R: So, if I'm not mistaken, you can see my screen now.
P: Yes, I can see your screen.
R: Okay. So first of all, thanks for participating with me in this study.
First I'll be explaining what I'm doing with my thesis and afterwards we'll be working on a task. So I'm doing my thesis on the concept of software product quality and specifically finding quantitative measures to objectively measure software product quality.
So there's a model, it's one of the ISO standards, ISO 25010, it's well accepted in science and it basically states that good software products follow eight quality characteristics such as reliability, maintainability, performance efficiency, etc. So during the first part of my thesis, I conduct discussion groups with engineers to find out how can we measure these abstract concepts as objectively as possible. So they came up with a few ideas and I'll be presenting these ideas and we'll enrich their ideas by adding a source and by saying something about how difficult it would be to obtain the data and how much technical expertise would be required. I think it gets a bit self-explanatory once we get into one of them. So we have two dimensions, the difficulty of obtaining data. That is low if the data is mostly already available or minimal effort is required to make the data available. Usually you can implement the measurements within one hour or a day. We have the category moderate, which is the data is possibly available but you do require some pre-processing and cleaning, usually one day to a week. And high, the data is mostly unavailable and extensive effort is required to make the data available, usually taking up more than a week. And then we have the dimension technical expertise required, which is low and you just need basic technical skills and moderate when technical skills are required. But most software developers can implement the measurement. So probably some knowledge about the tool, but no in-depth knowledge about the tools required and advanced is that really in-depth technical knowledge on a certain tool or technology is required. Are these dimensions clear for you? Is it clear what we are going to do in this session?
P: Yeah.
R: Okay, perfect. So in total there are eight and with each participant I'll discuss either one or two of the characteristics. So for today we have maintainability up for the question. And before we get started, it is important that you try to rationalize your ideas from your context, because obviously some measures, for instance, with reliability, we had the mean time between failures that's measured fairly differently in an on-premise environment versus in a serverless environment. So that's why for the level of abstraction, you can take your own context of the work you're doing at <name organisation>. So to further specify this context, can you tell me a bit about what you do at <name organisation> and also what technologies are used in your work?
P: Yeah. It feels like an interview.
R: It is.
P: So, yes. So I'm in an internal team at <name organisation> where we enable other internal teams to speed track their development and mainly focus on creating a data mesh so that teams can share data and benefit from that together.
R: What technologies are used for that?
P: The technologies we use is AWS for infrastructure, Node.js, TypeScript, GraphQL, Jest, et cetera, But that's the high level.
R: I think that's a good base for the context. So let's move on to the main body of the interview. We'll basically handle the same question 18 times because we found 18 measures for maintainability. So hopefully it doesn't get too tiring for you. And let me know if it does.
P: Okay.

R: So the standard defines maintainability as the degree of effectiveness and efficiency to which a protocol system can be. This is not even the right one. I should have changed this. But I know the right one from my head. So maintainability is the degree of effectiveness and efficiency to which a product can be maintained by the intended maintainers.
P: Okay.
R: So the first measure that someone proposed, and here again it is important that in this interview we're not talking about whether you agree or disagree with the measure.
We're just trying to enrich them because you might have different ideas about them. So the first measure proposed was the cycle time of a new feature. So the cycle time being the time when someone starts developing on a feature and the time when someone has finished developing the feature. How could we potentially measure this and where can we find these data values? So that's the first question.
P: So therefore we require a source system and that might be it might be a Gira, it might be a GitHub repo, it might be logging.It can be anything. So could you just give me an example of how I should maybe give an answer to this?
R: Yes. So we're looking at the measure, the cycle time of a new feature.
So the definition of that being the moment someone starts developing and finishes developing a new feature and we're trying to find out where can we find these points of data because we're trying to come up with a numeric value to measure this.
Right. What could we potentially find this numeric value or how could we compute this numeric value? What do we have to do?
P: Right.
R: Does that make sense a bit?
P: So to explain back what you just explained to me, you are trying to understand what metrics are used to measure the time for a feature.
R: Exactly. What systems are used? Where can we find this information? And then the next question will be how difficult is it to obtain the data? Is maybe the data already readily available in the system? Do we have to implement some code or maybe some other stuff to do this? And how much technical expertise is required to make the measurements?
Does that make sense?
P: Yes. So for sources, I would say first one is probably whatever ticket management tool you're using. For us, it's just a GitHub board. And that gives you a very high level idea of this was started at this time and it was moved to done at this time.
But it also gives you. Okay, now we might switch context a little bit. Let's assume you're using JIRA. You might also get a lot more context because I think JIRA also tracks when things got moved from one column to the other. So you can also do some measurement there because if the feature took very long to develop but was reviewed a bit too quickly, that might be an indication that maintainability isn't 100% that it could be. Next to that, I think you can also look at the commit history or the git tree as it's actually being developed. So I'll use that. Commit history or the pull request or something along those lines. But this is a very conditional thing because some people won't like to push their stuff until they're fully ready. And some people actually go back to their locally committed stuff and actually tweak the entire git tree so it makes sense and then push it one go. So this might not be a reliable source, in my opinion, but it is a source. Let me think as well. You can also… If you are actively pushing to CICD, you can also look at the runners. For example, in our context, GitHub runners that are tied to either a feature or a feature branch or a tag or whatever kind you have.
R: I think that's already a good answer.

P: If that's enough, then yeah.
R: And then the difficulty of obtaining data.
Is this data mostly already available or do we require some efforts to make it available?
P: I would say this data is pretty easy to obtain.
R: So could we classify it under low, moderate or high?
P: Yes, I would go with low. This should be available. It should be already available in systems.
R: And then for the technical expertise required, what technical skills do we need to achieve this?
P: Let me see. So I need to pick between low, moderate or high. I would say low. I think even product owners can figure out the GitHub stuff as well.
R: Okay, cool. Now, again, we're talking about a new feature, but then this time we're discussing the lead time. So the cycle time is the moment you start developing until you finish it. The lead time is a moment a feature is requested versus when it's finished.
So that's always longer than, that's always a part of the cycle time.
P: Yeah.
R: I mean, the cycle time is always a part of the lead time
P: So just to give you an example and let me know if I understood this correctly.Lead time is, let's say, your team gets a request to do feature X in January, 1st of January,and you pick it up in February and you finish it at end of February. The lead time is January till end of February?
R: That's all.
P: Understood.
R: That's entire time period. And we are looking for source. The same thing again.
P: Yeah. The same thing.
R: The sources and then it will be basically a pretty structured format from here.
So always doing the source first and then how difficult will it be to obtain it and then how much technical expertise is required.
P: Source, probably something along the lines of a roadmap or a rally. I think it's very dependent on the team, but let's take my team's context. I would say rally, so epic slash feature request whenever that is actually added.. Let me think, is it for a manager or product owner? Well, it could be that the source is similar. You're just using a ticket management system in combination with maybe the other sources you've indicated. But then it could be potentially more difficult to obtain the data. So data is definitely, I would say, moderate for.
R: Why is it moderate here? What additional steps are required to do this?
P: So I go with moderate because I think these things tend not to have the highest visibility, especially when the request is put in. Depends on what sort of system you have in place. So in my context, these usually come from higher up channels. The visibility is sometimes iffy. So let's say my team lead gets a feature request and for some reason they take two weeks to add it onto a board. And for those two weeks, some people might be in the dark about that. So maybe measuring the accurate lead time for a new feature could be a bit hard.
And a lot of these things usually happen during meetings or during conversations or discussions that aren't necessarily written down. Yeah, that makes it harder to really obtain the data.
R: I can understand. What about the technical expertise?
P: I'll say low. It's the same as for the previous one.
R: OK. Then now we have the same to measure measures again. But instead of a new feature, we're looking at a bug fix.

R: Would it be any different? What additional information do we need to find this out for a bug fix?
P: Right. So I would say source would be the same.
R: Yes.
P: The source for that would be there. Maybe, maybe a higher or like a different point of view on how you can get a source is maybe an external source. Another team flagging the bug. But I'm not sure that that ties into that. No, that doesn't tie into it. I'll go with this.
Let's see. Difficulty of obtaining data. Difficulty in obtaining data on just how long the cycle time of the bug fix took.
R: Right.
P: Yeah, exactly. I would say low because.
R: You're saying it's exactly the same as for the feature.
P: Yeah. Kind of.
R: Then the only question left for me is how do we distinguish between a feature and a bug fix? So how do we distinguish between a bug fix and a feature? Yeah, because when we're measuring this, we need to know whether to fit the measurement under the cycle time of a feature or under the cycle time of a bug fix.
P: Right.
R: Is this inherent to the source or do we require any more information?
P: Hmm. Maybe more information is required, but I would say a feature naturally should be a little longer.Yeah. Just the duration. It's harder. It's a lot harder to say because these are all measurements of time. So duration is all I really have to work with. Maybe also bug fixes would have shorter lead times. Based on the found measurement, you can naturally inherit. Is this something that makes sense within the context of being a feature or a bug fix?
R: Yeah. So if I'm not mistaken, this will be exactly the same as for the lead time of a new feature, lead time of a new bug fix. We say there are no other sources of difficulty obtaining data and technical expertise required.
P: Yeah.
R: Then the next measurement found, someone proposed that you could measure maintainability by looking at your code coverage. So how much of your code base is covered by tests? How do we get this number is basically the question. How difficult is it to obtain the number?
P: Yeah. So we can obtain this number by running coverage, test coverage tools, or those usually come from the test library itself. Next to that, you also have additional tools that analyze your code that you're writing. So think of some for our use case that give you an idea of just your coverage as well and can enforce coverage. But the test tools can also enforce coverage.Those would be it, I think.
R: And let's say you do not have this number available.
How difficult would it be to make it available and how much technical expertise is required to do that? As in you don't have the coverage?
P: Can you provide more context on that scenario? What am I working with, for example?
R: It's within the context of your own work.
P: Okay. So I have a project. I assume I have zero coverage. I don't know what the coverage is on it.
R: Exactly. You don't know what the coverage is. You want to implement some measurements so you can find out what the coverage is. How difficult would it be to obtain this number and how much technical expertise is required

P: Right. I would say moderate because you need to just be aware of both the tools you're using and what your code looks like because sometimes there are definitely pieces of code you just don't want to cover or are excluded for whatever reason. So you do need a lot of context on the code itself. That makes it moderate instead of just low.
R: Okay.
P: So that's for how difficult it would be to obtain the data and then for the technical expertise required to make this measurement I would say and also just very basic understanding of how the tools that collect these coverages work.
R: If only a very basic technical knowledge is required, does it mean that we could fit this under low?
P: Yes.
R: So we've done five so far. Is the way of working clear?
P: Yes. Sorry. My brain's kind of mushed today.
R: You're doing very well. So the next measurement is the percentage of intended maintainers who can maintain. So if not everyone who works in your team or who is intended to maintain your repository can maintain, you probably have lower maintainability. So what numbers could we find to do this?
P: So this one is basically just let's say half your team can actually maintain a project and the other half is not responsible for it.
R: Yeah.
P: Did I interpret that correctly?
R: Yeah.
P: If someone is not responsible for it, they're probably not an intended maintainer.
It's more a question. Someone is intended to maintain the code base, but the code is so not maintainable or it's such a mess that someone cannot actively contribute to it.
R: So how do we figure out if someone can or someone cannot? Yes.
P: Maybe lead time for maintainability. So how long does each author take to maintain a similar context code, a similar context problem? The other one is looking at probably the contributor's graph on GitHub or whatever management tool. you use just to see who is actually contributing consistently and probably has context.
R: So to get these numbers, is that a difficult thing to do and is there technical expertise required?
P: A little, I would say. It's not too difficult.
R: For both of them?
P: Yeah.
R: And then I would say that we could move on to the next one.
P: All right.
R: So when making a change, here someone said if you have to change more parameters, it's probably less maintainable.
P: Maybe that does give you some flexibility?
R: Yeah. So this number of config parameters, where could we possibly find this?
P: The environment variables, the CICD workflow variables, and the...
R: So just looking at the code base, so again, looking at the code in the repo, right?
P: Yeah. I don't want to say low, but moderate doesn't seem right either. I don't think it's too hard to understand just how much configuration or config parameters are needed in a code base, but sometimes you do need to be aware of what's happening in
the code base. So let's go with moderate because we do need to invest some time.
Yeah.

You need to invest some time to find out what parameters exactly need to be modified.
R: Yeah.
P: And for technical expertise, I would also go with moderate. You need to understand what happens in the workflows and what is being used.
R: Okay. Straightforward, straightforward so far. So the next one proposed is someone said that you could measure the time to end of life for your tool stack or the technologies that you're using because technologies that are close to the end of life are probably less maintainable, is what they argued.
P: Okay.
R: So the argument is maintainability is harder or maintainability depends on just how much longer your tool has till end of life.
P: Exactly.
R: So where can we find, how can we find out what the time to end of life is?
You could do it, so there's a proactive way where you yourself go ahead and just check out whatever tool you're using or whatever platform you're basing this off of and seeing if are you using the <inaudible> version or are you using a version that is nearing end of its life.There's also a reactive way to do this where you most well maintained should start giving you warnings that your version is coming to end of life and they will push you to upgrade. So the source is definitely during coding or during development. I'm not sure how you'd want to work that. And the other source would be just a proactive approach of checking against the current tool. And checking against because we're specifically looking for the place to find the time to end of life. So let's say you're in the Node.js version, I don't know. One of the really early versions. You would ideally be aware or if you're proactive, you just go look and see what the latest version is that support the long term. Just the tool documentation that is there?
R: And to obtain this end of life, is that usually available for a certain tool?
P: Yes.
R: So that means this would be low.
P: Yeah.
R: And to figure out the end of life, do you require how much technical expertise is required?
P: Low. It's very simple.
R: Is it just the reading of the value if I understand correctly?
P: Yeah.
R: So the next one someone proposed, we have some more here, is the cyclomatic complexity. With the cyclomatic complexity being the number of paths through your software system. So if you have an if-else condition, there will be two paths through your software system. Obviously in a bigger code base, there will be a lot of more paths. And someone says if there's more cyclomatic complexity, your code will be more difficult to maintain. So this number for cyclomatic complexity, how could we potentially figure that out? That is the question.
P: I don't know. But I'm sure there are external tools that can analyze your code for you. Well, if you're not sure, we can also just add a student other play code. I'll just go with tools because I know Sonar does some things like this where it identifies overly complex code examples, especially if you're doing too many ifs or whatever, that can be something. So I'll just go with external tools.
R: Cool.
P: This is, I would say, moderate because it just does take some configuration.

You really need to know what you're doing to set up these tools. You really need some knowledge on the tool to be able to implement this. And technical expertise, I would say the same.
R: Moving on to the next one being language popularity. How could we measure the popularity of the language? Where can we find this information?
P: So measurement is probably taking a look at the number of open issues in the languages. GitHub, for example. The other one is looking at the community size and the third one is probably I don't know the third one you want to add, but third one is just the availability of documentation and support from the community.
R: Where could you find this information, all these three things?
P: So first, GitHub, second, I would say GitHub slash blogging website slash community driven tech websites. And of course, the community's own website. So the community's own Discord or Slack channels, et cetera. And the availability of documentation is just that's a bit harder to understand, but it's definitely on the GitHub slash the tools website. So same answer. The obtainability of this data. This is definitely something that you need to spend some time on, especially the understanding the documentation, because you don't know what isn't there until you run into it. So I would say this one is definitely not just moderate, but maybe the one above, maybe expert or harder. Because it's mainly the documentation. You don't know what you're missing until you run into it and you just find nothing.
And that takes a lot of time and you actually need to build something that needs to use whatever that you're looking for. That doesn't exist.
R: Yeah.
P: So this might be tailored more towards the documentation slash just, are there enough questions and answers to give it to you on a silver platter or do you need to go into the source code yourself? And if, and if that happens that immediately, you would argue that a language is more popular if there's more availability of documentation. That's a difficult one to find out. Technically, technical expertise required. I would lower that to a moderate.
I feel like if you're already using the language, there should be some overlap that you can just go into the source code and try and understand things yourself.
R: The next up, someone said that your code is better maintainable if more functions are documented. So how many of your functions are documented as opposed to total number of functions? Where can we find these points of data?
P: You can look at your generated documentation.If you generate documentation from a JS stock, for example. You could also, sources, sources.
R: How did you describe it?
P: Just your generated docs. Yeah Okay. And then to get to this number, how difficult would that be? I would say low.
R: Okay. Why?
P: Wait. Actually, you know, this would be probably moderate. Because if you generate docs. Yeah.And let's say you understand that there's a hundred documentations for a hundred functions, but you still need to know a lot more context of just how many functions are there in the actual code base. Because if there's a thousand functions and you only have a hundred generated, you still need to do that leg work to understand the difference between the two. So there's a lot more effort in that. For the technical expertise required, I would go with low.
R: Because?
P: It's the generation itself is straightforward. It's mainly obtaining and comparison and understanding what's your actual threshold versus what you have.

R: That makes sense. So then we have one, two, three, four, five, six, seven more to do. You're still doing okay?
P: Yeah.
R: Perfect. So the next one is the percentage of that code with the death code being a code, which results are not used anywhere.
P: This one is you can probably identify that code by linters. Yeah.
Whatever. I'm just mixing things up. But your code linter or your code editor should give you an idea that this function is not being used or this piece of code is either unreachable or just never called. Yeah. Difficulty of obtaining this data, I would say low.
Most modern IDEs do a good enough job that you don't even need linters or whatever.
Yeah. I would also add IDE into sources to be completely honest with you. But I'm not sure if you want to do that. Technical expertise required low as well. Very straightforward.
Most editors will also just gray out things that are never reached or never used.
R: So the next one someone proposed was the time to first production deployment.
So when someone is starting you in a team. Yeah. Someone's saying let's give them the most simple piece of work. How long does it take them to put this piece of work into production? If the code is maintainable, it will probably be a short amount of time.
If the code is less maintainable, it will probably take them a bit longer.
P: Sources for that probably be the ticket management system. And the GitHub repo
Difficulty of obtaining data very low and expertise low. It's just a time measurement thing.
It's very straightforward.
R: That does make sense. So next up we have the number of domain knowledge holders still available. Yeah. So a way to measure this is looking at the contributors on GitHub.
P: Understanding the percentages. That definitely correlates to who has more domain knowledge. Yeah. And probably the ticket management system again.
If there's a filter of who picks up things related to a certain epic, you might see a pattern there as well. Yeah. Okay.
R: So to get these numbers of contributors and get these assignees on the ticket management system, how difficult would it be to obtain the data? And how much technical expertise is required?
P: Low and low. They're all the metrics you can filter on. They're all metrics that are readily available and already found in the GUI.
R: So next up we have the number of functions divided by the number of classes.
Someone proposed that your code becomes more maintainable if you encapsulate your functions inside their own classes.
P: So separation of concern of sorts.
R: Exactly.
P: Do you think along with me for the source?
R: I can, but I cannot give you any answers, unfortunately, because as a researcher I cannot.
P: You can think of, you can split them up.
R: Like if you want to, if you're working on a project and I'm asking you <name interviewee>, how many functions and how many classes are in your project? And you're saying to me, well, I don't have the time to count for you right now. But look at this piece of information. What would be the piece of information that you're referring me to?
P: The code. Exactly. Could be. So code, but what I would say is either A, just how isolated each component is. So does it do only one thing? Is it similar to like pure functions, but on a class basis where let's say all things, animal, all functions related to an animal are in the

animal class? Readability. I think these two kind of tie together, being able to have separation of concern improves readability and understandability.
But code base would probably be the place you want to go to understand this metric. Difficulty of obtaining data. This one needs a bit more legwork, so moderate, but this is just purely a time thing. You just need some more time than low. Yeah, because it's a lot of things you need to count. But I'm sure you can script it or something. Technical expertise required. I would say low. I think you can get a product person to do this in their free time if you just tell them what a function looks like and what a class looks like.
R:Okay, yeah, that makes good sense. Okay, so three more. Number one being the number of bad code smells in your code. Sources.
P: External tools, for example, does that. I know some IDEs also have plugins to do that as well for you.So those are nice ways to identify code smells.
R: How difficult are they to obtain?
P: I would say low. Because there's very much a standard to this to identify code smells, especially with these tools and plugins. Expertise required, moderate, because you need to really understand why it's telling you there's a code smell. Exactly, yeah. To figure out whether you agree and could kind of figure the number. And what to do next, yeah.
R: So for the second to last one, someone proposed that you could count the number of deployments given in a certain time frame. Because if code is maintainable, people will be more likely to be able to bring something to production quicker.
P: Yeah, this is a difficult one because many, many people argue that. There's always people on either end of the side. I would say number of deployments per X period of time.
So number of deployments per month, number of deployments per week or per sprint, some sort of metric. Measurement metric would be a way to get this. You could probably get this through. You can get this from GitHub, your deployments or releases. It's too variant to really pinpoint something. But I would just say GitHub because you have a lot of workflows there to deploy things. Difficulty, I would say low. This just depends on how it is, but low as well.
R: Because you could just read it from somewhere?
P: Yeah. Well, this is so dependent on your input.
But for my context or <team name> context.
R: So we're dealing with the context you're working on.
P: It's very easy to understand. Because we do GitHub releases as well.
R: And then for the final one that the focus group came up with is the experience level of the maintainers. And they argued that when someone is more experienced, they're more likely to be able to maintain the code quicker. So source here is how do you measure maintainers?
P: Exactly.
R: How do you measure experience level maintainers? Can that information be found somewhere? Maybe it's tacit knowledge as well. It's not described anywhere.
P: I'm not sure how you would extract that. Yeah. Could be not available. So you can't really do this.
R: That's an answer as well.
P: Just one more question. Am I looking at this? I'm looking at this with my own team's context. But am I looking at this with someone that can? Should I be able to approach the maintainers to ask them questions or?
R: Yes.
P: Oh, okay. Surveys, questionnaires, or maybe team meetings.
So understand what the maintainer's background looks like.

And with certain questions, you can definitely understand who's more senior, who's more junior, et cetera. So I would say a team survey or a team meeting or a team session. This would be low to obtain. And expertise required low, I think. I think this is the technical expertise here is self-explanatory because it doesn't have any technical aspects.
R: Okay. Those were all. So I can stop sharing my screen now. And I'll stop recording. Thank you so much for participating.
P: No worries.