Question Answering D2

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?

D2: Okay, so I mean, typically once uh once a developer starts working on a bug. They… First, uh. you know, there's the diagnosis stage often. A lot of the time, you know, you've got to figure out why it is that this is happening. And depending on you know how complicated the bug is you might have to bring in other stakeholders. you know, we work with things that touch web standards quite a lot And often you have to go and go to an expert in that section of the web standard and talk to them and say, hey. you know we have this bug and if I try to solve it this way.

I think that this violates the standard and we probably ought to be doing this. And very often you'll end up in situations where you identify a bug in the standard. And so you have to rope in somebody who works on those standards and figure out if the standard needs fixing. And… then that brings in even additional things where you start talking about, okay, well, the standard's broken, but what does Chrome do? What does Safari do in this situation? And if we were to change the standard. how disruptive would that be to people who have already written their website in you know, whatever this using whatever this feature is And so we constantly have in the back of our mind is there a spec fix and can we fix the spec without causing too much fallout? So… you end up pulling in people who are experts on web platform. And who understand how things are used in the field or can at least look into it. So pretty much everything if you're doing a good job. involves talking to people like that. But assuming you've done that work. How deeply do you want me to talk into uh Okay. We just want to get an overview and if you can give us one and yeah, tools is perfectly fine if you can share that with us. Okay.

So We do not have a prescribed local set of tooling. So developers kind of set up their machines really like. we get wildly different approaches to what local development tools are used. So none of that's really specified. So you work on your bug. Hopefully you write tests. And then you start trying to test against continuous integration so I don't know if you've looked at continuous integration set up, but it's It's a little overwhelming. I don't know if you've looked at Tree Herder. Have you poked at that at all. Okay, so figuring out what happened when you push some tests to that. is a bit of an art. Because there are always intermittent failures. that set of intermittent failures is constantly changing. And so you kind of have to have your finger on the pulse of what intermittent failures are considered normal at the moment. And try to make a determination, okay, have I made any of these worse? Have I accidentally fixed something? that I need to document somewhere. Or have I introduced some new intermittent failure.

So different developers exercise different degrees of carefulness on that. And there is not really a prescribed metric for how green your trip push needs to be before it's good enough. to land. Right. So that varies from developer to developer. I tend to be more on the careful side. But that uses up a lot of compute resources. These are a lot of tests run on a lot of platforms and we can burn quite a lot of uh quite a lot of machine hours doing this. I tend to be a little more conservative on that, but it does vary. So you have that aspect of testing. where there isn't really any formal rule around how good it has to be there isn't really a great easy to understand way of determining what set of intermittent failures are expected. Because indeed they are changing all the time. And if you were trying to keep that documented continuously. it would just it would be a disaster.

Now. there are some people who keep much closer track of that stuff. And I'll get to them later. They kick in when you land, when you actually push a fix. That's probably not important. It's probably a package. So then you have to go through the code review process. And the code review process is basically non-negotiable. at Mozilla. If you want to land something you have to get somebody to review it. there's basically no out. for this. very few exceptions. Hey, I've got a call going right now. Okay, so… Generally speaking, you have to get review from somebody who's considered a peer. whatever module you're working on. And so you've got a list of people that are considered the peers on a given module. If you're working in your own area, you know who these people are. And you probably have preferred ones for different areas. that you might be working on. But you have to get a reviewer. And we have a tool called fabricator. I think that was originally written at Facebook. And they open sourced it. And now we're using that tool. We've used a variety of different tools over the years. But we've been using this one for I don't know, probably six or seven years at this point. I'd have to go look back. And… So the reviewer goes through. And does code review.

We have official guidance on how you're supposed to review like what you're supposed to be looking for the guidance in the past has been does this make things worse? So you want to make sure that they aren't at least making something worse than it already was. And we're not supposed to nitpick too much about whatever approach was used. officially now officially it really does depend on the reviewer. how nitpicky the review is going to be. I have had, you know, COVID views where you know you put something up for review and you've got 50 comments on you know a fairly small thing and they are nitpicking every little you know style choice approach. you know, just the whole nine yards and then others are more, does this work? Does this make sense? And so there's a great deal of variation that you get on that as well. This is again a place where you want to pull in somebody who is familiar with the web platform. Because they may have some ideas around, oh, hey, this doesn't look quite spec compliant here. you probably need to do this instead.

But again, you're supposed to get review. Technically, you're only supposed to get one for each module you're touching. So if you're touching multiple modules at once, you got to get a reviewer that is a peer on all of them. before you can land, sometimes you want more. if you're doing something particularly dicey complicated, you might want to get an extra reviewer in there to see if they catch something that you know you're first choice or didn't. And there might be multiple rounds of this. the general pattern is Often you will get someone saying this looks good. I have some knits But I'm giving it the thumbs up, right? They might have some little things that they want fixed. before you land, but they don't really want to look at it again. After you've made those changes.

Our team is trying to shift towards not doing that. Because the tooling doesn't flag uh code reviews. that where that's been done very well, especially if you're asking a question. And haven't given a thumbs up or thumbs down yet. it does sometimes cause things to fall through the cracks because, you know, you ask a question, somehow they miss the email And then it's just kind of left in limbo. And so what we're trying to do now on our team at least is if you have any questions. that you need answered at all. you give it the thumbs down. So that it pops up.

You know, in their list of things to follow up on. And we're trying to adopt that right now, but that's not a company-wide thing. So once you've had code review. And you get you know the go ahead on everything. then you have to pay attention to where we are in the release process. So… we issue releases on about a six week kind of cadence. And… in the last week or so of that cadence, you know, the last week before we move we put out a release, we have a soft freeze And so that's where a only land really critical stuff. If it's a new feature or a minor fix or something like that, you want to let that wait. Until the release has been cut so that you get enough time on nightly. to shake out any bugs or problems with it. If all of that satisfied. you know you've got your review, try looks okay you're in a part of the release schedule where where it's appropriate. to land your patch. then you've got the go ahead. You can go ahead and land. And this is for regular bugs.

Now, when you're dealing with security bugs, there's a little bit more process. So bugs where you have like this could be if handled improperly. could get the attention of nefarious individuals And give them a hint about hey, I could exploit this before it makes it to release. So if you land a security fix in nightly and you're careless about broadcasting, hey, this is a security issue and this is the security issue that we're fixing. you can end up causing pain for users. So when you have a security bug, first of all. when you file it, it's supposed to be flagged security sensitive. And what that does is make it to where only a small set users, usually employees.

Occasionally we have volunteers who can work security bugs. not everybody can see it. First of all. And so you do your work there you have to explicitly grant access to your reviewers to be able to see the bug if they don't already have that privilege. And go through the code review. you are supposed to write your commit messages in a manner that don't paint a bullseye on the security bug that you're fixing. So what will sometimes happen is a cover bug So you'll see a developer open up a public bug and it's It's often couched as like oh clean up this code or you know, fix this corner case or something like that. But what's really going on is they're fixing the security bug that's over here in this hidden bug. And they're just kind of like doing it in a manner that doesn't immediately flag it as, oh, hey, this is a security bug being fixed. Because if you land a fix with a bug number. that when you go to try to look at it, it's access denied. that's a great big hint that, hey, a security bug is being fixed here and might not be explained in the commit log, but you still have a hint. And so when you do a cover bug. it makes it that much harder to figure out and perhaps exploit. So… when it comes time to land a security bug. Depending on the severity. the security problem in question you might need to get additional approval. from the security team. And so they kind of keep tabs on all of the security bugs, particularly the ones that are severe. And try to time landing of that in a manner to where it is pushed out, not just to nightly but to every affected release channel more or less simultaneously. So that you get as small a window as possible where it's fixed on one release branch. But not others.

And again, that depends on severity. sometimes you will have a bug that's so severe that it causes a new release build to be made. That's pretty rare. honestly doesn't happen a whole lot. to any given developer. But that is a thing that happens sometimes. So that's the basic security bug. blow. After landing. Of course, you've got the continuous integration running on, you know, head or Mozilla Central is what we call it, or Autoland is what we're using right now. Which is the tool that fabricator or the repository that the auto land tool in fabricator And then that periodically gets merged to Mozilla Central manually. So… We have a dedicated group of people They are called the sheriffs. And generally, there is one sheriff on duty at any given time. throughout the day and their job is to watch auto land or Mozilla Central, well, both. And look for any patch or landing that has seriously broken something. like causing a test to permanently fail.

We're causing build issues or causing this, that, or the other. And they are the ones who have their finger on the pulse of expected failures. Because that's what they do all day, every day. is the monitor continuous integration And look for bustage. And if they see bustage. Depending on how severe it is. they might try to reach out and say, hey, we think you've broken this test, can you fix this In short order like in the next hour, basically. If they can't reach you or it's a more involved fix than you can do and that kind of time period, you'll get backed out. So they'll take your change and pull it back out. and kick it back to you and say, hey. this broke, this broke they flag the bug that it came from. give you a need info and then you're supposed to go and figure out what happened and fix the problem. Bradley ended the gap And from there, once you've landed something on nightly. there is sometimes follow-up on the, hey, do we need to push this change out to beta early or even release early. is this crucial enough that we need to expedite it? to the more stable release branches or can it just ride the trains? And, you know, go to beta whenever nightly goes to beta. And, you know. do you need to uplift it to one of the extended support releases? And so that's kind of like the tying up loose ends like after you land the patch part of the process. So that's kind of That's kind of the flow. from start to finish.

Q2. What workflows for solving issues do you use more frequently?
Q3. In your opinion, could the identified issue resolution patterns be useful in any way for Mozilla stakeholders? If yes, how?

D2: In some of these cases, I see this this uh bar in the middle here where it's just implementation and landing. That is not supposed to happen. But it does, particularly bots will make changes to the expectations for web platform test, for example. And those just happen and get landed automatically. And it's a pain in my neck. I could definitely see it being useful to have like a percentage of you know, how much of our code work is being done by bots without code review? That would be a nice statistic to have.

[Interviewer clarified which comments we focused on]

So that would be good to know. Now, I don't know how often I mean, you say that you sampled over, you know, the last 14 years. I don't know how that percentage has changed over time. That would be interesting. it would, I mean, honestly, having longitudinal data on these would be pretty cool. But…

[Interviewer again clarified patterns distribution across years]

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. But as somebody who has authored things like that occasionally, sometimes that's how complicated the fix is. And decomposing it into little pieces ends up being very, very difficult. Particularly because oftentimes these things sort of evolve in a chaotic way on a developer's machine. And so they have this big lump of changes and then taking that and parsing it out into little logical bits. to where it makes sense at every step of the way is actually quite difficult. And part of that's just because of that diff tools being maybe a little bit limited.

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

D2: It could be useful in the sense that it shows that the canonical you know, reproduce design a solution write tests. You know go through code review make sure it works on continuous integration, you know, like the canonical simple doesn't always work out that way and that's okay. That's kind of expected. Because you know, it's good to have a plan, but you know, no plan survives first contact with the enemy so you kind of have to you have to be flexible, but the idea here is you have all of these components in there. So 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, because we have many, many test suites and figuring out which one to put a can sometimes be a little bit difficult." So, I'd be interested maybe to see like did you are all of these bugs that actually got finished. Okay. Because sometimes… Sometimes you go through this iteration and then it just, at least normal training evaporates like the interest is gone and That happens occasionally, but it sounds like that was out of scope for what you were doing. Okay.

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

D2: Are you talking about having input into estimation or are you talking about a purely retrospective kind of how hard was this?

[Interviewer clarified]

Okay, so trying to make some sort of forecast.

[Interviewer elaborated on our thinking with an example]

Yeah, I mean… Generally speaking. We don't do a lot of formal estimation, at least on my team. I mean, maybe there are other teams that are more focused on things like agile. Every team is a little bit different in terms of how they plan their work. And our team at least tends to be very much planned by whoever's working on the thing. And part of that's because on our team, in particular, we are all very seasoned. Most of us are developers who've been working for you know more than 10 years. We all tend to be fairly senior. We don't have a lot of newbies on our team. And so the old hands like to have more autonomy. And generally, they can be trusted with that level of autonomy. And so we don't do a lot of formal estimation now.

Of course, whoever's working on it is going to have an idea of -- oh this is going to be a total pain I can tell from the outset, or maybe they'll be able to look at it like, okay, yeah, this is pretty easy actually you know this is an easy fix. The test is easy to write. The way that we learned about it we're likely to notice if it regresses or if this fix actually worked pretty quickly. But in some cases, you don't have those things.

So trying to think from the perspective of from, say, product management it might be. 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. That might be useful.

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.

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

D2: So I guess that goes back to or what I was saying before you asked this question applies. Like the bots trying to like mark things as stalled that is a push towards a solution. So in that respect, data like this could be useful. As far as guiding developer efforts goes I'm not totally sure. I mean… if you have… I don't know whether you're angling at a like a machine learning application for data like this. I don't know if that's something you're thinking about at all. But um having that sort of guidance for, you know, hey, it looks like you're at this stage do you need to do like a circle back around to one of your reviewers, like just kind of a maybe even a local tool or something like this that doesn't necessarily poke the bug tracker, but just pokes the developer in some way. I guess I could see something like that. Again, it's… it's messy, right? And… anybody who's been in you know a software engineering project of this level of complexity knows that it's messy. Having process is great, but it's not always going to work and you're going to have to you know revisit things or circle back around.

Q7. Could the patterns be used by Mozilla stakeholders to evaluate how well the issue resolution process is executed at Mozilla? If yes, how?

D2: So, I mean, maybe 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. So I could see that maybe being useful.

But again, the developers know, and if their manager and product people are asking them, it's like, hey, you know if you had to pick one area that's like the tar pit in our corner of the code base like they'll tell you they know, believe me. But maybe having data could help justify additional spend, maybe? could be useful for product managers in that way. I guess.

Q8. Can you think of other potential usages of the patterns to help improve Mozilla's issue resolution?

D2: So, I mean, it's uh longitudinal data would be pretty cool. But again, that's, you know, it's cool, right? But the question is, is it actionable I am not sure whether longitudinal data or, you know, to what extent longitudinal data would be actionable. Let me think. I think it goes back to 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. Maybe. Well, that's not exactly related. But I guess you could track statistics on how long reviewers take on average, but Yeah, some people might get embarrassed. Right?

Q9. Do you think developers in other software systems follow a variety of workflows to resolve issues (as we found at Mozilla)?

D2: Oh, yeah. And particularly around code review, right? Mozilla is pretty heavy on requiring code review and other outfits aren't. Like, you know, I've worked, you know, Skype and code review was not really a thing there. So that varies tremendously. The degree to which people write tests varies tremendously.

And we've only recently added stuff to our process where you have to account for when you're it's on the reviewer, actually. So when the code reviewer is doing as they're supposed to mark the thing as this is been tested. And if it isn't, they've got to explain that. They've got to explain why in this case we don't want or can't make a test and record that. That is not something that we used to have, but we have this testing flag thing that we do nowadays. And that's encouraged more people to write tests, which is great.

But again, multiple, you know, like I know that not every software outfit writes tests. They just kind of say, eh, I think the solution is this. And then they might run it across continuous integration and then just land it and be like. It's probably fine. And the number of times where I've worked on something and said, I think that should fix it. I'm gonna test it now and found No, in fact, that did not address the issue. shows me that, you know, the testing is really important. And… Every single one of these things varies by by company and by team. I could tell stories about any of this stuff, really. But yes, it all varies tremendously and the degree to which there's a process varies tremendously.

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?

D2: I mean, it's kind of cool to see how often it's complicated and how often it's simple. That's actually pretty neat. It's… it makes me want to make me want to have more data, right? want to see, hey, how does dom do on this versus the networking people versus, you know. the poor souls who work on css. That would be kind of neat. But…

[Interviewer asked if there is a way to identify the teams]

So… I mean… publicly available data recording who works on what team I think you just have to glean it by looking at what component this person tends to work on. They you know we've got our internal you know, people directory that records, hey, this person's on thus and such a team. But sometimes it's even that's messy, like particularly with uh more senior engineers they tend to pick up things in multiple places. And it's kind of hard to tell what team they might be on. And, you know, what exactly their field of expertise is. I think you'd have to go by component, Okay. So, you know, we've got our component in the Bugzilla tracker. That would be how I'd try to do it if I were to do something like that. You could… If you wanted to get more involved. You could look at the change sets and see what files they were touching. Right. But that would be a lot more involved and a lot more messy. And you end up probably duplicating the thought process that every developer goes through who's like, what component do I put this in, because it’s not clear. So that would be duplicative effort. But I mean, it might also be interesting to see Hey, like… maybe these things need to be over in this other component.