14:04:49 Yeah, let me turn on the recording. Yep.
14:04:54 And to get started with the interview, I just want to share quickly about the interview plan.
14:05:00 So we structured the interview into four sections. At first, we want to know about your
14:05:07 background and work experience in software engineering.
14:05:11 Then in the second section, we will be learning from you about the Mozilla's issue resolution process.
14:05:18 In the third section, we will give a short presentation on our research findings
14:05:24 about Mozilla's issue resolution process.
14:05:26 And in the final section.
14:05:28 we will be asking a few questions about our findings to understand the usefulness of our findings for Mozilla stakeholders.
14:05:37 All right. So I
14:05:38 plangun.
14:05:39 Gosh, I will go ahead and get started.
14:05:43 I finished my degree in computer science and mathematics at utd
14:05:50 In spring of 2005.
14:05:53 And right after that, I joined a
14:05:57 pretty small startup that was local to the Dallas area.
14:06:01 they worked in
14:06:03 IP telephony, basically. So, uh.
14:06:06 the SIP protocol, if you have any familiar with that.
14:06:12 I worked there
14:06:14 for about six years.
14:06:18 And…
14:06:21 And then…
14:06:22 moved out to the Bay Area to take a job at Skype.
14:06:27 And worked there for about a year and a half.
14:06:29 And then in 2013, I took a job at Mozilla and I've been there since.
14:06:36 And, uh.
14:06:37 you know, throughout this whole time, I have been just
14:06:41 basically a developer.
14:06:46 In some cases, I have taken tech leadership type uh
14:06:50 responsibilities on, but never anything like
14:06:54 managerial
14:06:57 So.
14:06:59 It's been technical from the very beginning.
14:07:03 And I've spent my whole career working in
14:07:07 real-time media in one form or another.
14:07:11 And now I work on
14:07:13 Firefox's WebRTC implementation.
14:07:18 Thank you very much for sharing. So can I ask, do you have any issue resolution experience

14:07:26 at Mozilla.
14:07:29 Well, yes. I mean.
14:07:31 We have our process and uh
14:07:34 I'm very familiar with it at this point. It has changed over the years a little bit.
14:07:41 I haven't really been involved in
14:07:45 making policy around
14:07:48 issue resolution.
14:07:50 I've got my own things I want to see when I do code reviews, but
14:07:56 nothing like that's been formalized company-wide.
14:08:03 Yeah.
14:08:03 Sounds good. If I can follow up a little bit.
14:08:08 Are there any other products or components that you have worked on at Mozilla?
14:08:13 I mean, I've worked on…
14:08:18 some other pieces occasionally.
14:08:23 I've done a fair bit of work on the timer infrastructure
14:08:28 down in our
14:08:30 you know our engine.
14:08:33 But…
14:08:35 overwhelmingly, my work has been done in
14:08:39 the real time space in WebRTC in particular.
14:08:43 Sounds good. We can move on to the next section of the interview which is
14:08:49 And you mentioned a little bit about this. So we want to know about if there is a prescribed process by Mozilla, about how to solve issues, how to go through that process.
14:08:59 And let me clarify here what we mean by issue resolution. So we are interested in
14:09:06 Essentially, the process that developers actually go through
14:09:10 for actually solving the issues. So that's basically after triage, right?
14:09:15 Okay.
14:09:15 triage. We don't necessarily, we don't focus on three ash pretty much we want to
14:09:21 Okay.
14:09:22 After his triage, then what the developers do
14:09:26 to actually solve the issue. So is there any prescribed process and to what extent modular developers follow this process?
14:09:32 Okay, so
14:09:34 when you say triage, that could mean a couple of things.
14:09:38 Are you talking additionally
14:09:41 In terms of
14:09:44 Right.
14:09:45 assigning work out.
14:09:47 Right, right. So, yeah.
14:09:47 to developers. Triage is prioritization, but you have the additional step of

14:09:52 saying, okay, based on this information.
14:09:55 go work on this. Are you wanting to talk about that?
14:10:00 it's yeah after the issue is assigned. So there is a person who actually go through or a team actually go through the actual
14:10:08 resolution.
14:10:09 Okay, so I mean, typically once uh
14:10:13 once a developer starts working on a bug.
14:10:19 They…
14:10:21 First, uh.
14:10:23 you know, there's the diagnosis stage often.
14:10:27 A lot of the time, you know, you've got to figure out why it is that this is happening.
14:10:33 And depending on
14:10:34 you know how complicated the bug is
14:10:37 you might have to bring in other stakeholders.
14:10:40 you know, we work with things that touch web standards quite a lot
14:10:46 And often you have to go and go to an expert
14:10:50 in that section of the web standard and talk to them and say, hey.
14:10:54 you know we have this bug and
14:10:58 if I try to solve it this way.
14:11:01 I think that this violates the standard and we probably ought to be doing this.
14:11:06 And very often you'll end up in situations where you identify a bug in the standard.
14:11:12 And so you have to rope in somebody who
14:11:15 works on those standards and figure out if the standard needs fixing.
14:11:21 And…
14:11:22 then that brings in even additional things where you start talking about, okay, well, the standard's broken, but
14:11:31 what does Chrome do?
14:11:33 What does Safari do in this situation?
14:11:36 And if we were to change the standard.
14:11:39 how disruptive would that be
14:11:41 to people who have already written their website in
14:11:44 you know, whatever this using whatever this feature is
14:11:48 And so we constantly have in the back of our mind
14:11:53 is there a spec fix and can we fix the spec without causing too much fallout?
14:11:59 So…
14:12:00 you end up pulling in people who are experts on web platform.
14:12:05 And who understand how things are used in the field or can at least look into it.
14:12:12 So pretty much everything
14:12:15 if you're doing a good job.
14:12:19 involves talking to people like that.
14:12:22 But assuming you've done that work.

14:12:28 How deeply do you want me to talk into uh
14:12:37 Okay.
14:12:35 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.
14:12:42 Okay. So
14:12:45 We do not have a prescribed
14:12:52 local set of tooling.
14:12:54 So developers kind of set up their machines
14:12:57 really like.
14:13:00 we get wildly different approaches to what local development tools are used.
14:13:07 So none of that's really specified.
14:13:14 So you work on your bug. Hopefully you write tests.
14:13:19 And then you
14:13:22 start trying to test against continuous integration so
14:13:26 I don't know if you've looked at
14:13:29 continuous integration set up, but it's
14:13:31 It's a little overwhelming.
14:13:35 I don't know if you've looked at Tree Herder. Have you
14:13:38 poked at that at all.
14:13:41 Okay, so figuring out
14:13:45 what happened when you push some tests to that.
14:13:49 is a bit of an art.
14:13:52 Because there are always intermittent failures.
14:13:55 that set of intermittent failures is constantly changing.
14:13:59 And so you kind of have to have your finger on the pulse of
14:14:04 what intermittent failures are considered normal at the moment.
14:14:09 And try to make a determination, okay, have I made any of these worse?
14:14:15 Have I accidentally fixed something?
14:14:17 that I need to document somewhere.
14:14:20 Or have I introduced some new
14:14:23 intermittent failure. So
14:14:26 different developers
14:14:31 exercise different degrees of carefulness on that.
14:14:36 And there is not really a prescribed metric for
14:14:42 how green your trip push needs to be
14:14:46 before it's good enough.
14:14:49 to land.
14:14:51 Right.
14:14:53 So that varies from developer to developer. I tend to be more on the careful side.
14:14:59 But that uses up a lot of
14:15:01 compute resources. These are a lot of tests run on a lot of platforms and we can burn quite a lot of uh
14:15:09 quite a lot of machine hours doing this.

14:15:13 I tend to be a little more conservative on that, but it does vary.
14:15:18 So you have that aspect of testing.
14:15:22 where
14:15:23 there isn't really
14:15:25 any formal rule around how good it has to be
14:15:32 there isn't really a great
14:15:36 easy to understand way of determining
14:15:40 what set of intermittent failures are expected.
14:15:43 Because indeed they are changing all the time.
14:15:45 And if you were trying to keep that documented continuously.
14:15:49 it would just
14:15:51 it would be a disaster. Now.
14:15:55 there are some people who
14:15:58 keep much closer track of that stuff. And I'll get to them later. They kick in
14:16:03 when you land, when you actually push a fix.
14:16:11 That's probably not important.
14:16:12 It's probably a package.
14:16:17 So then you have to go through the code review process.
14:16:21 And the code review process is basically non-negotiable.
14:16:25 at Mozilla. If you want to land something
14:16:28 you have to get somebody to review it.
14:16:31 there's basically no out.
14:16:34 for this.
14:16:37 very few exceptions.
14:16:40 Hey, I've got a call going right now.
14:16:46 Okay, so…
14:16:49 Generally speaking, you have to get review from somebody who's considered a peer.
14:16:55 whatever module you're working on.
14:16:57 And so you've got a list of people that are considered the peers on
14:17:01 a given module.
14:17:03 If you're working in your own area, you know who these people are.
14:17:08 And you probably have preferred ones for different areas.
14:17:11 that you might be working on.
14:17:15 But you have to get a reviewer.
14:17:18 And we have a tool called
14:17:20 fabricator. I think that was originally written at Facebook.
14:17:25 And they open sourced it. And now we're using that tool.
14:17:30 We've used a variety of different tools over the years.
14:17:34 But we've been using this one for
14:17:39 I don't know, probably six or seven years at this point. I'd have to go look back.
14:17:46 And…
14:17:48 So the reviewer goes through.
14:17:52 And does code review.

14:17:54 We have official guidance on
14:17:57 how you're supposed to review
14:18:00 like what you're supposed to be looking for
14:18:03 the guidance in the past has been
14:18:07 does this make things worse?
14:18:11 So you want to make sure that they aren't at least
14:18:15 making something worse than it already was.
14:18:19 And we're not supposed to nitpick too much about
14:18:23 whatever approach was used.
14:18:27 officially now officially
14:18:29 it really does depend on the reviewer.
14:18:32 how nitpicky the review is going to be.
14:18:36 I have had, you know, COVID views where
14:18:39 you know you put something up for review and you've got 50 comments on
14:18:45 you know a fairly small thing and they are nitpicking every little
14:18:50 you know style choice
14:18:53 approach.
14:18:54 you know, just the whole nine yards and then others are more, does this work?
14:19:00 Does this make sense?
14:19:02 And so there's a great deal of variation that you get
14:19:05 on that as well.
14:19:08 This is again a place where
14:19:13 you want to pull in somebody who
14:19:15 is familiar with the web platform.
14:19:18 Because they may have some ideas around, oh, hey, this doesn't look quite spec compliant here.
14:19:24 you probably need to do
14:19:26 this instead.
14:19:28 But again, you're supposed to get review.
14:19:33 Technically, you're only supposed to get one for each module you're touching.
14:19:37 So if you're touching multiple modules at once, you got to get a reviewer that is a peer on all of them.
14:19:44 before you can land, sometimes you want more.
14:19:48 if you're doing something particularly dicey
14:19:52 complicated, you might want to get
14:19:54 an extra reviewer in there to see if they catch something that you know you're
14:19:59 first choice or didn't.
14:20:03 And there might be multiple rounds of this.
14:20:09 the
14:20:12 general pattern is
14:20:15 Often you will get someone
14:20:18 saying this looks good. I have some knits
14:20:20 But I'm giving it the thumbs up, right? They might have some little things that they want fixed.

14:20:27 before you land, but they don't really want to look at it again.
14:20:30 After you've made those changes.
14:20:33 Our team is trying to shift towards not doing that.
14:20:38 Because the tooling
14:20:41 doesn't
14:20:44 flag uh
14:20:45 code reviews.
14:20:47 that where that's been done
14:20:50 very well, especially if you're asking a question.
14:20:53 And haven't given a thumbs up or thumbs down yet.
14:20:57 it does sometimes cause things to fall through the cracks because, you know, you ask a question, somehow they miss the email
14:21:04 And then it's just kind of left in limbo.
14:21:07 And so what we're trying to do now on our team at least is if you have any questions.
14:21:12 that you need answered at all.
14:21:16 you give it the thumbs down.
14:21:18 So that it pops up.
14:21:19 You know, in their list of things to follow up on.
14:21:23 And we're trying to adopt that right now, but that's not a company-wide thing.
14:21:29 So once you've had code review.
14:21:33 And you get you know
14:21:35 the go ahead on everything.
14:21:39 then you have to pay attention to where we are in the release process.
14:21:45 So…
14:21:47 we issue releases on about a
14:21:50 six week kind of cadence.
14:21:54 And…
14:21:56 in the last week or so of that
14:22:00 cadence, you know, the last week before we move we
14:22:04 put out a release, we have a soft freeze
14:22:07 And so that's where a
14:22:10 only land really critical stuff.
14:22:12 If it's a new feature or a minor fix or something like that, you want to let that wait.
14:22:19 Until the release has been cut so that you get enough time on nightly.
14:22:25 to shake out any bugs or problems with it.
14:22:30 If all of that satisfied.
14:22:32 you know you've got your review, try looks okay
14:22:37 you're in a part of the release schedule where
14:22:42 where it's appropriate.
14:22:44 to land your patch.
14:22:47 then you've got the go ahead. You can go ahead and land.
14:22:50 And this is for

14:22:52 regular bugs.
14:22:56 Now, when you're dealing with security bugs, there's a little bit more process.
14:23:00 So bugs where you have like
14:23:03 this could be
14:23:06 if handled improperly.
14:23:10 could get the attention of
14:23:13 nefarious individuals
14:23:16 And give them a hint about
14:23:19 hey, I could exploit this before it makes it to release.
14:23:24 So if you land a security fix in nightly and you're careless about
14:23:29 broadcasting, hey, this is a security issue and this is the security issue that we're fixing.
14:23:35 you can end up causing
14:23:38 pain for users.
14:23:41 So when you have a security bug, first of all.
14:23:44 when you file it, it's supposed to be flagged security sensitive.
14:23:48 And what that does is make it to where
14:23:51 only a small set
14:23:53 users, usually employees. Occasionally we have volunteers who
14:23:58 can work security bugs.
14:24:01 not everybody can see it.
14:24:03 First of all. And so you do your work there
14:24:08 you have to explicitly grant access to your reviewers
14:24:13 to be able to see the bug if they don't
14:24:16 already have that privilege.
14:24:18 And go through the code review.
14:24:21 you are supposed to
14:24:26 write your commit messages in a manner
14:24:29 that don't paint a bullseye on
14:24:32 the security bug that you're fixing.
14:24:35 So what will sometimes happen is a cover bug
14:24:40 So you'll see a developer open up
14:24:43 a public bug and it's
14:24:46 It's often couched as like oh clean up
14:24:50 this code or
14:24:53 you know, fix this corner case or something like that. But what's really going on
14:25:00 is they're fixing the security bug that's over here in this hidden bug.
14:25:04 And they're just kind of like doing it in a manner that doesn't
14:25:09 immediately flag it as, oh, hey, this is a security bug being fixed.
14:25:14 Because if you land a fix with a bug number.
14:25:17 that when you go to try to look at it, it's access denied.
14:25:23 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
14:25:30 you still have a hint. And so when you do a cover bug.

14:25:33 it makes it that much harder to figure out and perhaps exploit.
14:25:40 So…
14:25:42 when it comes time to land a security bug.
14:25:47 Depending on the severity.
14:25:50 the security problem in question
14:25:53 you might need to get additional approval.
14:25:58 from the security team.
14:26:01 And so they kind of keep tabs on all of the security bugs, particularly the ones
14:26:06 that are severe.
14:26:09 And try to
14:26:13 time landing
14:26:14 of that in a manner
14:26:16 to where it is pushed out, not just to nightly
14:26:19 but to every affected release channel
14:26:22 more or less simultaneously.
14:26:26 So that you get
14:26:28 as small a window as possible where it's fixed on one release branch.
14:26:33 But not others.
14:26:36 And again, that depends on severity.
14:26:43 sometimes
14:26:46 you will have a bug that's so severe that it
14:26:49 causes a new release build
14:26:52 to be made. That's pretty rare.
14:26:55 honestly doesn't happen a whole lot.
14:26:58 to any given developer.
14:27:02 But that is a thing that happens sometimes.
14:27:05 D2, let me interrupt you for a second.
14:27:09 Do you have more time than to you know share your thoughts?
14:27:14 I have a fair bit of time.
14:27:15 Yeah. Okay. Just checking the time because we don't want to just extend too much either.
14:27:20 Right, right.
14:27:23 So that's the basic security bug.
14:27:28 blow. After landing.
14:27:31 Of course, you've got the continuous integration
14:27:34 running on, you know, head
14:27:37 or Mozilla Central is what we call it, or Autoland is what we're using right now.
14:27:42 Which is the tool that
14:27:45 fabricator or the repository that the auto land tool in fabricator
14:27:51 And then that periodically gets merged to Mozilla Central manually.
14:27:59 So…
14:28:01 We have a dedicated group of people
14:28:05 They are called the sheriffs.
14:28:08 And generally, there is one sheriff on duty at any given time.

14:28:12 throughout the day and their job is to watch
14:28:15 auto land or Mozilla Central, well, both.
14:28:20 And look for
14:28:22 any patch or landing that has seriously broken something.
14:28:27 like causing a test to permanently fail.
14:28:30 We're causing build issues or causing this, that, or the other.
14:28:35 And they are the ones who have their finger on the pulse of expected failures.
14:28:41 Because that's what they do all day, every day.
14:28:44 is the monitor continuous integration
14:28:48 And look for bustage.
14:28:51 And if they see bustage.
14:28:54 Depending on how severe it is.
14:28:58 they might try to reach out and say, hey, we think you've broken
14:29:03 this test, can you fix this
14:29:06 In short order like
14:29:08 in the next hour, basically.
14:29:12 If they can't reach you or it's a more involved fix than you can do and that
14:29:18 kind of time period, you'll get backed out. So they'll take your change and pull it back out.
14:29:23 and kick it back to you and say, hey.
14:29:27 this broke, this broke
14:29:29 they flag the bug that it came from.
14:29:32 give you a need info and then you're supposed to go and figure out what happened and fix the problem.
14:29:38 Bradley ended the gap
14:29:41 And from there, once you've landed something on nightly.
14:29:46 there is sometimes follow-up on the, hey, do we need to
14:29:51 push this change out to beta early
14:29:54 or even release early.
14:29:56 is this crucial enough that we need to expedite it?
14:30:00 to the more stable release branches or can it just ride the trains?
14:30:05 And, you know, go to beta whenever
14:30:07 nightly goes to beta.
14:30:10 And, you know.
14:30:11 do you need to uplift it to one of the extended support releases?
14:30:15 And so that's kind of like the
14:30:17 tying up loose ends like after you land the patch
14:30:21 part of the process. So that's kind of
14:30:26 That's kind of the flow.
14:30:28 from start to finish.
14:30:29 Great, great.
14:30:32 Thank you for sharing. And well, we have two more sections to go through. So let's move on to the next section.
14:30:38 And Anto is going to go through it.

14:30:41 Yes. So in the third section, we will be presenting
14:30:46 our research to you so that you understand what we understand
14:30:50 are thinking about Mozilla's issue resolution process. So let me get started with the goal. So our goal is to understand
14:30:56 Mm-hmm.
14:30:57 how developers resolve issues in Mozilla Firefox in practice.
14:31:02 To be specific, I mean, we want to investigate these stages of issue resolution
14:31:07 And we want to identify the common ways of
14:31:12 implementing the stages.
14:31:14 from reproduction to verification.
14:31:17 Mm-hmm.
14:31:17 So we want to identify the patterns, common ways of resolving the issues.
14:31:22 And our final goal is today some actionable guidelines.
14:31:26 so that we can assist Mozilla stakeholders in resolving issues
14:31:32 more effectively. So this is our final goal.
14:31:34 Mm-hmm.
14:31:37 So the question is how we can do that or how we can understand the process.
14:31:43 So our idea is to investigate developers discussion in the issue reports
14:31:48 and identify the resolution activities that is recorded in the issue report.
14:31:53 and then using the activities we can
14:31:58 understand the process of the resolution.
14:32:01 So let me give you some more details about the process.
14:32:06 so we
14:32:08 collected 356 issue reports from two modular products and they are core and firefox
14:32:14 And we qualitatively analyze the developers comments
14:32:18 And we identified the resolution activities in the issue report comments. And using the activities, we mapped the activities into the six issue resolution stages that I shared before
14:32:31 Mm-hmm.
14:32:30 For example, reproduction analysis, solution design and so on. And then by analyzing the
14:32:38 resolution stages, we identified the patterns or common noise of issue resolution.
14:32:44 So let me quickly share one real example to demonstrate the process.
14:32:50 Mm-hmm.
14:32:50 So here we present one-ish report. I'm sorry, because the figure is
14:32:55 visible enough. But I hope you understand this is a this is an issue report right
14:33:05 Mm-hmm.
14:33:00 So we investigated all the developers comments here and we identified resolution related activities for example
14:33:09 reproduction attempt or identifying the problem cause
14:33:12 or designing potential solution or implementing the code
14:33:17 and so on so

14:33:19 And using the activities, we map the activities to the
14:33:24 stages of issue resolution for example in this case we have four stages
14:33:30 that are like reproduction
14:33:33 analysis and solution design and implementation.
14:33:35 So that means for resolving this issue report.
14:33:39 Developers are found these four stages.
14:33:42 right so and
14:33:45 using these stages, we can build a stage sequence like the way how developers resolve the issue
14:33:51 So that means for this issue report, we can present the process
14:33:56 with the string and we represented it by this string
14:34:02 And here we can see there are four stages.
14:34:05 So in the same way, we identified the state sequences for all the 356 issue reports.
14:34:12 And then we grouped the stress sequences based on their similarity.
14:34:16 In this case, we identified five more issued reports
14:34:20 which have similar state sequences to this. And by analyzing this
14:34:27 sequences we derived a derived
14:34:30 hormone pattern to represent all the sequences so in this case this is the pattern
14:34:35 But we do not have to understand this because we can understand the pattern by this figure.
14:34:42 So what this figure means this figure means like for resolving these six issues here
14:34:48 developers and for us to reproduce the issue then they analyze the issue they design the solution, they implement the issue
14:34:54 And after the implementation, they optionally, apart from code reviews or verification.
14:35:01 to verify the implementation is correct or not.
14:35:03 So this is about the pattern
14:35:06 Let me share some more example to understand the patterns better.
14:35:11 So for example, this pattern for this pattern we can see there can be a repetitive cycle
14:35:18 So we have only three stages here, developers at first, apart from implementation, then code driven and verification and then they come back
14:35:27 with the same process again and again.
14:35:30 So this is about this process and we found this process in 28 issues.
14:35:36 And in the second example, we can see a similar
14:35:40 repetitive cycle. However, we have one extra status that is
14:35:45 solution design at the beginning that means at first developers
14:35:49 Apart from the solution design and then they apart from this portion
14:35:54 And I should just cut in right quick.
14:35:56 Yep.
14:35:57 often that solution design

14:35:59 is carried out somewhere else
14:36:03 So that is going to hide some information
14:36:08 it's sometimes it's done over like our internal
14:36:13 matrix instance, so basically chat maybe
14:36:16 get a video call going. So yeah, I just wanted to make sure you were aware of that.
14:36:22 Yeah, we completely agree with that and we have a question for you to discuss this situation actually yeah
14:36:29 So yeah, we know that some we can be missing some stages in the issue report comments.
14:36:36 And in the third example this is very similar to the previous one but here you can see like the solution design is a part of the repetitive cycle that means here developers can power from solution design multiple times
14:36:49 Mm-hmm.
14:36:49 So in this way
14:36:52 we identified 47 patterns among 356 issue reports.
14:36:57 And among these 47 patterns, we identified 27 simple, which means there is no repetitive stages.
14:37:06 They're very straightforward or linear and we observed
14:37:10 70% of the issues are resolved using this type of patterns.
14:37:14 On the other hand, we identified 20 patterns as complex patterns and we found
14:37:20 30% of the issues are resolved using complex patterns.
14:37:24 So the conclusion here is
14:37:26 maybe most of the issues are resolved in a simple way.
14:37:32 So we share here the top
14:37:36 drill patterns that we identified
14:37:43 Okay.
14:37:39 So we do not have to really understand the notation and everything, but we just need to understand that like
14:37:46 from this figure, you can see some parents are more frequent. For example, the first one we found in
14:37:53 64-issue reports. However, the last one we found in only seven issue reports and uh
14:38:00 We have other patterns which are found in only one or two issue reports. That means some patterns are more frequently found. So among the 47
14:38:10 patterns, we found 18 patterns in 80% of the issue reports.
14:38:15 So these 18 patterns are more frequent.
14:38:18 And we can see the issue resolution process for mosey lab deviates from the linear process that we shared before.
14:38:25 the linear sequential process
14:38:28 And we also investigated the patterns across issue types and problem categories.
14:38:34 And we found the diversity of the patterns throughout the Mozilla Firefox, you know.

14:38:40 evolution, 13 year, 14 years of evolution that means 2010 to 2023
14:38:47 And we found complex patterns are more frequent in
14:38:51 some problem categories, for example, code design or defective functionality or ui issues
14:38:57 And we also found that diverse resolution patterns are used for
14:39:02 defective functionality, code design, or UI issue that means
14:39:07 maybe this type of problems are comparatively difficult to resolve.
14:39:12 So we also have other findings based on our analysis, but we just highlighted here
14:39:18 the main findings.
14:39:20 Yeah, so please feel free to ask if you have any question.
14:39:24 No real questions. I mean, this all tracks
14:39:29 it.
14:39:31 Oh.
14:39:31 you know, the defects are hard.
14:39:35 That's mostly…
14:39:37 I don't really work on UI, so I can't really speak to
14:39:41 to that aspect of it. But I imagine
14:39:45 that's going to involve user experience, people being pulled in. And so you've got more
14:39:52 more people working on this thing and giving their input. So you're going to iterate on it more on average.
14:40:00 But yeah, this all tracks.
14:40:02 Thank you. Yep.
14:40:04 Interviewer-1, do you want to add something?
14:40:07 No, I mean…
14:40:08 So…
14:40:09 Go ahead, D2.
14:40:10 Oh, no, I'm sorry, sorry.
14:40:13 Okay, then I think we can move to our last session so
14:40:17 here we will be asking you a set of questions
14:40:20 to understand if we can use these findings somehow for Mozilla stakeholders.
14:40:28 So let me start with our first question.
14:40:31 So in your opinion.
14:40:33 could the identified issue resolution patterns be useful in any way for modular stakeholders?
14:40:38 So here for stakeholders, we mean any involved person in the issue resolution process, for example, developers, project managers or
14:40:48 other involved people.
14:40:54 So…

14:40:57 In some of these cases, I see this this uh
14:41:02 bar in the middle here where it's just
14:41:04 implementation and landing.

14:41:09 That is not supposed to happen.
14:41:11 But it does, particularly bots
14:41:15 will um
14:41:17 will make changes.
14:41:20 to the expectations for web platform test, for example.
14:41:25 And those just happen.
14:41:28 and get landed automatically.
14:41:30 And it's a pain in my neck.
14:41:35 I could definitely see it being useful
14:41:38 to have like a percentage of
14:41:40 you know, how much of our code work is being done by bots without code review?
14:41:48 That would be a nice statistic to have.

14:41:51 If I can interject a little bit. So by the way, we focused on comments actually written by humans, not by bots.
14:41:59 So in these cases, implementation was something that people wrote.
14:42:04 I was like, come in.

14:42:05 So that would be good to know. Now, I don't know how often
14:42:10 I mean, you say that you sampled
14:42:14 over, you know, the last 14 years.
14:42:18 I don't know how that
14:42:21 percentage has changed over time.
14:42:23 That would be interesting.
14:42:27 it would, I mean, honestly, having longitudinal
14:42:31 longitudinal data
14:42:34 on these would be pretty cool.
14:42:38 But…

14:42:38 Yeah, just to clarify as well, so we have the analysis over the years of these patterns, right? And
14:42:46 Yeah, we observe like some of the patterns are more
14:42:49 common throughout the years. Some of them are more common in the past, some of them are common recently and things like that.
14:42:55 This particular one, implementation only
14:42:57 Well, just to give you an idea, these actually are issues where developers try to fix the bugs, right?
14:43:06 Mm-hmm.
14:43:05 But then they realized at some point during the implementation that it was already fixed by other issues, for example.

14:43:11 Yeah, that happens.
14:43:17 So, yeah, I mean…

14:43:21 Just thinking about this.
14:43:36 it might be interesting to see
14:43:42 which modules
14:43:45 tend to have
14:43:47 more complicated life cycle for bug fixes.
14:43:54 it might be useful to have
14:43:56 metrics on this in order to
14:44:00 kind of identify
14:44:02 areas whereas
14:44:06 we need to be making an effort
14:44:08 to take smaller bugs
14:44:11 and break bugs up into smaller pieces.
14:44:14 Instead of having a
14:44:16 Gigantic bug that, you know, spills out into a 40 change set
14:44:23 Sequence of patches.
14:44:26 But…
14:44:29 as somebody who has authored things like that occasionally, sometimes there's uh there's
14:44:36 That's how complicated the fix is.
14:44:39 and decomposing it into little pieces ends up being very, very difficult.
14:44:46 Particularly because
14:44:48 oftentimes these things sort of
14:44:54 evolve in a chaotic way on a developer's machine. And so they have this big lump of
14:45:00 changes and then taking that and parsing it out
14:45:04 into little logical bits.
14:45:06 to where it makes sense at every step of the way
14:45:11 is actually quite difficult.
14:45:14 And part of that's just because of that
14:45:18 diff tools being
14:45:21 maybe a little bit limited.
14:45:23 So.

14:45:25 Okay, great. We can go to the next question. So could the patterns be used to train new Mozilla developers on how to solve issues? If yes, how?

14:45:40 I mean…
14:45:41 it could be useful in the sense that
14:45:46 it shows that the canonical
14:45:50 you know, reproduce
14:45:52 design a solution
14:45:55 write tests
14:45:58 you know go through code review

14:46:01 make sure it works on continuous integration, you know, like the canonical simple doesn't always work out that way and that's okay.
14:46:10 that's kind of expected.
14:46:13 Because…
14:46:14 you know, it's good to have a plan, but
14:46:17 you know, no plan survives first contact with the enemy so
14:46:22 you kind of have to you have to
14:46:26 be flexible, but the idea here is you have all of these components
14:46:31 in there so
14:46:43 most of the time
14:46:45 with a new developer.
14:46:48 They do spend a lot of time
14:46:51 looking at
14:46:54 old bugs or you know
14:46:56 maybe recent bugs that have already been closed in their module.
14:47:00 to kind of get an idea of
14:47:04 what?
14:47:05 what the general expectation is around
14:47:09 how done things need to be.
14:47:11 before you put them up for code review.
14:47:14 how much feedback you're expected to solicit
14:47:18 from your teammates on whatever solution you have in mind.
14:47:24 what kind of testing
14:47:26 is expected.
14:47:28 And where you would be writing a particular type of testing, because we have many, many test suites and
14:47:34 figuring out which one to put a
14:47:37 can sometimes be a little bit difficult.
14:47:43 So…
14:47:47 I'd be interested…
14:47:51 maybe to see like
14:47:57 did you are all of these bugs
14:48:01 that actually got finished.
14:48:03 Yes, all of them are fixed and resolved, yes.
14:48:05 Okay. Because sometimes…
14:48:08 Sometimes you go through this
14:48:11 iteration and then it just
~~14:48:14 At least normal training.~~
14:48:14 ~~evaporates~~ like the interest is gone and
14:48:18 That happens occasionally, but it sounds like that was out of scope for what you were doing.
14:48:29 Okay.

14:48:31 Yeah, I think we can move to next question so

14:48:35 Our next question is uh
14:48:38 could the developers be used sorry could the patterns be used to estimate developer C4?
14:48:45 to solve issues if yes how

14:48:49 Are you talking about
14:48:54 having input into estimation
14:48:58 Or are you talking about a purely retrospective
14:49:03 kind of how hard was this?

14:49:08 Yeah, more into how hard this issue could be. I mean, how hard it could be to solve yes
14:49:15 Okay, so trying to make some sort of forecast.
14:49:18 Yes.
14:49:19 Excellent. Yeah.
14:49:20 And this is what we're thinking. I mean, let me explain a little bit. So imagine that we have a new issue coming from a user or the developer, right? And then that issue is kind of similar to an existing issue.
14:49:32 And that issue has certain pattern. Maybe it's complex, it's simple, right?
14:49:36 Maybe with that information.
14:49:38 we could somehow, if we have a tool or something, right, maybe we can suggest that this new issue maybe may take a little bit
14:49:46 you know some effort or may take a little bit more time to solve right

14:49:52 Yeah, I mean…
14:49:54 Generally speaking.
14:49:58 We don't do a lot of
14:50:01 formal estimation, at least on my team. I mean, maybe there are other teams that are more
14:50:09 more focused on things like agile
14:50:14 every team is a little bit different in terms of how they plan their work
14:50:20 And our team at least
14:50:23 tends to be
14:50:26 very much planned by whoever's working on the thing.
14:50:31 And part of that's because
14:50:34 on our team in particular.
14:50:38 we are all
14:50:40 very seasoned.
14:50:42 Most of us are
14:50:44 developers who've been working for
14:50:46 you know more than 10 years.
14:50:50 We all tend to be
14:50:52 fairly senior. We don't have a lot of newbies.
14:50:56 on our team and so on.

14:51:00 the old hands like to have more autonomy.
14:51:03 And generally, they can be trusted with that level of autonomy.
14:51:08 And so we don't do a lot of
14:51:11 formal estimation now of course
14:51:13 whoever's working on it is going to have an idea of
14:51:17 oh this is going to be a total pain i can tell.
14:51:21 from the outset, or maybe they'll be able to look at it like, okay, yeah, this is pretty easy actually
14:51:26 you know this is an easy fix. The test is easy to write.
14:51:33 the way that we learned about it
14:51:37 we're likely to notice
14:51:40 If it regresses or if this fix actually worked.
14:51:44 pretty quickly.
14:51:46 But in some cases, you don't have those things.
14:51:50 So…
14:51:53 Trying to think from the perspective of
14:52:01 from, say, product management
14:52:09 it might be.
14:52:14 It might be interesting to have
14:52:16 some sort of tool that watches the bugs and
14:52:21 when it sees this whole
14:52:26 snowball effect.
14:52:27 that will happen sometimes with these bugs
14:52:30 maybe be like, okay, hey, this looks pretty heavy
14:52:34 maybe we should like
14:52:36 you know, make sure that a product
14:52:39 person knows about it.
14:52:41 and knows that, hey, this is chewing up a lot of time.
14:52:45 And you should be aware that
14:52:48 this is a difficult bug.
14:52:50 that is going to probably take up
14:52:52 a fair bit of time for maybe multiple engineers.
14:52:59 That might be useful.
14:53:02 if you could identify
14:53:05 shines that this bug is not
14:53:09 going to be solved or is about to fall through the cracks
14:53:14 That would be pretty cool.
14:53:16 We already have bots that attempt to
14:53:20 use heuristics to use
14:53:23 mark bugs that it thinks, hey, it looks like
14:53:29 something's gone wrong here. Like if this has fallen through the cracks
14:53:36 somebody needs to look at it and figure out why work is halted on it.
14:53:44 And so we have some bots that do that kind of work.
14:53:53 So it could be useful from that standpoint.

14:53:58 So insights from this might be useful
14:54:02 for the engineers that
14:54:05 work on these bots.
14:54:07 that kind of try to keep an eye on things and look for
14:54:12 areas where things have fallen through cracks or gone off of everything.

14:54:16 Sounds good. Sounds good.
14:54:19 We can move to the next question. By the way, we have a few more questions. It's 2.54. Is it okay we ask them or do you have any time restrictions?
14:54:27 i have i have
14:54:27 Yeah.
14:54:28 I have time.
14:54:29 Okay, okay. So, okay.
14:54:31 Thank you very much.

Q4
14:54:32 Next question is could the parents be used to solve new issues? If yes, how? Well, I guess we already touched on this to some extent, but do you have any thoughts on
14:54:40 Any additional thoughts?

14:54:43 actually solving
14:54:46 I mean…
14:54:50 we mean like broadly, right? I mean, obviously resolution has different stages, right? So and
14:54:55 if we can if the parents can help in any of these stages
14:55:00 Amazing.
14:55:08 So I guess that goes back to
14:55:12 or what I was saying before you asked this question.
14:55:16 applies.
14:55:18 like the bots trying to like mark things as stalled
14:55:23 that is a push.
14:55:25 towards a solution.
14:55:27 So in that respect, data like this could be useful.
14:55:34 As far as guiding
14:55:38 developer efforts goes
14:55:44 I'm not totally sure. I mean…
14:55:49 if you have…
14:55:51 I don't know whether you're angling at a
14:55:56 like a machine learning application for data like this.
14:55:59 I don't know if that's something you're thinking about at all.
14:56:03 but um
14:56:06 having that sort of guidance
14:56:10 for, you know, hey.

14:56:13 it looks like you're at this stage
14:56:15 do you need to do like a
14:56:21 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.
14:56:32 In some way.
14:56:34 I guess I could see something like that.
14:56:39 Mm-hmm.
14:56:40 Again, it's…
14:56:43 it's messy, right?
14:56:46 And…
14:56:47 anybody who's been in
14:56:51 you know a software engineering
14:56:54 project of this level of complexity.
14:56:58 knows that it's messy.
14:57:01 having process is great.
14:57:03 But it's not always going to work and you're going to have to
14:57:06 you know revisit things or circle back around
14:57:11 So…

14:57:13 Yeah. Okay. That makes sense.
14:57:16 Yep.
14:57:17 Let's get to the next question.
14:57:18 Yeah, so our next question is the follows.
14:57:22 could the patterns be used by Mozilla stakeholders to evaluate how well the issue resolution process is executed at Mozilla?
14:57:30 If yes, how?

14:57:33 So, I mean, you could
14:57:46 you could track maybe you could track
14:57:51 And maybe you could track the complexity
14:57:55 of the issue resolution process
14:57:58 in a given module
14:58:02 And…
14:58:03 get insights like, hey.
14:58:06 it looks like it looks like
14:58:08 most of the time when you touch this area of code.
14:58:13 it ends up being a slog.
14:58:17 like it ends up being this just
14:58:19 thing that runs on and on and on and requires multiple iterations
14:58:24 and pulling in multiple people and like trying to land it and then finding out it doesn't work.
14:58:30 And that could be a flag for a, hey.
14:58:34 maybe it's time to refactor this?

14:58:36 Maybe it's time to clean this up. Maybe you've got some pile of technical debt that needs some attention over here.
14:58:45 that would be maybe useful for people in
14:58:49 product management to be able to see, hey, look.
14:58:53 this part of the code base is
14:58:56 a tar pit and
14:59:01 we probably want to spend some resources
14:59:04 making it less ornery.
14:59:08 So I could see that maybe being useful. But again.
14:59:12 the developers know.
14:59:16 And if their manager and product people are asking them, it's like, hey, you know.
14:59:20 if you had to pick one area
14:59:23 that's like the tar pit in our corner of the code base like they'll tell you.
14:59:29 they know, believe me.
14:59:36 But maybe having data
14:59:38 could…
14:59:40 help justify additional spend, maybe?
14:59:43 Okay.
14:59:44 could be useful for product managers in that way.
14:59:49 I guess.

14:59:51 Yeah.
14:59:51 Sounds good.
14:59:55 And while this is a more general question, like, can you think of other potential usages of these findings and the patterns specifically to help improve the process at Mosino official resolution?

15:00:07 So, I mean, it's uh
15:00:12 longitudinal data would be pretty cool.
15:00:17 But again, that's, you know, it's cool, right? But the question is, is it actionable
15:00:25 I am not
15:00:27 sure whether longitudinal data or, you know, to what extent longitudinal data would be actionable.
15:00:37 Let me think.
15:00:46 I think it goes back
15:00:49 to
15:00:51 detecting when
15:00:54 a bug has fallen through the cracks.
15:00:59 That's a thing that…
15:01:02 happens uh and
15:01:05 it's good to know that it's happening and it's good to know why it happened.
15:01:11 So detecting that
15:01:13 Earlier

15:01:15 would be pretty useful, I think.
15:01:29 Maybe.
15:01:36 Well, that's not exactly related.
15:01:39 But…
15:01:44 I guess you could track statistics on how long reviewers take on average, but
15:01:50 Yeah.
15:01:49 some people might get embarrassed.
15:01:53 Right.

15:01:56 Okay, I guess we can go to the next question. Right, so this is about the older software systems or project maybe in other companies so do you think
15:02:05 They also follow a variety of workflows to resolve issues.

15:02:09 Oh, yeah. Yeah.
15:02:12 And particularly around code review, right?
15:02:17 Mozilla is
15:02:20 pretty heavy on
15:02:21 requiring code review.
15:02:24 and uh
15:02:26 other outfits aren't.
15:02:28 Like, you know, I've worked, you know, Skype and
15:02:32 code review was
15:02:35 not really a thing there.
15:02:39 So that varies tremendously.
15:02:43 the degree to which people write tests.
15:02:46 varies tremendously.
15:02:48 And we've only recently added stuff to our process
15:02:55 where you have to account
15:02:58 for when you're it's on the reviewer, actually. So when the code reviewer is doing a
15:03:04 they're supposed to mark the thing as
15:03:07 this is been tested. This is been tested.
15:03:10 And if it isn't.
15:03:14 they've got to
15:03:16 explain that. They've got to explain why in this case
15:03:22 we don't want or can't make a test.
15:03:28 And record that.
15:03:30 That is not something that we used to have, but we have this
15:03:34 testing flag thing that we do nowadays. And that's encouraged more people to write tests, which is great.
15:03:42 But again, multiple, you know, like I
15:03:47 I know that not every
15:03:50 software outfit writes tests.
15:03:54 They just kind of say, eh, I think the solution is this.

15:03:58 And then they might run it across continuous integration and then just land it and be like.
15:04:05 It's probably fine.
15:04:09 And the number of times where I've worked on something
15:04:12 And said, I think that should fix it. I'm gonna
15:04:16 test it now and found
15:04:19 No, in fact, that did not address the issue.
15:04:24 shows me that, you know, the testing is really important.
15:04:31 And…
15:04:36 Every single one of these things varies
15:04:39 by by
15:04:41 company and by team.
15:04:45 I could tell stories about any of this stuff, really.
15:04:49 But yes, it all varies tremendously and
15:04:53 the degree to which there's a process
15:04:56 varies tremendously.
15:04:59 So…

15:17:03 So this is our last question. Do you think our findings improved your understanding of Mozilla's resolution process?
15:17:11 If yes, how?

15:17:13 I mean, it's kind of cool to see uh
15:17:17 you know how often it's complicated and how often it's simple
15:17:23 That's actually pretty neat.
15:17:27 It's…
15:17:30 it makes me want to make me want to have more data, right?
15:17:35 want to see, hey, how does dom
15:17:38 do on this versus the networking people versus, you know.
15:17:43 the poor souls who work on css
15:17:49 That would be kind of neat.
15:17:53 But…
15:17:52 D2, is there a way to identify the teams from the issue tracker from some other system?
15:17:57 because yeah that that analysis is very interesting to us as well right so but we need to be able to
15:18:01 So, I mean.
15:18:03 to know who is working on which team, right?
15:18:07 So…
15:18:14 I mean…
15:18:19 publicly available data
15:18:23 recording who works on what team
15:18:25 I think you just have to glean it by looking at
15:18:30 Okay.

15:18:30 what component this person tends to work on.
15:18:38 they you know we've got our internal
15:18:41 you know, people directory
15:18:44 that records, hey, this person's on
15:18:46 thus and such a team. But sometimes it's even that's messy, like particularly with uh
15:18:54 more senior engineers
15:18:56 they tend to pick up things in multiple places.
15:19:01 And it's kind of hard to tell.
15:19:04 what team they might be on.
15:19:07 And, you know, what exactly their field of expertise is.
15:19:14 I think you'd have to go by component
15:19:19 Okay.
15:19:17 So, you know, we've got our component in the Bugzilla
15:19:23 tracker.
15:19:25 that would be how I'd try to do it if I were to do something like that.
15:19:32 Sounds good.
15:19:34 You could…
15:19:36 If you wanted to get more involved.
15:19:40 you could look at the change sets
15:19:42 and see what files they were touching.
15:19:46 Right.
15:19:48 But that would be a lot more involved and a lot more messy.
15:19:54 And you end up probably duplicating
15:19:58 the thought process that
15:20:00 every developer goes through who's like, what component do I put this in
15:20:05 Because it's not clear.
15:20:09 So that would be duplicative effort.
15:20:12 But I mean, it might also be interesting to see
15:20:16 Hey, like…
15:20:19 maybe these things need to be over in this other component.

15:20:26 Yeah. Okay. Yeah. Great. Thank you so much for the feedback and uh
15:20:32 That's it. I mean, let us know if you have any thoughts and if you're not thought about this and
15:20:41 Mm-hmm.
15:20:38 We'll follow up later. I mean, we're more than happy to share you know the result of this investigation so that you guys can learn from it as well.
15:20:47 We are still in the process of finalizing this investigation so
15:20:51 as soon as we finish, we can share you know our results for sure
15:20:55 That's really cool.
15:20:57 All right. Well, thank you.
15:21:00 And it's been a pleasure.
15:21:02 It's kind of fun to talk about this stuff with you know people

15:21:06 who are both interested and
15:21:09 don't already know it.
15:21:12 Absolutely.
15:21:14 Very small number of people.
15:21:17 So, uh.
15:21:18 Yeah, well, thank you so much, D2. It was really, really cool to learn from you.
15:21:23 So.
15:21:23 Yeah, yeah. Thank you very much for your time yeah we learned a lot about the process
15:21:28 Cool. All right. Well, thank you. Have a great day.
15:21:33 Yeah. Yeah.
15:21:32 Good luck on all your work and uh
15:21:35 You know, ping me when you've got uh
15:21:38 some more stuff to share.
15:21:40 Absolutely.
15:21:39 I'd like to see it.
15:21:41 Sure. Thank you so much.
15:21:42 That's good. Yeah.
15:21:43 Great, great.
15:21:43 Have a good day. Bye.