[Interviewer]
the different processes that

[Interviewer]
that you guys implement, right and um

[Interviewer]
Yeah, so thank you again for joining and i'll

[Interviewer]
Anto is going to lead the interview. I'm going to just ask a few questions here and there, but he's there.

[Interviewer]
He's the guy in charge.

[D1]
Sounds good.

[Interviewer]
Yeah. Thank you, Mehu, for signing the consent form.

[Interviewer]
And as a reminder, the results of this interview will not be personally identifiable

[Interviewer]
Which means we will not

[Interviewer]
I mean, we will anonymize the results if needed

[Interviewer]
And also we will record the meeting

[Interviewer]
to facilitate further analysis for research purposes.

[Interviewer]
Yeah, so let me start the recording.

[Interviewer]
Yes.

[Interviewer]

To get started, let me share real quickly

[Interviewer]
about the interview plan today.

[Interviewer]
So we structured the interview into four sections

[Interviewer]
In the first section, we will learn from you about your background in software engineering.

[Interviewer]
In the second section, we will know

[Interviewer]
more about how developers resolve issues in

[Interviewer]
Mozilla from you.

[Interviewer]
In the third section, we will give a short presentation

[Interviewer]
about our research findings on Mozilla's issue resolution process

[Interviewer]
And in the final section, we will be asking a few questions

[Interviewer]
to understand the usefulness of our findings.

[Interviewer]
Is the plan good?

[D1]
Sounds good.

[Interviewer]
Okay, so let's

[Interviewer]
get started with the first section.

[Interviewer]
Yeah, so the plan of this section is to know from you

[Interviewer]
to know about you better.

[Interviewer]
I mean, we want to know about your background and your experience in software engineering

[Interviewer]
including your development and issue resolution experience at Mojila.

[Interviewer]
your current and past positions

[Interviewer]
and the products and

[Interviewer]
components you have worked on

[Interviewer]
at Mozilla.

[D1]
Yep, sounds good.

[Interviewer]
Yes, can you please share us your background?

[D1]
Okay, so, uh.

[D1]
I joined Mozilla…

[D1]
Seven years ago.

[Interviewer]
Mm-hmm.

[D1]
So I have been on the same team the entire time.

[D1]
So I work on the SpiderMonkey JavaScript engine.

[D1]
And…

[D1]
Four, five, four…

[D1]
Somewhere around that five or four years of that time, I've been working on our triage team

[D1]
Which means that once a week we meet every Tuesday, we go through all incoming bugs.

[D1]
We tag them, point them at various developers as needed.

[D1]
A recent thing we do is we also link them into a

[D1]
tree of bugs based on topic and

[D1]
Yeah, at this point.

[D1]
I am…

[D1]
one of the more senior developers on the team

[D1]
But our team has very long tenures.

[D1]
Don't over rotate on that.

[Interviewer]
Yeah, sounds good. Thank you very much for sharing. So may we ask like.

[Interviewer]
do you have any other experience like for other projects other than Mozilla products?

[D1]
So prior to Mozilla, I worked at IBM as a full-time employee for three years.

[D1]
Prior to that, I worked as a student on call

[D1]
And then prior to that, I also did a 16 month internship at IBM.

[D1]
So all told, I worked at IBM for probably five or six years as well.

[D1]
You know, from internship through part-time work through full-time work.

[D1]
At IBM, I worked on

[D1]
three different projects that worked on a C and C++ compiler

[D1]
And then I worked on a transactional memory system

[D1]
And then I also worked on a Java JIT compiler that got turned into a Ruby JIT compiler that then kind of got canceled.

[Interviewer]
Wow, well, well, thank you. Sounds great. I mean, you have a lot of experience in development and software engineering.

[Interviewer]
That's cool. Thank you for sharing.

[Interviewer]
D1, can I ask you another question? So you mentioned that you're working on

[Interviewer]
spider monkey, which is a JavaScript engine, I believe, of Mozilla.

[D1]
Yes.

[Interviewer]
We have experience in other components or products at Mozilla?

[D1]
So I did the reimplementation of our streams implementation so the

[D1]
HTML streams specification used to exist inside of spider monkey

[D1]
We did an analysis and decided that it shouldn't live there anymore. And so

[D1]
I did probably 75% of the reimplementation of streams.

[D1]
And so that was all work inside of like normal Gecko code.

[D1]
And so on that half, I did that.

[Interviewer]
Mm-hmm.

[D1]
And then I was the module owner for DOM streams for about a year.

[D1]
After that, at which point, given that I'm not actually a DOM person.

[D1]
I eventually handed that responsibility off to somebody much more suitable to it.

[D1]
So I am, I guess, module owner emeritus.

[Interviewer]
I see. All right. Sounds good.

[Interviewer]
I think we can move on to this next section, which is about the issue resolution.

[Interviewer]
process at Mozilla. So we want to know if there is any prescribed process that Mozilla variables should follow.

[Interviewer]
And to what extent these processes follow in practice by Mozilla developers?

[D1]
Okay, so…

[D1]
There are…

[D1]
pieces which are largely okay so

[D1]
back up half a step.

[Interviewer]
Right.

[D1]
So it depends a lot on what you say when you or what you mean when you say Mozilla.

[D1]
And I mean this because

[D1]
Mozilla as an organization exists in many different pieces.

[D1]
And the overlap across pieces varies dramatically. So for example.

[D1]
While Firefox is our major dominant product.

[D1]
There are also products that are not Firefox, like Mozilla VPN. There are products like

[D1]
Pocket, fake spot, now Anonym that exist in like wildly different repos that have wildly different methodologies.

[D1]
Some of these exist on GitHub and are managed entirely through private GitHub issues.

[D1]
Some of them exist on GitHub and are managed through public GitHub issues.

[D1]
And then you have Firefox

[D1]
And a couple of other components.

[D1]
which live in Bugzilla and have follow like the Firefox development process

[D1]
And so when you say, is there an overall issue process prescribed by Mozilla.

[D1]
No.

[D1]
But I can talk to you about like the Firefox aspects. And then I can also talk to you about how Firefox's aspects

[D1]
you know, are diverged from in my team.

[Interviewer]
Yeah, sounds good. I mean, we would love to hear from about that.

[D1]
Okay, so bearing in mind that I can't speak about anything outside of the Bugzilla processes.

[D1]
There are a couple of pieces.

[D1]
that are important.

[D1]
And so these are relatively universal.

[D1]
um the

[D1]
Issues kind of have a life cycle.

[D1]
Like many things, they will start out

[D1]
as either public or private issues, which are either classified into three different buckets.

[D1]
There's defects, tasks, and enhancements.

[D1]
Various teams take different approaches to task versus enhancement.

[D1]
or sometimes even task versus defect.

[D1]
But this classification does matter in the sense that people pay different attention to

[D1]
what type of issue you have.

[D1]
So, uh.

[D1]
defects are tracked the most seriously by a variety of people within the teams.

[D1]
Across Mozilla.

[D1]
Things which are not defects.

[D1]

tend to have less visibility from other parts of the company.

[D1]
So by this i mean

[D1]
For example, when you fix a defect.

[D1]
one of the things that we try to identify is

[D1]
When was the defect introduced?

[D1]
Was it in a release which is still in service?

[D1]
Because we do maintain multiple in-service

[D1]
branches. And if it is a

[D1]
in-service branch, we have to make a decision. Do we want to take that

[D1]
fix and either build a new one for an older version that is in service.

[D1]
uplift the existing one.

[D1]
And take an effort and try to estimate the risk of doing so.

[D1]
And so this process is all run largely by a group of people

[D1]
We call them release management.

[D1]
So they are responsible for sort of keeping an eye on

[D1]
defects that are fixed, ones that affect in-service releases

[D1]
they go through and they will ask developers and be like, hey, I see this was introduced in an in-service release.

[D1]
Should this be uplifted? Could you request that it get uplifted?

[D1]
And then a developer will fill out a form that says, you know.

[D1]
here's my uplift request. It has a certain amount of risk. Maybe that risk is none. Maybe that risk is a lot.

[D1]
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.

[D1]
So they'll go through.

[D1]
And if the decision is made, they'll apply the patch to an in-service branch.

[D1]
And bring it along.

[D1]
We…

[D1]
defects also another important classification is secure.

[D1]
whether or not a bug is secure or not.

[D1]
Bugs which could have bad impacts on our users.

[D1]
And I mean this very broadly.

[D1]
tend to be opened in a secure state.

[D1]
So they will be, you know, if either we don't know what the impact is, but we hypothesize it could be bad.

[D1]
We will open these bugs in a secure state. This restricts the visibility down to a small subset of the company.

[D1]
From there, it will be triaged

[D1]
determined how severe it is, what is the impact? Is there actual user impact? Because sometimes the answer is that no, like

[D1]
There isn't. And then it'll be opened up.

[D1]
then what we will do is we'll go through and we will fix the bugs.

[D1]
Different teams

[D1]
have very different approaches for how they manage their bug backlogs.

[D1]
This is very much not

[D1]
like dictated from the top down

[D1]
The one thing I would do want to highlight that is kind of dictated is we we have

[D1]
The most important things that the higher level aspects of Firefox care about are

[D1]

Impacted releases and severity.

[D1]
Particularly for defects.

[D1]
Defects which are above a severity of three.

[D1]
we consider to be important enough that various people will pay attention to them. And severity one bugs

[D1]
have a lot of attention paid to them.

[D1]
We attempt to try to minimize the number of severity to our severity one bugs.

[D1]
SEV1 bugs are usually like

[D1]
very bad security bug must fix as soon as possible.

[D1]
This will involve an out of cycle release of Firefox, also known as a chem spill release

[D1]
um but

[D1]
Sev 2s are more common.

[D1]
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.

[D1]
is number of open severity to bugs.

[D1]
So, um.

[D1]

And sorry, open severity to defects specifically. Severity to enhancements, severity to, you know.

[D1]
tasks, nobody cares about. No, that's not true.

[D1]
They have much less visibility.

[D1]
uh…

[D1]
So, you know, to answer your question, to what extent do Mozilla developers follow this process?

[D1]
These aspects are followed

[D1]
pretty religiously, like almost everybody understands them

[D1]
follows them quite clearly.

[D1]
Everything beyond that becomes

[D1]
much more team level.

[D1]
there's a lot more aspects of it that are

[D1]
not as rigorous, not as well defined.

[D1]
And vary a lot more by team. Like, for example, how do you prioritize your work?

[D1]
different teams have very different perspectives on how this works.

[D1]

You know, there are a variety of strategies that various teams apply.

[D1]
etc.

[D1]
Sound good?

[Interviewer]
Okay. Yeah, no, great. Thank you for sharing. This is very useful

[Interviewer]
So let me tell you something that our

[Interviewer]
definition of issue resolution

[Interviewer]
So it's more focused as in we want to know a little bit if there is a

[Interviewer]
kind of a methodology or process to actually fix the problems in the code or change the code itself.

[Interviewer]
So what you talked about is more like

[Interviewer]
managing the books, managing the issues, right? And how to handle them like at a higher level

[Interviewer]
But we're wondering if there is some sort of guideline from Mozilla or about how people should actually go ahead and

[Interviewer]
try to implement the fixes, the issues, defects and tasks and enhancements you know

[D1]
I don't think so. Not at a high level.

[Interviewer]
Okay.

[D1]
Teams have various teams have various approaches for how to handle this. So to give an example like

[Interviewer]
Yeah.

[D1]
And I don't know how much you make a distinction between like

[D1]
defects and like enhancements or like feature work, that sort of thing

[D1]
But so for an example, in our team, one of our responsibilities is that as the JavaScript language evolves.

[D1]
And gains new features, changes definitions, et cetera. We have to keep the language up to date.

[D1]
And so one of a frequently recurring task for us is

[D1]
there is a new proposal that we have to implement.

[D1]
And so…

[D1]
we don't really dictate like, ah, go forth and implement it by doing X, Y, Z.

[D1]
But what we do have is, you know, like I wrote

[D1]
I don't know, three years ago, I wrote a feature development checklist.

[D1]
And so this checklist is

[D1]

you know, hey, you are implementing a new feature. Here's a checklist of things that you ought to be doing.

[D1]
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.

[D1]
You know, all of this sort of thing because

[D1]
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

[D1]
They vary in severity, but they can be problematic, right?

[D1]
you know, if you do it wrong, you could, for example, break add-ons.

[D1]
And of course, add-ons, it sucks because you don't notice that until some add-on author tries to actually use it

[D1]
And then they're like, hey, why doesn't this work? And it's like.

[D1]
oh crap, I forgot that add-ons are actually a little special. The world that they exist in is weird.

[D1]
And so you may have forgotten to deal with that.

[D1]
And so the checklist is very helpful to try to avoid that kind of problem.

[D1]
at a more high level goal.

[D1]
We don't have a lot of great like

[D1]
thou must go forth and do X, Y, Z.

[D1]
there are cultural expectations.

[D1]
So for example, like.

[D1]
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.

[D1]
That's a pretty strong one.

[D1]
But after that, it's like.

[D1]
How much testing should you write? Well, that's going to vary a lot depending on team.

[D1]
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.

[D1]
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

[D1]
You know, machine learning model to select what tests probably

[D1]
should run, you know, and different teams have different approaches to this

[D1]
Similarly, it's like

[Interviewer]
Mm-hmm.

[D1]

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.

[Interviewer]
Okay, awesome.

[Interviewer]
Thank you so much for the information. And well, this is, I mean, the last question is about you, like how do you personally go through actually solving the issues have like coding a fix

[Interviewer]
If you have any process that you use more frequently or it depends on the problem.

[Interviewer]
Yeah.

[D1]
So my typical approach

[D1]
And I would say this covers probably about 70% of my bug fixing

[Interviewer]
Okay.

[D1]
is…

[D1]
take bug report.

[D1]
turn into test case if test case is not already provided.

[D1]
I want to be able to reproduce this test case.

[D1]
Ideally, I want to be able to reproduce this test case in our JavaScript shell rather than the full browser.

[D1]
Because that eliminates an entire swath of complexity.

[D1]
And so it's much easier and faster for us to iterate inside of our JavaScript shell.

[D1]
So take shell test case.

[D1]
Hopefully, fingers crossed, it's not

[D1]
architecture specific. If it's not architecture specific.

[D1]
If I don't know just by looking at the backtrace

[D1]
approximately what I'm going to have to change.

[D1]
I personally, one of the very first things I do is I take a recording using RR.

[D1]
which is the record and replay debugger.

[D1]
generated by or that was built by Mozilla years ago. And then I submit it to a tool called Permosco.

[D1]
Pornosco is a…

[D1]
It describes itself as an omniscient debugger.

[D1]
So the omniscient debugger means that essentially it takes a recording of a program play and turns it into a database.

[D1]
And then it gives you a web interface to this database of program state.

[D1]
And you can debug through this.

[D1]
And Pernosco traces, in my experience, are anywhere between

[D1]
2 and 10x faster for me to actually like find problems.

[D1]
It is a huge improvement in my workflow.

[D1]
So I try very hard whenever I have a bug.

[D1]
to get a Bronosco trace because it is just

[D1]
amazingly better.

[Interviewer]
Okay.

[D1]
Once I have a Pronaska trace, I go to Pronosco, I poke around, I debug.

[D1]
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.

[D1]
I will fill in the notebook as I'm debugging.

[D1]
This is very useful because if I get stuck, I'm able to send this Pernosco trace the URL,

[D1]
to another developer and have them look at that and they can see all my notes.

[D1]
And when you click on a note, it jumps you immediately to the points in program state. It's great.

[D1]

So I'll debug it.

[D1]
And then I'll typically write up

[D1]
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.

[D1]
expect that this happens, then this happens, then this happens, but there's some ordering constraint that wasn't maintained.

[D1]
So now we have to make sure that that happens. And we can do that by doing this, that, and the other thing.

[D1]
I'll go and I'll start developing a fix.

[D1]
I use…

[D1]
Locally, I use VS Code Remote.

[D1]
I have a build machine that sits in my basement.

[D1]
And I use VS code remote to get onto it.

[D1]
I personally have switched lately to using jujitsu for version control.

[D1]
So I will just like start working on a new jujitsu change.

[D1]
start writing the code.

[D1]
Try to get fixed once I get something that is working, I'll then go back and I will

[D1]
try to make that fix sensible, you know, remove debugging code, sometimes split it, layer it, you know, whatever.

[D1]
my goal with

[D1]
My personal goal with writing patches is always to try to like

[D1]
Make sure that every individual unit is both reviewable, buildable, and correct.

[D1]
Ideally, if I'm being very, very good, I would like to tell a story through the development of this

[D1]
I succeed with that less than I used to but

[D1]
And then I will…

[D1]
Once I've got this fixed, so long as it's not a security bug.

[D1]
Then I'll submit a try build

[D1]
So I'll push it to our

[D1]
continuous integration server. It will do a whole bunch of building for me.

[D1]
I'll go off and do other things.

[D1]
in anywhere from half an hour to two hours, I'll come back and I'll look. I'll check the results.

[D1]
About 30% of the time I've made a dumb mistake, you know.

[D1]
something that is incorrect on some platform or like

[D1]
code that doesn't compile in opt but did compile in my debug builds, that sort of thing.

[D1]
So go back, rectify that.

[D1]
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.

[D1]
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

[D1]
why do these two things not agree?

[D1]
I get this all working. I'm pretty happy with it. I submit it for review.

[D1]
It goes up on fabricator. I tag a particular reviewer for my code.

[D1]
Usually because I know the team, I know who is the best, has the best context for this bug to actually do the review.

[D1]
And then I wait and I wait for them to actually provide review.

[D1]
Sometimes they'll provide review feedback where they're like.

[D1]
Cool. It's good. Land it.

[D1]
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.

[D1]
you know, oh, look, here's a little test case that will blow up in your thing.

[D1]
that sort of thing happens.

[D1]
iteration through review

[D1]
And finally, once it's been approved for review.

[D1]
I will request landing, which goes through a tool we use called Lando. And I press a button.

[D1]
And then eventually it shows up on our

[D1]
what we call central, which is our like main branch

[D1]
And it will get built through our continuous integration.

[D1]
And in about 12 hours, it shows up in a nightly build.

[D1]
And so then I can test it in a nightly build.

[Interviewer]
Okay.

[D1]
If I really want to.

[Interviewer]
Awesome. Great. No, this is very, very useful.

[Interviewer]
Quick question, I mean, before we move on to the next section is this process that you described is it

[Interviewer]
fairly common, commonly used by other developers as well.

[D1]
I would say that for people who are working on gecko, the rendering engine.

[D1]
As well as Spider Monkey.

[D1]
About 75% of this will have pretty strong overlap between everybody.

[D1]
like the

[D1]
you know, try to build a test case, the part where you

[D1]
build and iterate through review.

[D1]
Using the tri server

[D1]
landing it through Lando, doing review, that sort of thing. That all is pretty good. That's going to be pretty common all the way across.

[D1]
the place where I'm slightly unique is, you know, I am one of the biggest proponents inside of Mozilla of Pronosco. So I use it a lot more than maybe a lot of people.

[D1]
Despite my best efforts.

[D1]
Um.

[D1]
And, you know, uh.

[D1]
I tend not to end up having to work on things that are

[D1]
just the nature of the stuff that we do

[D1]
the nature of the kinds of things I personally work on, I tend not to end up in situations where I have to do a lot of like

[D1]
I need to boot up a whole browser and visit some websites. I don't have to like visually compare you know

[D1]
this rendering looks different than this rendering, but it's okay, that sort of thing.

[Interviewer]
Okay.

[D1]
And so other people in Mozilla will have more of that experience.

[D1]
And then there's the broader like Firefox organization where it's like building the actual user experience and things like this. And they will have whole difference problems

[D1]
about like, ah, now you have to do like

[D1]
user testing or now you have to like localize strings or now you have to do, you know.

[D1]
CSS work and like what happens if it becomes night mode and like all

[D1]
All of this stuff is like well outside of my wheelhouse.

[Interviewer]
or best friend.

[Interviewer]
there's different settings right of the browser, right? User settings and things like that.

[D1]
Yes.

[Interviewer]
Sounds good. Thank you. Thank you for sharing. And I think we can move on to the next section.

[D1]
Sure.

[Interviewer]
So in this section, we will be sharing our

[Interviewer]
research methodology and findings.

[Interviewer]
on Mozilla's issue resolution process.

[Interviewer]
So let me get started with the goal.

[Interviewer]
So our research goal is to understand how Mozilla developers resolve issues in practice.

[Interviewer]
That means we want to know

[Interviewer]
which steps they perform specifically

[Interviewer]
for resolving the issue. One thing that I want to clarify here is that

[Interviewer]
We are more interested in the activities that developers perform

[Interviewer]
After the issue.

[Interviewer]
is assigned to a specific developer. That means we assume issue triaging is already performed.

[Interviewer]

So we are more interested in the activities from pre-production, the analysis, solution design

[Interviewer]
implementation until code review and verification.

[Interviewer]
So we are particularly focusing on these stages or steps.

[Interviewer]
So our

[Interviewer]
Second goal is to

[Interviewer]
understand the workflow

[Interviewer]
of these stages that means how mozilla developers

[Interviewer]
Apart from these stages and

[Interviewer]
with the help of this, we want to understand the issue resolution patterns. That means the common ways of resolving issues.

[Interviewer]
And our final goal is to derive actionable guidelines for

[Interviewer]
Mozilla stakeholders so that they can

[Interviewer]
you know, resolve the issues more effectively.

[Interviewer]
Because for improving the process or for assisting developers, at first we need to understand the process.

[Interviewer]
So that's why we are interested to know

[Interviewer]
how Mozilla developers resolve issues in real life.

[Interviewer]
So now the question is how we can

[Interviewer]
understand the process.

[Interviewer]
So we hypothesize that

[Interviewer]
we can investigate the developers discussion in the issue report comments

[Interviewer]
And then we will be able to know which activities they perform for resolving that issue.

[Interviewer]
And from those activities we can understand the process.

[Interviewer]
So we assume that developers

[Interviewer]
record the resolution related activities in the issue report comments.

[Interviewer]
So this is our plan.

[Interviewer]
And with the plan in our mind we

[Interviewer]
identified the issue resolution patterns

[Interviewer]
in two phases.

[Interviewer]
So in the first phase, we conduct

[Interviewer]

conducted a qualitative analysis on the issue reports focusing on Mozilla Core and Firefox products.

[Interviewer]
So we collected a sample of 356 issue reports from these two products.

[Interviewer]
And we engaged seven researchers

[Interviewer]
where they investigated all the issue report comments

[Interviewer]
from this issue report

[Interviewer]
And they identified the issue resolution activities, for example, issue reproduction or

[Interviewer]
designing the solution or implementing the solution and so on.

[Interviewer]
so and then in the second phase

[Interviewer]
by utilizing these resolution activities, we identified six issue resolution strategies that I shared

[Interviewer]
earlier and

[Interviewer]
with the stages we created

[Interviewer]
a state sequence for each issue report

[Interviewer]
And finally, we identified the patterns. So let me share one real example.

[Interviewer]
to demonstrate the process so that we can easily understand

[Interviewer]

So here I have one example issue report

[Interviewer]
I'm sorry because this figure is not clearly visible, but I hope you understand this is a this is an issue report from Mozilla, right?

[Interviewer]
So we investigated all the comments

[Interviewer]
developers made.

[Interviewer]
one by one and we identified by

[Interviewer]
the issue resolution activities, for example, reproduction attempt.

[Interviewer]
That means one developer

[Interviewer]
attempts to reproduce the issue.

[Interviewer]
And we also found one developer stated the problem cause

[Interviewer]
or they design the potential solution or they implemented the code and so on

[Interviewer]
So we identified these activities.

[Interviewer]
which are related to issue resolution.

[Interviewer]
And then by using these activities, we group the activities into the stages.

[Interviewer]
For example, reproduction attempt implies

[Interviewer]
issue reproduction, right?

[Interviewer]
and potential solution implies

[Interviewer]
designing the solution.

[Interviewer]
And code implementation implies implementation.

[Interviewer]
That means for this particular issue, we identified four issue resolution stages.

[Interviewer]
And their reproduction analysis, solution design and implementation.

[Interviewer]
And we can create a sequence

[Interviewer]
by these four stages.

[Interviewer]
So we call it the test sequence for this issue.

[Interviewer]
So that means for this particular issue we have

[Interviewer]
a stress sequence which we can represent by this string

[Interviewer]
Now, in the same way, we identified

[Interviewer]
the stress sequence for all the 356 issued reports.

[Interviewer]
And then we grouped

[Interviewer]
the state sequences based on their similarity.

[Interviewer]

In this case, we identified five more issue reports which have the similar

[Interviewer]
stress sequence.

[Interviewer]
like this so and then

[Interviewer]
by combining these stages

[Interviewer]
we created a more abstract pattern

[Interviewer]
which can represent all the state sequences.

[Interviewer]
And we represented by this string.

[Interviewer]
However, we don't need to understand this dream

[Interviewer]
Because we showed this

[Interviewer]
process by this figure so this figure

[Interviewer]
That means for resolving these six issues here

[Interviewer]
developers at first reproduce the issue, then they analyze the issue

[Interviewer]
They designed the solution, they implemented the issue.

[Interviewer]
I mean, they implemented the solution

[Interviewer]
And based on the solution, maybe there is some optional code review or verification.

[Interviewer]
So this is the workflow for this

[Interviewer]
six issue reports.

[D1]
Shouldn't there be a back edge from code reviewed implementation here?

[D1]
For the last issue.

[Interviewer]
So for particular

[Interviewer]
so this for this particular six issues

[Interviewer]
we did not observe like there there can be multiple implementation or code review

[D1]
I mean, the last one has one.

[Interviewer]
Just to clarify.

[Interviewer]
Just to clarify yes i mean

[Interviewer]
So just to simplify the graph here, but you're right. I mean, you can go from code review to implementation, right?

[D1]
Okay.

[Interviewer]
Yes. Yeah. So, yeah.

[Interviewer]
So let me share some more example to understand the patterns.

[Interviewer]

So, for example, this pattern means there can be iterative stages

[Interviewer]
like implementation, then code review and then verification and then it can be cyclic like developers can pass from these stages repetitively

[Interviewer]
And we found this pattern among

[Interviewer]
28 issues. And remember, note that there are no other stages here

[Interviewer]
However, in the second example.

[Interviewer]
is very similar to this one

[Interviewer]
But here we have one solution design stage at the beginning that means

[Interviewer]
developers are from solution design first and then they go to implementation then code driven verification which can be repetitive

[Interviewer]
And we found this in 13 issues.

[Interviewer]
And in the third example.

[Interviewer]
We have a similar pattern however

[Interviewer]
Here, the solution design is a part of the repetitive cycle that means

[Interviewer]
like after implementing the solution and reviewing the code or verifying the code.

[Interviewer]
they needed to design the solution again because maybe the previously implemented solution

[Interviewer]
could not resolve the issue. That's why they needed to redesign the solution.

[Interviewer]
So please notice that although these three patterns seem similar, but they are fundamentally different.

[Interviewer]
So yeah, so in this way, we identified 47 patterns

[Interviewer]
for 356 issue reports.

[Interviewer]
And here.

[Interviewer]
I'm going so fast because like we do not have much time

[Interviewer]
We have to ask you some questions about the patterns.

[Interviewer]
So let me ask if you ask me

[Interviewer]
have any uh you know

[D1]
No, keep going.

[Interviewer]
Yeah, so among these 47 patterns, we have 27 simple patterns which means there is no cycle or repetitive stages

[Interviewer]
However, 20 are complex patterns which contains the repetitive stages.

[Interviewer]
And if you notice here like uh

[Interviewer]
70% of the issues were solved

[Interviewer]
using 27 patterns that means

[Interviewer]
they were potentially easier

[Interviewer]
to resolve.

[Interviewer]
Yeah, here we represent we presented uh

[Interviewer]
top 12 patterns

[Interviewer]
from the 47 patterns

[Interviewer]
So we do not have to understand the notations and everything. Here, just we need to understand that like

[Interviewer]
you can see some patterns are more frequently used

[Interviewer]
For example, this one we found in 64 issue reports.

[Interviewer]
However, some patterns are less frequently found. For example, here, in this case, we have

[Interviewer]
only seven issues for this pattern.

[Interviewer]
And we also have other patterns which is only found in one or two issue reports. So yeah.

[Interviewer]
so and we listed some findings based on our investigation

[Interviewer]
across these patterns, across issue type or problem categories and so on.

[Interviewer]
Yeah, so that's all from my side

[Interviewer]
Let's briefly mention the findings and just i mean

[Interviewer]
go through.

[Interviewer]
Okay, sure. So you can see like as we found uh a lots of patterns for 356 issue reports that means

[Interviewer]
there can be a diverse way of

[Interviewer]
resolving issues in Mozilla, right? And the process deviates from the linear process

[Interviewer]
which we shared before. So it's not the sequential process and the top 18 patterns I found

[Interviewer]
in 80% of the issues which means

[Interviewer]
like there are recurrent patterns.

[Interviewer]
And we observed the diversity of the patterns

[Interviewer]
across 14 years, that means 2010 to 2023

[Interviewer]
And we found some complex patterns are more frequent in

[Interviewer]
problem categories, for example, code design, defective functionality, or UI issues.

[Interviewer]
And, uh.

[Interviewer]
We also observed that issue resolution is more diverse in certain issue

[Interviewer]
problem categories that that is

[Interviewer]
a defective functionality code design and UI issues.

[Interviewer]
Yeah, please feel free to ask if you have any question.

[D1]
I have a whole whack of questions, but in the interest of letting you guys ask me the questions.

[D1]
I can save mine until the end. I've taken notes, so…

[Interviewer]
Okay, sure, sure. Thank you. Thank you. So yeah, now I think as we do not have much time, let's move to our next session section

[Interviewer]
where we will be asking you if sir a few questions

[Interviewer]
to know about the usefulness of these findings.

[Interviewer]
of our research. Yep.

[Interviewer]
So let me start with the first question.

[Interviewer]
So in your opinion.

[Interviewer]
could the identified issue resolution patterns be useful in any way for modular stakeholders?

[Interviewer]

So please note that here by stakeholders we mean

[Interviewer]
developers, project manager, or anyone who is involved in the issue resolution process of Mozilla.

[D1]
Maybe. I…

[D1]
I think it could be interesting.

[D1]
If, you know.

[D1]
let's say, let's say the end state of this is that we automate the ability to highlight certain patterns in a bug.

[D1]
It may be interesting.

[D1]
to have bugs which have gone

[D1]
down particularly complex paths be highlighted

[D1]
with a question of like.

[D1]
Okay, it feels like something went wrong here.

[D1]
like if you went through three different implementations and if you went through three different verifications and it still didn't work.

[D1]
Like something went wrong.

[D1]

And so there's a, you know, 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?

[D1]
And then you can say, you know, root cause is a logic error. It's a memory safety error.

[D1]
And then from that, you can draw conclusions where it's like.

[D1]
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.

[D1]
I could imagine that this sort of thing could be useful to highlight like

[D1]
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.

[D1]
Given that we thought we had a solution and we went through multiple solutions.

[D1]
Having said that.

[D1]
it feels pretty obvious when you look at a bug, but like

[D1]
if you've got this sort of thing, like.

[D1]
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

[D1]
gone wrong by some you know

[D1]
gut feeling, maybe we can say, hey, like, yeah, something happened here and maybe we should have changed how we developed this.

[D1]
Beyond that.

[D1]
I think one of the things that I'm very, you know, this is one of my questions

[D1]
I don't think you have the answer yet but

[D1]
qualitatively

[D1]
This feels like the normal structure of, this is kind of what I would expect from the average software project.

[D1]
If you went through the bug tracker at IBM, if you went through the bug tracker at, you know.

[D1]
Red Hat or something like this, I feel like you would actually end up seeing a pretty similar pattern across all of them.

[D1]
And so I would be curious if there was

[D1]
actually something that stood out here as being like.

[D1]
in Mozilla much more than

[D1]
other comparable company

[D1]
you see a lot more of patterns

[D1]
that look like this, you know, complex, large, recursive patterns about, you know, issue resolution

[D1]

Or maybe not. Maybe, you know, ours is more linear than the average company. I don't know.

[Interviewer]
Yes.

[Interviewer]
That's a really good question. And we still need to get there, right? This is kind of the first step in the investigation and uh

[D1]
For sure.

[Interviewer]
We totally agree and um

[Interviewer]
Yeah, it would be interesting to see the differences across companies.

[Interviewer]
And well, yeah, we will get there eventually and happy to share the results with you and mosil in general about that. But that will take some effort from our side as well.

[D1]
For sure.

[Interviewer]
Yes.

[Interviewer]
Let's go to the next question.

[Interviewer]
So by the way, it's right now 1.48.

[Interviewer]
Are you okay if we ask a few more questions like maybe

[D1]
Sure, I can go until about the hour, maybe plus or minus, plus about five minutes, but

[Interviewer]
Okay, okay, okay.

[Interviewer]
Sounds good.

[Interviewer]
Could the parents be used to train new Smozilla developers on how to solve issues? If yes, how?

[D1]
I think so, because I think, you know.

[D1]
As I said, we have a lot of diversity about how things actually work.

[D1]
And one of the things that would be good is to highlight common patterns

[D1]
for developers so that

[D1]
And this is just sort of one of these things about sort of psychological safety in a sense, where

[D1]
By sharing common patterns, being like, hey, look, like.

[D1]
You know, in your little graph here, you have the recurring solution of impulse

[D1]
code review and verification.

[D1]
happening 28 times. If we can tell people it's like.

[D1]
you can expect that sometimes you will write a solution

[D1]
And it will not be the solution you land. And that's okay.

[D1]
like it happens. It's normal. It's a normal part of the process, particularly for junior developers, this can be very helpful

[D1]
Because people set a very high expectation for themselves to get it perfect.

[D1]
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.

[D1]
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.

[D1]
In order to have that conversation about like, okay.

[D1]
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?

[D1]
having these kind of conversations concretely against code is something that I find juniors are not always amazing at.

[D1]
Because they want to be succeeding.

[D1]
And so if we have these patterns where we can be like, no, look, like, listen.

[D1]
During your onboarding, here's a normal pattern for doing development at Mozilla.

[D1]
you will probably have to iterate. You will probably have to go through this solution more than once.

[D1]
And that's okay. That's expected. It's part of the job.

[D1]
That could be helpful, yes.

[Interviewer]

Awesome.

[Interviewer]
Thank you. Yeah, so a third question is, could the patterns be used to estimate developer seafood?

[Interviewer]
to resolve issues? If yes, how?

[D1]
I don't think so.

[D1]
you know it's

[D1]
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

[D1]
Or are you saying prospectively

[D1]
looking at a bug or a sequence of comments and being like, this is going to take

[D1]
some amount more, you know, some amount more

[Interviewer]
So let me explain very briefly. So this is the idea that we have actually. So imagine that we get, it's actually for a new book.

[Interviewer]
or a new issue that comes in.

[Interviewer]
we want to kind of try to estimate if it will take a little long, right, than expected

[Interviewer]
So if the issue is related to a

[Interviewer]
prior issue is kind of similar to a prior issue, right? And we know the pattern and the sequence of that issue based on the comments, right?

[Interviewer]
And if it's long, for example, if it's complex pattern, then maybe that will give an indication that this new issue also may be

[Interviewer]
potentially complex, right? So that's the idea that we have in mind. What do you think about that?

[D1]
Maybe. The thing I think about it is that it's not so much about…

[D1]
estimate in a sense. It's more about maybe highlighting

[D1]
You know, if you can identify common causes for why issues end up doing these sort of cycles.

[D1]
that could be, or even issues that should follow a certain pattern

[D1]
And maybe aren't currently structured to.

[D1]
So for example, like, um.

[D1]
Some bugs that we do require human verification.

[D1]
We can't automate the test cases. It requires, you know, installing third party software on a machine. Sometimes it requires human verification.

[D1]
And it's up to developers to highlight when this is the case. And so if you had a tool that said, hey.

[D1]
This bug that you open looks like it might need some third-party support for verification.

[D1]

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

[D1]
maybe this bug's too complicated.

[D1]
Maybe you've tried to cover too much in one bug.

[Interviewer]
Mm-hmm.

[D1]
We have this idea of a meta bug, which is a bug which hosts a whole bunch of related bugs.

[D1]
you know, this could be saying

[D1]
you know, the tool could come out and say.

[D1]
This kind of sounds like a meta bug

[D1]
And then maybe you could spit out a suggestion of like, could this be split? You know, here are some topics that it sounds like you could split this down or something like that.

[D1]
I emphasize the maybe on this just because I think

[D1]
Tooling like this is very, very hard and people are extremely opinionated.

[D1]
about how to manage work and workflows.

[D1]
And so it's

[D1]
it kind of has to be really, really compelling in order to even get people to even try it.

[Interviewer]
That makes sense, yes.

[Interviewer]
Yes.

[Interviewer]
Okay, in the interest of time, let's move on to the next question.

[Interviewer]
Yeah, so…

[D1]
Um.

[Interviewer]
Right. I mean, I think we've already talked about this so next

[D1]
Okay.

[D1]
Yeah.

[Interviewer]
Next.

[D1]
I think, yeah, just to give the concrete answer, I think the thing that would be helpful is maybe highlighting where like.

[D1]
a bug is expected to follow a certain pattern and maybe will require

[D1]
some aspect of this.

[Interviewer]
Right. Okay, awesome.

[Interviewer]
We already talked about the evaluation of the process, I think.

[Interviewer]

Yes.

[Interviewer]
And.

[Interviewer]
Yeah, I think we discussed this already.

[D1]
Okay.

[Interviewer]
Yeah, I think we also

[Interviewer]
I think we've already discussed this. Yes.

[Interviewer]
this as well. Yeah, yeah.

[Interviewer]
Okay.

[D1]
Yeah, the other thing I would highlight is also matrix. Like we used to use irc

[Interviewer]
All right.

[D1]
transition to Matrix about four years ago, three years ago, somewhere around there

[D1]
So that's another place that conversation happens that

[Interviewer]
Yeah, yeah, yeah.

[D1]
Yeah.

[Interviewer]
Can you go up uh sorry before can you

[Interviewer]
Previous one?

[Interviewer]
This one.

[Interviewer]
Previous one.

[Interviewer]
Yeah, this one so so

[Interviewer]
Are the developers supposed to record activities in issue reports?

[Interviewer]
Or to what extent? Yeah, to what extent

[D1]
It's definitely varies by team.

[Interviewer]
Mm-hmm.

[D1]
And it varies, frankly, by people's personality.

[D1]
Some people are much more willing and interested in sort of sharing

[D1]
you know, good write-ups of like what happened

[D1]
Here's what we did. Here's the experiment we ran, et cetera, et cetera, et cetera.

[D1]
Other people are much more like

[D1]
issue, hear a bug or like here's patch like you know not not interested in talking about it, but just like.

[Interviewer]

Right.

[D1]
I will provide you the solution.

[D1]
The biggest requirements are things related to like

[D1]
And backboarding and security and like, you know, does this need a chem spill release? Does this need to have a CVE associated with it? Does this need to have a

[D1]
You know, backported fix, you know, that sort of thing.

[D1]
Those are the places where like resolution activities are much, much more rigorous.

[Interviewer]
Okay.

[Interviewer]
Yeah.

[Interviewer]
Yeah, I think we covered this question.

[Interviewer]
Do you think our findings improve your understanding of the process? If yes, how?

[D1]
I don't think so. This largely matches my like

[D1]
personal feelings about how a lot of pieces fit together.

[D1]
I think it all seems fairly normal.

[Interviewer]
Okay.

[D1]
I feel like…

[D1]
You know, there's that part of me that has curiosity about like.

[D1]
What is the longest path through some of these and what happened in those bugs and why did that happen?

[D1]
And like that that

[D1]
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.

[D1]
you know, it happens. So it doesn't

[D1]
I didn't find it changed my understanding that much.

[Interviewer]
Okay. Sounds good.

[Interviewer]
I think at this point we can

[Interviewer]
conclude the interview. Let us know if you have any question, any comment before we actually

[Interviewer]
Well, D1, thank you so much. We need to stop right now because we have another interview coming up.

[D1]
Yeah. And my son's knocking on my door anyhow, so.

[Interviewer]
All right. Sounds good.

[D1]

I got to go.

[Interviewer]
We'll get in touch via email about what's next. But it's about mostly about thanking you and maybe if you want to know more about the research, we're happy to give you more details.

[D1]
Sounds good.

[Interviewer]
Thank you, D1. Have a good day.

[Interviewer]
Yeah, thank you very much for your time, D1.

[D1]
Bye.

[Interviewer]
Bye. Have a good day.