**Summarized codable data:**

**Theme 1: Technical Debt Identification(Not Correct)**

- Q4: Stages in prototype dev: Req. gathering, design, impl., testing. Challenges: tool compatibility, standards, language choice.

- Q5: ID technical debt: Code reviews, analysis tools, feedback, periodic assessments.

- Q6: Indicators of tech. debt: Complex code, poor metrics, delayed releases, increased bugs.

**Theme 2: Technical Debt Measurement(Not correct responses)**

- Q7: Measure tech gaps: Code analysis tools, metrics (duplication, complexity).

- Q8: Current tools: SonarQube, metrics (maintainability, coverage, defects).

- Q9: Prioritize gaps: Impact on reliability, performance, core functionality.

**Theme 3: Technical Debt Impact Evaluation**

- Q10: Impact of gaps: Reduced reliability, slower performance, lower maintainability.

- Q11: Examples: More defects, longer fixes, delays, lower user satisfaction.

**Theme 4: Early Debt Repayment**

- Q12: Practices: Continuous integration, code reviews, sprint refactoring.

- Q13: Motivation: Recognition, sprint allocation for debt reduction.

**Insight:**

- Q14: Balance feature delivery and debt mgmt. Continuous vigilance and adaptation are vital.

**Full Transcript:**

Thank you for taking your time and being part of this. My name, once again, is Mugoya Dihfahsih. I'm doing a master's degree in software engineering from Makerere University, and I'm carrying out research in software architecture where I'm looking at student projects, especially the architecture of the software. The main goal of this research is to come up with a framework, with a set of guidelines to enable students to minimize their technical gaps or their bugs or errors in their projects as soon as possible.

So, have you ever heard about technical debt before?

No. Like a debt, you know when you go to a bank and you borrow money, it becomes a debt, right?

Yeah. So, technical debt basically is understood as the consequences of taking shortcuts or making sub-optimal solutions or compromising the software during development. Just like a financial debt, it accumulates interest over time, making it harder to maintain and update the software in the future. It also represents the extra effort that you normally take when implementing this software.

It's due to improper design, documentation of your code, maybe poor testing, poor team collaboration, all those causes of technical debt. Not forgetting, taking shortcuts while developing the product. So, it can also slow down software development. It may introduce bugs into your software. Technical debt may also require extra effort when fixing these errors. So, basically, that's what technical debt is.

In a language, technical debt is sub-optimal solutions. These are bad practices of software development. Instead of using correct development practices, such as code documentation, maybe writing short code lines, maybe writing modernized code. You don't need to write long lines of code. Maybe you use outdated packages. Maybe you use code that you don't understand. You've copied code from ChatGPT and used it in your project, and you don't understand what it is. All those are causes of technical debt.

So, I hope you've understood what technical debt is. I mean the debt with d-e-b-t.

Yeah, I've understood.

You've understood. Thank you so much.

So, in this research, we are going to look at basically four themes. The first one is going to be technical debt identification. The next one is going to be technical debt measurement. Thirdly, we shall look at technical debt impact evaluation. Then we shall look at technical debt repayment.

Yeah, maybe you can start by introducing yourself, telling me your name, your course, the year of study, and the project you've worked on, like the role you've played on the project. Which role have you played on any project you've worked on?

So, my name is Muganda Charles, and I am doing a Bachelor's degree in Computer Science from Uganda Christian University. I'm in Year 3, Semester 1. For the projects, I've worked on both software and hardware. For hardware, I've been doing embedded systems. For software, I've been doing web programming, for mobile and web. And for the role, I've been doing back-end and some small front-end development, basically. Then for my projects, I'm having one which is running currently. It's in software, and it's a data science project.

Okay, so is it part of your final project?

No, it's not part of my final project.

Okay, so you've not started on a final project yet?

Not yet. But you're about to, right?

Yeah, about.

Okay, so in your experience, what are some of the stages that you involve yourself in while developing software?

From my experience, I think when I'm developing software, first of all, I have to know the purpose. Then I have to have a target and know how it will be efficient for my target. Target, I mean the end users. Yes. So that's what I first think. Secondly, I also have to know which platform am I going to use. Is it USB? Yes.

Okay, what are some of the key challenges that you normally encounter while using the tools, standards, the frameworks, and the programming languages that you normally use?

The challenges I get, okay, let me put this in, like any developer, it is really very hard to first of all go through the documentation. So first of all, the basics, I go with that. Then, when you find challenges, that's when I will run back through the documentation, that comes with a step over flow, looking for solutions. Yeah, so as in we don't have that starting point, we don't come when we are totally ready, but we repeat the process, that's when we start now running through.

Okay, so maybe the starting point is always the challenging one.

Yeah.

Okay, so, but the documentation I'm talking about is the documenting of your code that maybe I've implemented this feature, and this is what it's doing, this is what it's lacking, and this is the way forward of it. That's the documentation maybe I'm talking about of overcoming technical debt in your project.

Oh, yeah, not the documentation of maybe the languages, maybe not the documentation of how to use code, something like that.

Hmm. Yeah, the documentation, I mean, yeah, yeah, yeah. Can I ask a question?

Yes.

Yeah, so are you trying to mean that, okay, a kind of documentation, where maybe you are going to get collaborators, that they may understand what's currently going on, and it may be easy for them to add in something. Basically, like documenting your code, like maybe say, like you said, you have a software, yes, and each feature that you are working on, do you see like writing comments in your code?

Yeah. That's part of documentation, but there has to be a clear one, that maybe I'm done with the login, the login has these features, it has this, and this, this is lacking, maybe you know when sometime when you may be working as a team, sometime maybe you fall sick, and someone has to start from where you stopped. If you don't have a documentation that details what you're doing, they can't proceed from where you stopped. So technically, that becomes a technical debt, because they can't understand your code.

Hello, Charles?

Charles, are you there?

Yeah, so I was saying, yeah, that I've learned that, because I didn't think about it earlier, because I've been working on, okay, currently the platform I've been using, Django, but I've been, I hadn't tried out that yet, because for me, I knew what I was doing, I know this is this page, it is done, but not necessarily documenting it in the way you are trying to explain it. Yeah, basically, maybe I never introduced myself well, me too, I'm a Python developer, and basically a web developer, I would have been developing web applications, I think for the last four years, and I've been basically using Django, so when you talk about Django, and maybe you are trying to mean the documentation of Django, that one you will never understand it, it's really complicated, so I know what you mean, yeah, documentation of these tools is always high level, it's always hard to understand them, but

when you are using, when you are implementing your own code, it's a bit easier to understand what's happening. But that being said, do you think there are gaps in your current project?

Yeah, there are gaps, yeah, like when you are getting the data from the, okay, my project is on SMS, where I'm trying to get data from SMS, analyze it, then maybe visualize it. So, some of the gaps come when you're trying to get data from the, not the website, but the data providers, you're trying to get data from them. Sometimes they send you data which you don't even understand, sometimes they send you data late, sometimes they send you data when it's too late, then also another challenge comes in when maybe you are not documented code and you fall sick, and your workmates are not able to understand, those become the gaps, the code that is not properly documented becomes a gap.

Okay, I see, so for you, how do you normally identify technical gaps in your project?

In my project, the way we identify is through testing, so you test your project and see which points did I get wrong, which points did I not complete. Second, is through the team members, because normally we work in a team of five, so you get a colleague who is free, and he goes through your work, and he tells you, this point, I think you should have maybe done this, or he tells you, this point is lacking this and this, maybe you should do this, then also other ways of checking, maybe you can check your own code, because you can just run the code. You know when you're testing, you just test your code, so you can just run the code.

Yeah, yeah. Maybe also you can use maybe some tools like VS Code. Normally, VS Code has extensions, and I'm sure you have seen that when you're writing, let me say, writing a for loop, if you miss the colon, normally it will always highlight it for you, like there's an error there.

Yeah, yeah, that's true.

Yeah.

That's one way of knowing that there's an error, another way of maybe identifying that there's a gap in your code.

Then another way also I have, like also, there are some packages, you know, when you're trying to install a package, like the one that is going to help you with front-end. It's not, maybe it's too old, it's already deprecated, so you get that notification in your terminal.

Yeah.

So those are some of the ways, okay, I get you, that's good, that's good.

Yeah.

Okay, that's good, that's good. So, have you ever, or have you ever tried to measure the amount of technical debt in your project?

Maybe measuring the debt as a whole, I've not tried.

You've not tried?

Yeah, I've not tried to measure it.

Okay, so maybe you use these other ways of identifying that there's a gap, but you don't really try to measure it, to see how bad it is, or to see how much of it is in your project. You don't do that?

No, no, no.

You don't do that?

Okay, thank you so much. Do you know of any tools that you can use for measuring your technical debt?

I think the tools I've been using are tools like VS Code, they're enough for me. Yeah, so I've not known any tool that maybe I can use to measure.

Yeah, yeah. Like the VS Code that you've been using, it has extensions, right?

Yeah, yeah.

Maybe you can say you've seen a linter in the VS Code, so I think a linter is also one of those tools that you can use to measure the amount of technical debt in your project. I don't know whether you've come across such tools.

I have not seen, let me see, I've not seen that linter. I have not seen, let me see.

Okay, I'll show you how it looks like.

Okay, so maybe another question, like I've asked, so, now that you've identified the technical debt, now that you know maybe there are gaps in your project, how do you go about addressing those gaps?

So, when I have identified the gap, basically I sit down, then I start by maybe removing the points which are basically giving me errors, because you know, first of all, when I'm developing my code, I like testing it, because when I'm testing it, I can tell if the project is running. So, first of all, when I identify that point, it's not really performing well, so I remove it, then I write a new one. So, I don't actually always correct that point, because when I correct it, I always run into errors, so it's better if I remove it and write a new code from scratch, because that code is not working. So, normally I like testing, because when I'm testing, I'm able to see which points are not working. Maybe it's missing an argument, or maybe it's missing a package, maybe the package is not installed. So, I'm able to see which points are not working.

Okay, so for you, you like working in that way, that is good. Yeah, that is good. So, but the research I'm conducting, we are trying to see if maybe there's a certain way that maybe can be introduced into the software development, like some practices that you can have to minimize your technical debt as early as possible, that's why I'm trying to collect this information. But it's nice to hear that you've been using some kind of methods for minimizing the technical debt. But we have other people who have been saying that they don't use any method for minimizing the technical debt. So, I can see maybe for you, when you see that there's a gap, the first thing you do is to test. If it is not working, then you remove and start afresh, which is a good thing to do.

Yeah, that is the way I see it.

You don't want to start maybe editing that code and try to make it work. You just start from scratch. That is really nice. That is really nice. So, do you have, do you think there are practices that students can apply to minimize technical debt in their projects?

Yeah, the practice is like, one of them is collaboration, like teaming up, because like it's teamwork, teamwork is really nice. Like you say, you can't run away from testing, because maybe when you're running, like for example, when you're running like your project, you need maybe data. So, you know where to get data. Maybe you get your data from a website. So, you need that website to run your project. So, like collaboration, it is really the key. Then, I think one of the practices that maybe it can help students is first to understand their code, because when you're working on a project, for example, when I was developing a web application for a certain company, I was not understanding the code that I was writing. You know, you can't

like, the day you'll get a bug, you'll find you can't do anything about it, because the code is not yours. You were copying from this website, and pasting there, and pasting here, and running, and it worked. Then, you take it to the company, and the company says it's not working, then you see you're in a fix. You can't help because you don't understand what you wrote. So, understanding the code is really the key.

Yeah, yeah. So, when you are copying and pasting code, so, what you are trying to mean, like maybe you are just looking for a solution, and then you find maybe a code snippet, then you copy and paste it into your project, and you don't even understand what that code snippet is doing.

Yeah, like, for example, when you're developing a web application, maybe you're trying to add maybe a form, where the user is going to be filling in, and maybe you're trying to change the way that the form is going to look like. So, you're trying to search for a way, you're searching for a solution on how that form is going to look like. You've never done that before. So, you see, maybe the way you've seen it is that the way you've seen it, you can't be able to do that. Maybe you've seen a different way of implementing it. So, you see, okay, let me try this one. So, you see that code snippet there, you copy it, you paste it. You're not caring about where it's coming from, who wrote it, who's the author, you don't care about that. What you care about is it's working in your project. That's what you care about. So, you copy and paste. So, the day you'll get a problem, you're trying to change the same code that you pasted. You don't even understand what it's doing. You'll find it's very hard. So, that's really the challenge we have in students.

So, for you, you think copying and pasting, maybe, because it's like, it's like not your code. You don't understand it. It's just something that you've gotten from the internet, and then you've pasted it in your project. So, you think that's a practice that students should stop doing?

Yeah, they should stop doing it, yeah.

They should stop doing that. They should stop doing that. I see, I see. So, that's one of the practices that you think that they should stop doing. I think also it is good to maybe, when you're doing that, try to understand what the code is doing, like go through it and see if maybe you can be able to understand it.

So, for you, you are saying that understanding the code is the key thing. That's really nice, that's really nice. So, maybe another question, maybe that can also help in your research. So, how do you think students can be motivated to adopt best practices in software development, like practices for minimizing technical debt?

I think students can be motivated maybe by taking, having these experiences of the challenges that you get when you copy and paste, when you don't understand your code, when your code is not running, or it's running very slowly. So, basically, experience is the best teacher. So, they have to go through that experience of, they have to go through those challenges of understanding the code.

I see.

And like, as of now, basically, I'm self-employed. I work from the house. I don't have any, any money for the organization. It's not paying me anything. Yeah, so I can't really say, I can't really say because I have, because I have not, I have not experienced that part of employment. But in the company I was working in, you know, when you're working in a company, and then you're, maybe when you're working, the company is, I mean, your boss is paying you every day, and the code is not working, and they're paying you every day. You see, they can tell you, maybe you just go to the, the employer, the employer can just, I mean, terminate the contract because you can't deliver. So, you see, it's a very big challenge. So, you see, you can't run away from it.

So, for you, you think maybe students can be motivated by going through these experiences, maybe facing these challenges, and then, once they face them, they will learn from them, and then they will be motivated to adopt best practices.

Yeah, that's one way, yes.

That's one way, yes. But I think maybe there can be other ways, like maybe mentors, like maybe lecturers, people who can mentor them, who can tell them the best practices. Maybe there can be also some kind of reward system, like if you're able to adopt best practices in software development, then you are rewarded in some way, like maybe you're given a prize or something like that. That's also a way of motivating. But what you've said is also correct.

Yeah, it's the best.

It's the best.

It's the best way, because you see, when you're motivated, I mean, when you're, like for example, you get a reward, maybe you're given money for, I mean, developing a software and it's running very well. So, the next time, you'll not be motivated by developing a very, like, very nice, very, very nice software, because you know, okay, if I'm going to develop the software, it's going to be good. So, when I do it, I'm going to get money. So, that is what, I think it's not the best, because at least when you get that experience, when you're doing it the hard way, you're getting, it's in your mind, it's in your heart that if I don't, if I don't do this, then this, this, this, this will happen. So, you see, you'll always be motivated. You'll always be, I mean, having in mind that I need to do my best. So, I think it's the best way.

Yeah, it's the best way. It's the best way, yes. Yeah, thank you so much, Charles. So, I think those are all the questions I had for you. Thank you so much for your time, and thank you so much for taking part in this research. Thank you so much.

Yeah, welcome, welcome.

Thank you, Charles. I'll also share with you my contact so that maybe you can be able to contact me when you have more information or more things to share.

Okay, that's really nice. That's really nice. Thank you so much, Charles.

Thank you. Have a nice day.

You too, Charles. Bye.

Goodbye.