the identified patterns, with a focus on understanding their potential benefits for Mozilla. The main questions asked about the patterns' overall usefulness, potential to train new developers, estimate issue effort, solve new issues, and evaluate Mozilla's issue resolution process.

Follow-up questions were asked when clarifications or additional information were needed. Please see **"Interview-Protocol"** for more details on interview protocol.

Finding Potential Participants

Our target population consisted of Mozilla developers and other stakeholders with experience in software development and issue resolution at Mozilla, particularly within Mozilla Firefox. To identify potential participants, we employed several strategies. We explored the Mozilla Research [1] website to find researchers interested in software engineering. We also visited the LinkedIn profiles of Mozilla Corporation developers, focusing on those involved in the issue discussions we had annotated. Additionally, we searched Mozilla's issue tracker to identify users with the highest number of reported, assigned, and resolved issues. We also explored Mozilla's Forum [3] and Matrix [2] systems to identify developers who contributed to the annotated issue discussions. These strategies led to a shortlist of 42 potential participants, all of whom were invited to participate via email.

Participants Background

Two developers responded to our call and participated in our interview. Both are current Mozilla developers with 7 to 11 years of experience at the company. D1 works on the SpiderMonkey JavaScript engine, which interfaces with Mozilla Firefox, while D2 is a developer for Firefox's WebRTC component. Both have extensive issue resolution experience, having resolved between 496 and 858 issues and contributed to approximately 5K to 14K issues. Additionally, they have 6 to 8 years of prior development experience at companies such as IBM and Skype.

Interview Answers Analysis

With participants' consent, we recorded and transcribed the interviews using Zoom to facilitate the analysis of responses. As the transcripts could contain errors, we manually reviewed both the transcripts and videos, correcting inaccuracies such as misspellings, incorrect phrases, and punctuation. Based on the revised transcripts, one researcher analyzed and grouped the participants' answers to each question into themes representing use cases of the patterns. A second researcher reviewed these answers and themes for accuracy. Any misinterpretations or discrepancies were resolved through discussion. Finally, we answered the research question by enlisting the potential use cases of the identified patterns suggested by the two Mozilla developers.

Results: Use Cases for the Patterns

The two Mozilla developers (hereon referred to as D1 and D2) mentioned the following use cases of the derived issue resolution patterns:

1. Identifying issues with a complex resolution: D1 suggested that the identified patterns could be used to detect issues with repetitive stages, potentially signaling excessive time and effort required by developers to resolve them. D1 illustrated this use case as follows: "It may be interesting to have bugs which have gone down particularly complex paths be highlighted with a question of like: Okay, it feels like something went wrong here; like, if you went through three different implementations and if you went through three different verifications and it still didn't work; like something went wrong". D1 also said, "I could imagine that this sort of thing [the patterns] could be useful to highlight like something has happened that was bad in this bug. We spent a lot more time and effort and energy on this than maybe we expected to. Given that we thought we had a solution and we went through multiple solutions."

D2 also suggested that a tool that identifies issues having a complex resolution could be useful: “It might be interesting to have some sort of tool that watches the bugs and when it sees this whole snowball effect that will happen sometimes with these bugs maybe be like, okay, hey, this looks pretty heavy maybe we should like you know, make sure that a product person knows about it and knows that, hey, this is chewing up a lot of time. And you should be aware that this is a difficult bug that is going to probably take up a fair bit of time for maybe multiple engineers.”

D1 further emphasized that the purpose of this detection is to understand why the process is taking longer than expected and to implement timely corrections. D2 echoed this opinion, underscoring the importance of early detection for issues that are particularly complex to resolve. As D2 put it, “detecting when a bug has fallen through the cracks. That’s a thing that happens uh and it’s good to know that it’s happening and it’s good to know why it happened. So detecting that earlier would be pretty useful, I think.”

2. Identifying issues not following the expected process: D1 noted that certain issues are expected to follow specific workflows, and if a tool detects deviations from these workflows, it could alert Mozilla developers to take corrective actions. D1 illustrated this scenario with issues that require human verification: “It’s more about maybe highlighting common causes for why issues end up doing these sort of cycles. [...] or even issues that should follow a certain pattern and maybe aren’t currently structured to [follow it]. So for example, some bugs require human verification. We can’t automate the test cases. It requires, you know, installing third-party software on a machine. Sometimes it requires human verification. And it’s up to developers to highlight when this is the case. And so if you had a tool that said, hey, this bug that you open looks like it might need some third-party support for verification. That might actually be a helpful thing.”
3. Identifying potentially complex code components: D2 highlighted the potential to trace Mozilla Firefox’s code components or modules associated with issues resolved using complex patterns. The rationale is that such process complexity may arise from quality concerns within these components, such as technical debt or accumulated code complexity. As D2 put it, “you could track the complexity of the issue resolution process in a given module. And get insights like, hey, it looks like most of the time when you touch this area of code it ends up being a slog. Like it ends up being this just thing that runs on and on and on and requires multiple iterations and pulling in multiple people and like trying to land it and then finding out it doesn’t work. And that could be a flag for a, hey, maybe it’s time to refactor this? Maybe it’s time to clean this up. Maybe you’ve got some pile of technical debt that needs some attention over here. That would be maybe useful for people in product management to be able to see, hey look, this part of the code base is a tar pit and we probably want to spend some resources making it less ornery.” D2 also mentioned that detecting the patterns in issue discussions and associated code components/modules could be used as evidence of code complexity that might help justify a refactoring effort.
4. Improving bots to detect unsolvable issues: D2 suggested that the patterns could be used to inform engineers to enhance the heuristics of existing bots that process Mozilla

issues, helping identify issues likely to be unsolvable or particularly challenging to resolve. As D2 put it, “If you could identify signs that this bug is not going to be solved or is about to fall through the cracks that would be pretty cool. We already have bots that attempt to use heuristics to mark bugs that it thinks, hey, it looks like something’s gone wrong here. Like if this has fallen through the cracks somebody needs to look at it and figure out why work is halted on it. And so we have some bots that do that kind of work. So it could be useful from that standpoint. So insights from this might be useful for the engineers that work on these bots. That kind of try to keep an eye on things and look for areas where things have fallen through cracks or gone off of everything.”

5. Suggesting and decomposing meta-issues: Both D1 and D2 noted that issues with complex resolutions (as indicated by the complex patterns) might represent meta-issues (a.k.a. meta-bugs): large issues that could be broken down into smaller, more manageable issues. They suggested that a tool capable of flagging such cases and proposing possible decompositions would be highly beneficial. As D1 put it, “We have this idea of a meta bug, which is a bug which hosts a whole bunch of related bugs. The tool could come out and say. This kind of sounds like a meta bug And then maybe you could spit out a suggestion of like, could this be split? You know, here are some topics that it sounds like you could split this down or something like that.” D1 also supported this idea stating that “It might be interesting to see which modules tend to have more complicated life cycle for bug fixes. It might be useful to have metrics on this in order to kind of identify areas whereas we need to be making an effort to take smaller bugs and break bugs up into smaller pieces. Instead of having a Gigantic bug that, you know, spills out into a 40 change set Sequence of patches.”
6. Training junior developers: Both D1 and D2 noted that the patterns could serve as valuable training tools for junior Mozilla developers, providing insights into the practical aspects of Mozilla’s issue resolution. D1 noted that junior developers often have high expectations and tend to approach issues linearly, aiming to design and implement a perfect solution. However, D1 emphasized that the patterns demonstrate that solution design and implementation are often incomplete and imperfect, as the issue-resolution process is typically incremental and iterative, typically involving multiple cycles of code review and verification. As D1 put it, “one of the things that would be good is to highlight common patterns for developers. [...] If we can tell people it’s like you can expect that sometimes you will write a solution. And it will not be the solution you land. And that’s okay. like it happens. It’s normal. It’s a normal part of the process, particularly for junior developers, this can be very helpful. Because people set a very high expectation for themselves to get it perfect. [...] so if we have these patterns where we can be like, no, look, like, listen. During your onboarding, here’s a normal pattern for doing development at Mozilla. you will probably have to iterate. You will probably have to go through this solution more than once. And that’s okay. That’s expected. It’s part of the job.” D2 implied that the patterns could help junior developers learn about the process that others followed to solve prior issues, as “most of the time with a new developer, they do spend a lot of time looking at old bugs or you know maybe recent bugs that have already been closed in their module to kind of get an idea of “what the general expectation is around?”, “how done things need to be before you put them up for code review?”, “how

> much feedback you’re expected to solicit from your teammates on whatever solution you have in mind?”, “what kind of testing is expected?”, and “where you would be writing a particular type of testing...?"

Most of the use cases suggested by the developers imply some level of automation to identify patterns in issue discussions and classify them as simple or complex. In our future work, we plan to develop such tooling, which will likely involve the automated extraction of textual content from issue comments (e.g., codes representing issue resolution activities or stages) and algorithms to derive sequences of stages and patterns. This automation will likely combine advanced machine-learning techniques with heuristic-based approaches.

References

[1] https://research:mozilla:org/, 2024.
[2] https://wiki:mozilla:org/Matrix, 2024.
[3] https://www:mozilla:org/en-US/about/forums/, 2024.