Question Answering D1

Note: *Some questions were not asked because they were already discussed by other questions and/or there was insufficient time.*

Q1. Is there an overall issue resolution process prescribed by Mozilla? If so, what is it like? To what extent do Mozilla developers follow Mozilla's issue resolution process?

D1: Okay, so… There are… pieces which are largely okay so back up half a step. So it depends a lot on what you say when you or what you mean when you say Mozilla. And I mean this because Mozilla as an organization exists in many different pieces. And the overlap across pieces varies dramatically. So for example, while Firefox is our major dominant product, there are also products that are not Firefox, like Mozilla VPN. There are products like Pocket, fake spot, now Anonym that exist in like wildly different repos that have wildly different methodologies. Some of these exist on GitHub and are managed entirely through private GitHub issues. Some of them exist on GitHub and are managed through public GitHub issues. And then you have Firefox And a couple of other components which live in Bugzilla and have follow like the Firefox development process. And so when you say, is there an overall issue process prescribed by Mozilla. No.

But I can talk to you about like the Firefox aspects. And then I can also talk to you about how Firefox's aspects, you know, are diverged from in my team. Okay, so bearing in mind that I can't speak about anything outside of the Bugzilla processes. There are a couple of pieces that are important. And so these are relatively universal. um the Issues kind of have a life cycle. Like many things, they will start out as either public or private issues, which are either classified into three different buckets. There's defects, tasks, and enhancements. Various teams take different approaches to task versus enhancement or sometimes even task versus defect. But this classification does matter in the sense that people pay different attention to what type of issue you have. So, uh. defects are tracked the most seriously by a variety of people within the teams. Across Mozilla. Things which are not defects tend to have less visibility from other parts of the company.

So by this i mean, for example, when you fix a defect one of the things that we try to identify is When was the defect introduced? Was it in a release which is still in service? Because we do maintain multiple in-service branches. And if it is a in-service branch, we have to make a decision. Do we want to take that fix and either build a new one for an older version that is in service. uplift the existing one. And take an effort and try to estimate the risk of doing so. And so this process is all run largely by a group of people We call them release management. So they are responsible for sort of keeping an eye on defects that are fixed, ones that affect in-service releases they go through and they will ask developers and be like, hey, I see this was introduced in an in-service release. Should this be uplifted? Could you request that it get uplifted? And then a developer will fill out a form that says, you know. here's my uplift request. It has a certain amount of risk. Maybe that risk is none. Maybe that risk is a lot. And then the release developers or the release management will make the decision, the final decision as to whether or not it's worth it to uplift. So they'll go through. And if the decision is made, they'll apply the patch to an in-service branch. And bring it along.

We… defects also another important classification is secure whether or not a bug is secure or not. Bugs which could have bad impacts on our users. And I mean this very broadly. tend to be opened in a secure state. So they will be, you know, if either we don't know what the impact is, but we hypothesize it could be bad. We will open these bugs in a secure state. This restricts the visibility down to a small subset of the company. From there, it will be triaged determined how severe it is, what is the impact? Is there actual user impact? Because sometimes the answer is that no, like There isn't. And then it'll be opened up. then what we will do is we'll go through and we will fix the bugs. Different teams have very different approaches for how they manage their bug backlogs. This is very much not like dictated from the top down The one thing I would do want to highlight that is kind of dictated is we we have The most important things that the higher level aspects of Firefox care about are Impacted releases and severity. Particularly for defects. Defects which are above a severity of three. we consider to be important enough that various people will pay attention to them. And severity one bugs have a lot of attention paid to them.

We attempt to try to minimize the number of severity to our severity one bugs. SEV1 bugs are usually like very bad security bug must fix as soon as possible. This will involve an out of cycle release of Firefox, also known as a chem spill release um but Sev 2s are more common. And we try to make sure that they don't last, they don't linger. And so this is monitored by various groups as like a metric that we try to drive down. is number of open severity to bugs. So, um. And sorry, open severity to defects specifically. Severity to enhancements, severity to, you know. tasks, nobody cares about. No, that's not true. They have much less visibility. uh… So, you know, to answer your question, to what extent do Mozilla developers follow this process? These aspects are followed pretty religiously, like almost everybody understands them follows them quite clearly. Everything beyond that becomes much more team level. there's a lot more aspects of it that are not as rigorous, not as well defined. And vary a lot more by team. Like, for example, how do you prioritize your work? different teams have very different perspectives on how this works. You know, there are a variety of strategies that various teams apply. etc. Sound good?

[Interviewer clarified the question]

I don't think so. Not at a high level. Teams have various teams have various approaches for how to handle this. So to give an example like And I don't know how much you make a distinction between like defects and like enhancements or like feature work, that sort of thing But so for an example, in our team, one of our responsibilities is that as the JavaScript language evolves. And gains new features, changes definitions, et cetera. We have to keep the language up to date. And so one of a frequently recurring task for us is there is a new proposal that we have to implement. And so… we don't really dictate like, ah, go forth and implement it by doing X, Y, Z. But what we do have is, you know, like I wrote I don't know, three years ago, I wrote a feature development checklist. And so this checklist is you know, hey, you are implementing a new feature. Here's a checklist of things that you ought to be doing. you should do this kind of thing. Hey, if your feature touches these sorts of things, you'll have to write these sorts of tests. You'll have to go and fix these kinds of files. You'll have to go and look at this kind of thing. You'll have to send this kind of email. You know, all of this sort of thing because it's just a lot like we have a lot of things that you have to cover and it's very easy to forget them. And when you forget them, the problems They vary in severity, but they can be problematic, right? you know, if you do it wrong, you could, for example, break add-ons. And of course, add-ons, it sucks because you don't notice that until some add-on author tries to actually use it And then they're like, hey, why doesn't this work? And it's like. oh crap, I forgot that add-ons are actually a little special. The world that they exist in is weird. And so you may have forgotten to deal with that. And so the checklist is very helpful to try to avoid that kind of problem. at a more high level goal.

We don't have a lot of great like thou must go forth and do X, Y, Z. there are cultural expectations. So for example, like. You must get code reviewed. Actually, no, that's not true. I guess we have one, which is you got to get your code reviewed. That's a pretty strong one. But after that, it's like. How much testing should you write? Well, that's going to vary a lot depending on team. You know, how much testing should you do before you land your code? We have this ability to push out test builds to what we call our try server. How big should your tribe be? Should you run like every single test or should you run only a very focused subset of it? Or should we let our like You know, machine learning model to select what tests probably should run, you know, and different teams have different approaches to this Similarly, it's like much should you be sitting down and doing performance testing before you actually land your code? Well, different teams, again, have different perspectives on this.

Q2. What workflows for solving issues do you use more frequently?

D1: So my typical approach and I would say this covers probably about 70% of my bug fixing is: take bug report, turn into test case if test case is not already provided. I want to be able to reproduce this test case. Ideally, I want to be able to reproduce this test case in our JavaScript shell rather than the full browser. Because that eliminates an entire swath of complexity. And so it's much easier and faster for us to iterate inside of our JavaScript shell. So take shell test case. Hopefully, fingers crossed, it's not architecture specific. If it's not architecture specific. If I don't know just by looking at the backtrace approximately what I'm going to have to change.

I personally, one of the very first things I do is I take a recording using RR which is the record and replay debugger generated by or that was built by Mozilla years ago. And then I submit it to a tool called Permosco. Pornosco is a… It describes itself as an omniscient debugger. So the omniscient debugger means that essentially it takes a recording of a program play and turns it into a database. And then it gives you a web interface to this database of program state. And you can debug through this. And Pernosco traces, in my experience, are anywhere between 2 and 10x faster for me to actually like find problems. It is a huge improvement in my workflow. So I try very hard whenever I have a bug. to get a Bronosco trace because it is just amazingly better.

Once I have a Pronaska trace, I go to Pronosco, I poke around, I debug. Pronosco has this little feature where you can have a notebook where you can leave notes on like points and program state. And you can say, okay, at this point, I know this. At that point, I know that. I will fill in the notebook as I'm debugging. This is very useful because if I get stuck, I'm able to send this Pernosco trace the URL, to another developer and have them look at that and they can see all my notes. And when you click on a note, it jumps you immediately to the points in program state. It's great. So I'll debug it. And then I'll typically write up you know, in my own personal notes or sometimes on the bug, I'll write up like, okay, I know what the problem is. Problem is that A, B, C, D. expect that this happens, then this happens, then this happens, but there's some ordering constraint that wasn't maintained. So now we have to make sure that that happens. And we can do that by doing this, that, and the other thing.

I'll go and I'll start developing a fix. I use… Locally, I use VS Code Remote. I have a build machine that sits in my basement. And I use VS code remote to get onto it. I personally have switched lately to using jujitsu for version control. So I will just like start working on a new jujitsu change. start writing the code. Try to get fixed once I get something that is working, I'll then go back and I will try to make that fix sensible, you know, remove debugging code, sometimes split it, layer it, you know, whatever. my goal with My personal goal with writing patches is always to try to like Make sure that every individual unit is both reviewable, buildable, and correct. Ideally, if I'm being very, very good, I would like to tell a story through the development of this I succeed with that less than I used to but And then I will… Once I've got this fixed, so long as it's not a security bug. Then I'll submit a try build So I'll push it to our continuous integration server. It will do a whole bunch of building for me. I'll go off and do other things. in anywhere from half an hour to two hours, I'll come back and I'll look. I'll check the results.

About 30% of the time I've made a dumb mistake, you know. something that is incorrect on some platform or like code that doesn't compile in opt but did compile in my debug builds, that sort of thing. So go back, rectify that. Rinse and repeat, you know, go back, run it through the tri server until I'm confident in the fix. Sometimes the try server identifies that my fix is incorrect. So that I have to go back to first principles and fix not only the bug that I had, but the new bug that like why do these two things not agree? I get this all working. I'm pretty happy with it. I submit it for review. It goes up on fabricator. I tag a particular reviewer for my code. Usually because I know the team, I know who is the best, has the best context for this bug to actually do the review.

And then I wait and I wait for them to actually provide review. Sometimes they'll provide review feedback where they're like. Cool. It's good. Land it. Other times they will be like, yeah, no, please take a different approach, you know, solve it a different way. Did you consider trying this, that sort of thing. you know, oh, look, here's a little test case that will blow up in your thing. that sort of thing happens. iteration through review And finally, once it's been approved for review. I will request landing, which goes through a tool we use called Lando. And I press a button. And then eventually it shows up on our what we call central, which is our like main branch And it will get built through our continuous integration. And in about 12 hours, it shows up in a nightly build. And so then I can test it in a nightly build. If I really want to.

Q3. In your opinion, could the identified issue resolution patterns be useful in any way for Mozilla stakeholders? If yes, how?

D1: Maybe! I think it could be interesting. Let's say the end state of this is that we automate the ability to highlight certain patterns in a bug. 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. And so, occasionally we've done previous work where we've done, for example, like retrospectives on security bugs where the idea is like every security bug you highlight, like what is the root cause? And then you can say, you know, root cause is a logic error. It's a memory safety error. And then from that, you can draw conclusions where it's like -- ah, you know, we're seeing a lot of logic errors and it's particularly in these components. Why is that? You can start to formulate ways to address that. I could imagine that this sort of thing 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.

Having said that it feels pretty obvious when you look at a bug, but like if you've got this sort of thing, like. I'm not sure how much value there is in automating this beyond just being like, maybe this is a process we should have where like when a bug has gone wrong by some you know. gut feeling, maybe we can say, hey, like, yeah, something happened here and maybe we should have changed how we developed this.

Beyond that. I think one of the things that I'm very, you know, this is one of my questions. I don't think you have the answer yet but qualitatively This feels like the normal structure of, this is kind of what I would expect from the average software project. If you went through the bug tracker at IBM, if you went through the bug tracker at, you know Red Hat or something like this, I feel like you would actually end up seeing a pretty similar pattern across all of them. And so I would be curious if there was actually something that stood out here as being like. in Mozilla much more than other comparable company. you see a lot more of patterns that look like this, you know, complex, large, recursive patterns about, you know, issue resolution. Or maybe not. Maybe, you know, ours is more linear than the average company. I don't know.

[Interviewer replied]

Q4. Could the patterns be used to train new Mozilla developers on how to solve issues? If yes, how?

D1: I think so. As I said, we have a lot of diversity about how things actually work. And one of the things that would be good is to highlight common patterns for developers. And this is just sort of one of these things about sort of psychological safety in a sense, where by sharing common patterns, being like, hey, look, like. You know, in your little graph here, you have the recurring solution, implementation, code review, and verification happening 28 times. 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. They want it to be the thing. They don't want to get review feedback where it's like, oh, you should do it differently. So, and this can really negatively affect particularly juniors because they're not willing to sort of put up a partial solution or put up a solution that maybe doesn't actually fix all the problems.

In order to have that conversation about like, okay. I built this. It works. I don't like it or like, you know, what do you think? Do you think this is something that we could land? having these kind of conversations concretely against code is something that I find juniors are not always amazing at. Because they want to be succeeding. And 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. That could be helpful, yes.

Q5. Could the patterns be used to estimate developers' efforts to solve issues? If yes, how?

D1: I don't think so. you know it's like, I guess let me let me, let me ask a little bit about definition here. Are you saying retrospectively evaluate the effort applied to a bug Or are you saying prospectively. looking at a bug or a sequence of comments and being like, this is going to take some amount more, you know, some amount more.

[Interviewer clarified what we meant by effort]

Maybe. The thing I think about it is that it's not so much about estimate in a sense. It's more about maybe highlighting common causes for why issues end up doing these sort of cycles. That could be, or even issues that should follow a certain pattern. And maybe aren't currently structured to. 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.

Similarly, if you have a bug that you expect to go through a cycle, that could be better hinted as maybe this bug's too complicated. Maybe you've tried to cover too much in one bug. 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. I emphasize the maybe on this just because I think Tooling like this is very, very hard and people are extremely opinionated about how to manage work and workflows. And so it's it kind of has to be really, really compelling in order to even get people to even try it.

I think, yeah, just to give the concrete answer, I think the thing that would be helpful is maybe highlighting where like a bug is expected to follow a certain pattern and maybe will require some aspect of this.

Q6. Could the patterns be used to solve new issues? If yes, how?

Q7. Could the patterns be used by Mozilla stakeholders to evaluate how well the issue resolution process is executed at Mozilla? If yes, how?
Q8. Can you think of other potential usages of the patterns to help improve Mozilla's issue resolution?
Q9. Do you think developers in other software systems follow a variety of workflows to resolve issues (as we found at Mozilla)?
Q10. Do you have any additional thoughts about our identified issue resolution patterns for Mozilla?
Q11. Do you think our findings improved your understanding of Mozilla's issue resolution process? If yes, how?

D1: I don't think so. This largely matches my personal feelings about how a lot of pieces fit together. I think it all seems fairly normal. You know, there's that part of me that has curiosity about like. What is the longest path through some of these and what happened in those bugs and why did that happen? And like that you know that seems interesting, but it's not particularly useful for understanding because like I've been part of some of these bugs where, yeah, you know, it takes you three times to go through and try and figure something out and then you land something and then, oh, look, the bug comes back and like. you know, it happens. So it doesn't I didn't find it changed my understanding that much.