Dawid Wozniak:

So, thank you for participating the first thing at the beginning I will show you the presentation. Then what we have now and then we talk about something that I would like to know - your answers for some questions. I will just seat here. I'm doing the transcription of the meeting that you can see on the screen now. So, you don't need to give me any feedback outside of this meeting. I will just get everything from it. It is a proof that we actually did it and I just didn't create it by myself. So, let's start with some introduction. Let's make a commit history great again. That's the first slide and my problem. The full research question is, which insight about commit messages are useful for stakeholders when visualising project using GitTruck? We don't use commit messages to help us with our work. It is just the general statement that I figured out when I look into this topic. So if you think about it in the terms of the traffic, I would imagine it as we are going somewhere, we are putting some information there. Then, we just turn around and going in the different direction. The real problem is that we don't know really how to do it, how to get some useful insights from those commit messages as you can see there are many different ways how you can present you need to turn left, right or go straight and it is what we actually want to figure it out. So, we want to get there. We want to drive in the right direction, and you can help me to get there. Basically, imagine yourself as a mechanic and I will give you my car so the product that I'm building and you will help me to give it there. This kind of a new brand new look and brand new features that actually we need. I will give you the tools I will implement the changes that you request to implement and you just can say whatever you want and then I will try to make sense of all feedback together to make the car as perfect as it can be. So, you only provide the feedback. You have a car and then you give me some lists of the things that you think might be relevant for this particular model. At the end we were iterate over the solution. So, the idea is that at some point of the project some new people and some people who did the previous iteration will join and talk about what is good, what is not, how they understand changes. So, we have fresh eyes and the people who also knew the previous versions that they can compare with, see what the progress is. That's the end of presentation. Now we can go to the product. So the product looks like this. This view has just one repository. It's analysed because I pre-analysed it for now, otherwise it takes a long time and that is why I did it. We have a small issue with the scalability, it's not fixed yet. That's why we decided for the first iteration it‘s just this project analysing itself. So, you can also see how it's built. So we have on the left side some information about branches, when it was analysed, which commit, how many files are in the project. Some things that are not relevant, like open an issue and ask the question. Here, you have the whole project. So, each bubble is a file. When you have some folder, it is a big bubble containing other bubbles. If you click on something specific like *demo* file. You get some extra information about size, how many this file was changed, last change, where it's located, author distribution and commit history. This is actually the part that I am responsible for, and I take care of it. So, it's a commit history. For now, it's very simple. So, by default you have just three last commits visible here and they are sorted by date. It means that you will have more than one commit during the first September or 1st February this year they will show up here. Here We had just one. It is some auto generated “update of the dependencies” and then you can expand the view to see some previous commits. It is from the latest to oldest. Also, if we have some duplicates then we just remove duplicates, but it needs to be 100% of duplicate. The repository that we're looking at is GitTruck.

Ayrton:

You run it on itself. Essentially.

Dawid Wozniak:

Yeah. If it's for the first thing because we have this problem of scalability. So, if you want to load it for the big repository, you will get out of memory stack JavaScript error. So. you can see that in the URL. Now, we have just the one static URL. It means that everything for all files needs to be preloaded, precomputed. I'm working on that. It's now in progress. It should be that you have some information precomputed, but if you click on the file you get this information on demand. So, this behaviour will change and it will allow us to load some extra information for the biggest projects, so let's take look at another file here. You have some component *details*. It was a little more times changed, so the list can be actually a long list and now you have just this wall of dates. You can just open it and see that at this day there was a quite few changes. It was one „correction", another, another, and another. So, it seems like someone was working on this day on this particular part of the code and this is what I want to say. Now if you have any suggestion, any first impression, it's time to say it before I ask specific questions. Yeah.

Ayrton:

You showed me this and we talked about this before, right? I thought about it in terms of like, how would we use this, right? What you said… you didn't think it was so useful because like, if you're running it on a small repository then generally it's very easy to get familiar with the repository. So, you don't really need this, right? You look at it once and then you never look at it ever again because you've already known it, without a product, I don't think that that is true. I think it would actually be really useful because maybe I don't know, I think it depends how your code is organised, right? But at least in terms of business central and the platform, essentially, it's like many different projects like different processes and like modules of the product but they're actual executable programmes themselves. So, they listen to code that runs in the browser, which is all like JavaScript, TypeScript, C#. There's code that runs on the web server side which is like C and then the backend server that communicates with the data, that's also C. There are test projects and like release manifests and things like that. So, it's like lots of distinct projects where somebody would work on one small area in a repo of thousands of files and hundreds of directories, right? This project is huge and maybe on our team we only took like 1/5 of those files and there's some other teams looking at completely different 1/5. There's like slight overlap between those two projects. But generally, like, you work in your small area and some other teamz work in other small areas and at least within our area it's the same problem you were saying where it's a small code base where I'm aware of who works on what, which files belong to which feature, blah blah blah. But when it comes to other teams, that's where I really struggle, right? So, we really struggle if I have a bug that appears to be on the server team to do with like database management or, you know, like part of the product that makes an API call to their service and this problem is in their services. I can, sort of see, where it is. I can see the part that I'm calling, but I can't see any further than that because I don't really know enough about that structure of that project to understand what happens next, after I make my call? So, what I wanna know is who is the person that I should speak to about this problem. So, what would be really nice is if I could just go to a diagram like this, look at the files of their components, see where I'm, see who has edited files in the history of that file. Right? Often it's not good enough for me to look at the file where the problem is, because that might be a file that has not been touched in a really long time or has been touched very few times, like it was written once and then that was it. Maybe, the person who edited that file, doesn't work here anymore. So, what I would like to see it's who has made the most changes around that file, right? Who's made the most significant aspect of that particular service. It would probably be who has made the most changes in that directory or a parent directory and you keep going up, right? So, like for different areas of this map, I would like to see who has made changes within sort of this general circle. Right. Who was like, top changes maker?

Dawid Wozniak:

Yeah. So here we have this metric that is called top contributor, and they're assigned some colours to people so here we have the list of people who contributed to this particular file, and you can see how many people did changes. The changes were done by this particular person. So, for example, if let's say that these all represented individual components and team's developers are on the metrics.

Ayrton:

I have a problem with one of these components, then I wanna look at that component and see not only who has touched this component the most, but who's touched the components that surrounded the file. Those which are related to it the most, right? It is not just in terms of like files that are in the same folder but files that are often submitted in the same commit, right? Because you might have some components like all of the components live in a folder called *components* and maybe there's another folder called like *styles*. So, like if you look in the components and the files around it, they all could be completely unrelated features that are in the same folder. Like a folder is not necessarily a boundary for distinct features or areas of work for one person. It could be like files spread across many folders that one person touches. It's not like there's a correspondence of my work in area to the particular directory but there is my working area spread to the files across multiple directories, so instead of grouping things by folder, it might be nice. It's a grouping things by like commonly committed together files. Not necessarily visually grouped them, but like highlight them based on commits. Files that often appear in the same commit. Then I can see like maybe this file is only touched once, but it's often touched in pull requests to all these other files and then I can look at the usages of those files and see who edits those files, right? Because for me this tool, the primary use would be to find ownership of areas that I'm unfamiliar with, and there are a lot of areas that I'm not familiar with cause the product is massive. So, you know, like, I think on a very small scale. So, it doesn't make sense to use it for that. But when you work on a really huge legacy code base, it definitely has a use for that purpose. I find myself doing that all the time, like how many times have you asked me a question where, I'm like, I don't know the answer to that. So, let's just look in the Git history to find out who the right person is. It's often annoying because looking through the commit history, I might see somebody from the infrastructure team who like moved it to that folder or I might see Cody, who renamed all the TypeScript files at some point. Right? It's like he doesn't know those files. Obviously, he just touched them once, but if you look at his commit usage, he's everywhere in the company, so being able to group things by like commonly touched files together and see who owns those. That's the thing. Another thing that I would really wanna see is there's a lot of time that I am trying to write a pull request and I know that there's something that I need to do. It's like if I had a new image to the product and then we have a file called the DVD manifest. If you added an image that needs to be on the DVD, then you need to add the name of that file in the location to the DVD manifest. So, if you add an image to this folder, then you must make the DVD file changed and there's lots of things like that where like if you edit this file, you must also edit this file, like feature keys. For example, if you add a feature key on the browser side then generally you wanna add the feature key on the server side. But those files have different names. They live in different folders, and you know usually I'll do one of them, then I would forget one and I'll build my changes and test it and be like why doesn't this work? It's because I forgot to make the change to one thing and then I can't remember what the file is. So, I messaged Simona to say like can you remember what the file is. What's the feature key file called in the server code? But if I could just go in here and say like if I took this file and there's like a 90% correlation between commits to edit this file and commits to edit this file then generally, I need to make changes in both, right? I can see that if I change this I also need to change this because when you're dealing with

legacy code, it's really difficult to remember it. If I update some things that I should also update other things that I have to update. That would save us a lot of time in terms of not having to wait for more builds failing. Then, let's say I forgot that file and manually test it. I have wasted my personal time manually testing and manually rebuilding it, but also in terms of sending jobs in the gate where I sent a job and it failed. I wasted Azure resources, and it costs us money to run those test jobs where some obvious thing should have been done. Just like two things that I would say they are important.

Dawid Wozniak:

OK, I have a question specifically about commit messages. So, in this scale from one to ten, when ten means it's very important and one it's not important at all. How important, for you, are the messages in the commits? So, when you have something like commit history, it's supposed to be on both sides. I mean one, it's when you create the commit, another it's when you actually read some history.

Ayrton:

Ohh, cycle changes for them. Yeah. So, for me it's very dependent on the commit message itself, right? You know like often what you'll find is that your work on feature and then several years later you'll get a bug on that feature and you go to that file, you blame it. you look at the commit and if the commit message is helpful right, then it can be really useful in debugging the situation because the commit message describes the changes that were made but not just describes the changes that were made, but describes why. It also describes the issue that the person is facing on the 1st place. I don't know if you've ever seen the poor request that I write where it's like I start with linked issued this and I don't write down like the cause of the bug. I write down like what was the reason that the bug was logged, right? So, some customers contact said that they can't log in. That's the issue. What is the cause of that? Well, it's because of this particular thing in the code. What is the change that I've made? And sometimes, like you know, for maybe every 100 commit messages that I write, only one of them is useful. But when it is useful, it's really useful because I can see, like, this is the decision that I made and this is why I made this decision and not just in terms of my commit message, but other people's messages as well. There's been times when, like, I've been looking at a file and I just do not understand the changes that person has made. But then I read the commit message and they've explained why they made that change. Maybe they were wrong. Maybe they shouldn't have made that change. But the reason why they thought that was the correct change is evident from the commit message. Right? And maybe now, there's new information. So, I understand the situation better so I can make the change. Or maybe I thought I understood the situation, but they've highlighted an example of a case where this bug occurred that I have not thought of. Right? So, we have a bug on the other day, that was like we had a particular HTML element that was on top of some others. And several years ago, I just topped the Z -ndex to 9999, right? Make sure, it was absolutely top one, but now we're having this problem where that element was appearing on top of things that shouldn't appear on top. I had written in the commit message "this element should always be on the top" therefore I set it to the maximum so that it's always on the top, but since then we've introduced new UI controls where it shouldn't always be on the top. There are new controls that can appear on top of that control, so it should no longer be the top. I also wrote in the message like example how this bug can be reproduced. So, the guy who was trying to fix it, was asking me about this. Do you know like this one? Why did you set it so high? Can't you just set it to different value? But then I had described some scenarios where this didn't work, and we tested those scenarios against his change, and they also didn't work. So, they had to reapproach fix

from different angle but again, like that happens very rarely on say like once every three or four months, I need to check one of these messages.

Dawid Wozniak:

So, if it would be a number from one to 10, which one?

Ayrton:

Yeah, it would be. But this is the thing, right is dependent on what the person has written. And I would say 10 or 0, but it's very dependent on the content of the message, right? Like the commit messages are always useful if they're descriptive, so whenever it's worth the investment from the person who writes that message, they actually write something that is useful, you know, there's no point writing bad messages. It needs to either give more context to the situation, or you might as well like not put anything cause it should be self-evident from the code, right? So yeah, it's a difficult question to ask on scale of 1 to 10, like if I had to give an answer, then I would say like 8. But I would say like it's 10, if the commit message is good and it's zero if the commit message is like a one liner, that doesn't say anything .

Dawid Wozniak:

Now let's talk about the UI. So if you see this, is it clear for you what it represents that when I click on this file, it's coming history just for this file. When I click on this, it is a history for the whole folder and it’s sorted by days. Do you know why you will open it? Actually, it might be more than one coming during one day and then it's just the new line. How do you like it? Is this some helpful information here?

Ayrton:

Six. So this information is all useful. But the thing that is missing here is like the link to the actual bug, because, for example, if you're using GitHub or you're using Azure DevOps or something like that, then there is gonna be some work item associated with this commit, right? Generally, the commits come from a request or they come from a work item that is associated to that. Like maybe in the commit message, you've put like hash tag 422 and that relates to stage 422 and maybe the commit message is very light on details, but the issue itself will describe what the problem was, right? It will have links to various pull requests and comments in the discussion and things like that. Where if I'm in a position where I need to look at this history and I need to look through the messages, it's because I need to understand why a certain change was made, right? I mean, one thing is to just look at the repository just for fun, just to say, like how many files have I touched, who touched the most files and other stuff. That's nice. But in terms of actually using this at work, I'm trying to find the context for why something was done and who can help me. So, what I would really need here is to link to somewhere else where I can find more information like the commit message on its own. It doesn't give me enough. I need to be able to see the rest of the discussion. This can be done even if you can just give me the commit ID so that I can copy and paste that into Azure DevOps. I'll get there whatever I like. You don't necessarily need to write it here. Integration - where you can pull work items and link out to stuff. Just give me something that I can use as a cross reference because I can't search for this text here it's not that advanced tool.

Dawid Wozniak:

It gives me some ideas. So, the last question from one to ten. How useful would it be for your work and how you would grade the current state from one to ten? So, then I can see that there is some improvement or decrease of satisfaction when I make some changes.

Ayrton:

Well so I think if you made the changes I suggested that it would be like 10 out of 10 very useful, right? Because I have a lot of frustrations with the tools that we have today where things that I'm describing to you show me files that are commonly edited at the same time, show me people who own, not this specific file, but files in this area or files that were associated with this file and like you know, in terms of just usability things as it stands today, like the ability to click a link or showing ID for this, then it would be extremely useful. This is something that I would use at work all the time as soon as I'm faced with an unfamiliar area of the code, I would go to this tool and I would use it to figure out what I should do next. Who should I ask next? Or is there a work item that I can read or whatever, right? This would be very easy to navigate when you're in a huge legacy code base with lots of different services in the same repositories. Then, it is 10 out of 10. You said as it stands today… I would say like as it stands today like 5 out of 10, right. Because it's still good to see how things are related. But I think you're not giving me enough. Folders are not really a good measurement of how related things are, right? There's a lot of like small usability improvements that could be made like linking service, external repos, just things that I think you could make. It gives a lot of small tweaks to get a very high increase in the rate and you know if you just added the commit ID or link then then it would immediately be like 7 out of 10 instead of 5 out of 10, you know.

Dawid Wozniak:
Okay so I would I wanted to also ask one open question and then I have some statistical questions that I need to ask you. So, the one question that is still open for me, it is about the insights from commit messages. Would it be useful for you to group the same commit messages in some groups or something that would help you to understand the flow?

Ayrton:

That is not related to files or it is…

Dawid Wozniak:
It's not. So the question overall is what would be useful to do? Something else that you didn't describe yet with the commit messages.

Ayrton:

Well, I think you would struggle to find the most useful commit messages are often the ones that are very lengthy descriptions on what and why, right? I think those are the exact kind of things that are difficult to categorise. Because it's a lot of free text, it's not like a „update this file", „do this thing" you know, like anything that is like a very short few words, you probably find duplicates of the same message and things like that. You start grouping them together, but the messages that I find useful that I would want to look for and that I would want to read would be the long ones that are like very long form free text descriptions. You know, so it's less about sort of analysis of the messages but more helping me find those messages in the 1st place. That's what I would need.

Dawid Wozniak:
Yeah. So is there anything else that you would like to say about the problem except the discussion we had?

Ayrton:
No, I think I said most of what I wanted to say before you startasking them. Before the questions, right? The only thing that I would say is like you know this would be great if this was an extension that was available in DevOps, for example, like instead of the current view. You click on file and you can see either see pull requests with that file or you can see commits that sort of build a file. If there was another view that was like I can go out to this map and I can see who touched things in this area because at the moment I need to dig around. Whereas if this was a tool that was integrated into the sort of like GitHub, DevOps, those sorts of thing, this is something that I would use all the time, right? If it was like this, I think it would be less useful. If it was something that I needed to run manually and took me a long time. It needs to be something that runs in the background, and it's constantly updated every time somebody makes a change, so that when I visit it I don't need to run it on my own computer. It's just like an interface to some data that has been precalculated and is constantly updated. You know, like the idea would be like every time you complete a pull request, some job would run in the background. So update this with the diffs of what has just happened. And then if I click this, I don't need to do any computation, I'm just visualising data that is stored somewhere.

Dawid Wozniak:
OK. So thank you for your feedback. I think that's all I desired from the first meeting. So, I need to ask some questions just for the record. So how many years have you worked in the IT industry including part time jobs?

Ayrton:
Yeah. Well, so including part time jobs and internships, it's like 9 years in total, full time at Microsoft, it is 6 years.

Dawid Wozniak:

OK, so your role is…

Ayrton:

A developer. That is what we established.

Dawid Wozniak:

When you have the team, including all roles in the team, what is the average size? So developers, QAs, product owners, product managers.

Ayrton:

So at Microsoft, it's like test engineers. I think so generally like 5 or 6 full time software engineers per team and maybe two student workers, and then they'll be 1 lead of that team like engineering lead who usually focuses on sort of like making sure that the engineering work is done, and then there'll be a project manager and the project manager is the person who interface with customers and is sort of less technical, more of customer focused service, you know one lead, one project manager, six enginners, two student workers.

Dawid Wozniak:

Yeah, that's great. And when you have an average project, what is the number of commits count that it's committed per day? What is the number of files?

Ayrton:
Yeah, it depends because if it's a feature then generally like maybe four or five and it will be very large ones. That is the majority of the future because we usually make the feature as one commit and then a few small additional feature updates and then many bugs. So, feature is like four or five, but then bugs are fixed like maybe one or two a day. So, of course, in the year, we're talking like 300 commits. Some of those are bigger features, but the majority, the vast majority of commits like 95% of gonna be bug fixes and maybe 5% of the commits. The feature commits would be like vastly larger like in terms of the description that is written, the message that is written but also the number of files that touch about those features. It also depends on whether you squash the commit messages or not. If you choose to squash your branch when you measure into the main one, then you're gonna lose all the small comments. Like if you ask me how many commits, I actually make. It's like hundreds not for years, like hundreds for a feature, but maybe only four or five commits are actually committed to the master batch. The idea behind squashing stuff is that the commit message that is checked in is ultimately something that is long form, meaningful text. So, I've seen a lot of the messages that you're showing me on the screen are things like „fix this bug", „update this file", whatever. It's like, very small rapid file, like small change. We don't really operate that. It's more like we make one commit and it's huge and there's big, long description. Then only a few of those mean a lot, because things are just crashed together.

Dawid Wozniak:

Great, So if you would like to participate in the next survey, I will probably send the invite at some point. I will try to implement your feedback and some other people feedback and then we will talk more about the improvements or what change comparing to this version. Of course, I will not repeat the statistical questions at the end about your profession, your project etc. Hopefully I will be able to implement at least some of the suggestions we talked and show you what we improve to see if we are going in the right direction. So that's all. Thank you, Ayrton.

Ayrton:

Yeah, no problem.