**Codeable Data**

**Theme 1: Technical Debt Identification**

4. In your experience, what are the stages involved in developing software prototypes and what are the key challenges you have encountered related to tools, standards, frameworks, programming languages, and conventions?

- Stages: Research, component study, and coding.

- Challenges: Tool limitations, high costs, complex code sourcing, and variations in search methods.

5. How do you identify or become aware of technical debt in your project?

- Identify through requirement changes and project lead input.

- Indicators: Slow loading, responsiveness issues, complex code, ownership problems, and lack of documentation.

**Theme 2: Technical Debt Measurement**

7. How would you like to measure technical gaps in your processes or product?

- Measure based on project requirements and feature dependencies.

- Error quantification via linters and debugging.

8. Which are the current tools if any or measurements you would use to measure technical debt?

- Manual debugging, limited use of automated tools like SonarQube.

9. How would you prioritize which technical gaps to address first?

- Address on a "first come, first served" basis to avoid issues piling up.

**Theme 3: Technical Debt Impact Evaluation**

10. How does un-resolved software gaps affect the quality attributes of your software prototype (e.g., reliability, performance, maintainability)?

- Impacts: Slow loading, data fetching issues, and responsiveness.

- Affects reliability, performance, user experience, and functionality.

11. Can you provide specific examples of how technical gap has affected the project outcomes or the end-user experience?

- Example: Project failure to load user profiles required extensive rework, affecting project outcomes negatively.

**Theme 4: Early Debt Repayment**

12. Are there any practices or strategies in place to encourage early repayment or fixing of technical gaps during the software prototype development process?

- Practices: Structured searching, setting repayment goals, code reviews, and documentation.

13. What incentives or mechanisms exist to motivate the team to actively manage and reduce these gaps?

- Incentives: Meeting deadlines, ensuring functionality, and providing a better user experience.

**Insight:**

14. Is there anything else you would like to add or any additional insights you would like to share regarding technical debt in software prototype development? If yes, please share with me.

- Suggested providing advice, blogs, and structured development practices to minimize technical debt.

- Emphasized maintaining a clear project structure for easier debugging and problem-solving.

**Summarized Transcript**

**Theme 1: Technical Debt Identification**

4. In your experience, what are the stages involved in developing software prototypes and what are the key challenges you have encountered related to tools, standards, frameworks, programming languages, and conventions?

Conrad mentioned the stages in software development, which include research, seeking inspiration, studying the flow of the application, researching different components, and dealing with CSS. Challenges included limitations and costs of tools, difficulties in sourcing components, and variations in how people search for solutions.

5. How do you identify or become aware of technical debt in your project?

Conrad identified technical debt through changes in requirements that necessitated code changes and by observing red flags such as slow loading times, responsiveness issues across different browsers, complex or duplicated code, and poor documentation.

6. What are the indicators/red flags that suggest that there is technical debt in your processes or product?

Indicators of technical debt mentioned by Conrad included slow loading times, differences in performance across browsers, complex or duplicated code, ownership issues, and poor documentation.

**Theme 2: Technical Debt Measurement**

7. How would you like to measure technical gaps in your processes or product?

Conrad proposed measuring technical debt based on project requirements, features that require extra effort or payment, and error quantification through tools like linters and debugging.

8. Which are the current tools if any or measurements you would use to measure technical debt?

Conrad mentioned that he mainly uses manual methods for debugging, relying on tools like linters and manual code inspection. He noted that there are automated tools available such as SonarQube and others, but he hasn't utilized them extensively.

9. How would you prioritize which technical gaps to address first?

Conrad prefers addressing issues on a "first come, first served" basis. He emphasized the importance of fixing errors as soon as they are identified to avoid forgetting their causes.

**Theme 3: Technical Debt Impact Evaluation**

10. How does un-resolved software gaps affect the quality attributes of your software prototype (e.g., reliability, performance, maintainability)?

Conrad explained that unresolved technical gaps can lead to slow loading times, difficulties in fetching data, and issues with responsiveness. He also mentioned how it impacts user experience and functionality.

11. Can you provide specific examples of how technical gap has affected the project outcomes or the end-user experience?

Conrad provided an example where a project failed to load user profiles due to code issues, which required redoing a significant portion of the work, illustrating the impact of technical debt on project outcomes.

**Theme 4: Early Debt Repayment**

12. Are there any practices or strategies in place to encourage early repayment or fixing of technical gaps during the software prototype development process?

Conrad emphasized the importance of structured searching for solutions and adhering to best practices. He mentioned that setting repayment goals during projects and conducting code reviews, along with proper documentation, are beneficial practices.

13. What incentives or mechanisms exist to motivate the team to actively manage and reduce these gaps?

Conrad discussed the motivation for early resolution, including meeting deadlines, avoiding functionality issues, and providing a good user experience. He also mentioned that some issues are interdependent, so addressing them early is crucial.

**Insight:**

14. Is there anything else you would like to add or any additional insights you would like to share regarding technical debt in software prototype development? If yes, please share with me.

Conrad suggested providing advice or blogs from experts on managing technical debt, focusing on project management and structured development practices to help students minimize technical debt. He also highlighted the importance of maintaining a clear project structure to ease debugging and problem-solving.

**Full Transcript**

Hello, Conrad, good afternoon. Yeah, I can hear you. Yeah, so let me start by introducing myself. My name is Mugoya Dihfahsih. I'm doing a master's degree in software engineering at Makerere University. And I'm carrying out research currently in software architecture by young teams, especially students doing their final year projects. And the main objective of this study is to identify metrics for measuring technical debt in software prototypes.

When I talk about maybe to introduce technical debt, technical debt basically is understood as consequences of taking shortcuts or making compromises during software development that may result into a product that is not easy to maintain, that requires extra effort to add maybe features to it, due to poor documentation, poor design, maybe never tested. And it slows down the development through the introduction of maybe bugs and requiring you an additional effort of fixing these bugs.

Basically, we are coming up with a framework or a set of guidelines that students that are young teams or that are novices in programming to be able to write projects that follow a standard such that we minimize technical gaps in their products or prototypes as soon as early as possible. The main emphasis of this research is to have a look into how students can minimize these bugs, these errors in their prototype as early as possible in their software development lifecycle.

So maybe you could tell me your name, the course you're doing, the year of study, and then the role you're playing on the current project you're working on. Yeah, I can hear you but the voice is a bit low. You don't know if you can hear me now. Yeah, I can hear you now. So I'm Mario Conrad-William, Computer Science, year three. What else would you love to know? Yeah, the role you're playing on the current projects. Oh, I'm a UI UX designer, stroke front end developer. All right. Project that you're doing. Yeah. Which kind of project are you working? Is it a web application, desktop, or IoT, something like that? It's a web application.

Okay. So have you had a technical debt before? Technical? Technical debt before? No. Maybe I may know, like, in case you'd like to explain. Yeah, like I explained, you know, a financial debt, a financial debt when you go to a bank and you borrow money, you always pay the interest in the form of maybe monthly something like that. But as soon as you pay the principal and all the amount, the least interest you pay.

Let me say you borrowed money today, and you are supposed to pay it in five years and you pay it in one year, you're not going to be the same person who has paid through five years. So the same applies to technical debt when take shortcuts in developing software products. Let me say you get code from the internet, you just use it without understanding, you will have to pay at some point, maybe you're adding a feature to the product. You don't know what that code was doing. So you need to understand what it was doing.

So that time that you're spending, it's more of a technical debt. You imagine if you're employed somewhere in an organization, and you're taking these shortcuts to come up with a product that is going to be on the market. Then after a while, they tell you to add in a feature. But because you copied just code to get the product going, you can't tell what you are doing. So you end up in that you have to fix that technical debt in terms of time, resources, and everything that is needed for you to get started.

Maybe you did a code as a project with our fellow students, and maybe you didn't document one of your students or one of your teammates is sick. And maybe he's the only one who knew the end of the product. So when you're sick, that means the project is at a standstill. So that's a technical debt. But if you guys are documented, and so that your fellow student who is maybe well can start from where the person who is not available.

Yeah, so that's why what we call a technical debt, but we really want to fix this as early as possible so that students don't miss out on grants, maybe you have maybe a grant that you want to pitch your product to such that you access such grants. But because you have a lot of errors in your code, when someone tells you that you run from your application, it can't be run because it has a lot of bugs. Maybe they tell you you give us maybe your documentation, but you don't have the documentation because you never you never had it in your processes as you're developing the product. So all those are what we call term as technical debt or technical gaps in your software development.

So maybe I would like you to tell me like, what are some of the stages that involve yourself as when you're developing a software product? But even by some of the stages? Yeah, what are stages of software development that you engage yourself in when you're coming up with a product? Oh, first of all, I do research. Yes. And then from research, my research involves scouring the internet. Look for now for them is in the front end. I have to look for like inspirations. And also I have to like study the flow of what, let me say my application is going to do.

I have to look for, let me say also make research on the different components that I'm going to put on the framework that I'm going to use. The CSS, very many things come into play. All right, which is what actually holds everything together. It's what determines I think 85% of what will be the outcome. Yeah, that research that you do, right? Yes. Okay, so what are some of the key challenges that you normally encounter that are related to tools, frameworks, and programming languages that you use? You may find that some things are limited and okay, you have to pay for them. Some are far-fetched. Like let me say you you have to go through lots of like redirect links just to get a simple, a simple what? Let me say something like let me say you wanted to get a component. A card component from a certain framework, but it needs you to pay. So what you could do is you could let me say go on Pirate Bay or something and find it there. That's a good idea.

Other thing. The programming languages, what maybe any challenges you find with the learning curve? Hmm, it depends on the prerequisites of the project. But most times the by the time you're going for a project, you know, you know what is needed. Okay, so how do you normally become aware of technical gaps in your project or errors in your code? How do you normally identify them? It comes from changes made by changes made according let's say you're working on a project, and then you the end user or like a project lead or someone says that this needs to be changed.

So that's where the problem comes in. You may find that it needs to, let me say going to have to change it, which will be doable to them. But it may not be possible given where it's supposed to get a certain resource. Yeah

, so you'd see the change requests that suggest that your prototype or your application has a technical gap or has bugs, something like that? First of all, it comes from times you may see it in like, let me say when you push to production. Let me say that. Let me say now you're doing locally before you even host it somewhere. Let's say you're pushing it to Netlify.

It may load on your machine. But then when it comes to like, Netlify CDNs may be loading it on their end, it becomes hard. Let me see. Let me give you an example of like, there's an application that has working one. Yes. By locally in my machine. I could, okay, the JavaScript could run, you get. But when it came to like, hosting it on Netlify for me to actually tap on that, on that button, I had to like, get, you know how like you you get tailwind.css and then you put it in your code. Yeah. Well, I had to get that tailwind.css and then open the like, open get that whole JavaScript file and then load it manually for that thing to just load. Yeah, that was so complicated for you. Yeah. Then also a time comes from like, responsiveness. Given framework, let me say using bootstrap. And then, uh-huh. And then, uh-huh. Yes. And then, uh-huh. And then, uh-huh. You know how people use different browsers. Yeah. You know, that's some CSS frameworks don't work on certain browsers. Yeah, like Explorer. And also Safari. Yeah. Yes. Yeah, that's where another problem comes in. You may find that the user is going to give your product to use Safari. And then yet you for you're looking at the responsiveness from Google Chrome. Yeah. So for them, they'll think that you didn't do it. Well, get you're using one, you know, one generic, let me call it browser. Yeah, you're testing using one browser. Uh-huh. Yes, there's also that. So those are some of the indicators, right? Yes. Yeah. But also those indicators could have some, they could have metrics, metrics could be in form of a duplication of the code. Maybe your code is too complex for you to understand. Sometimes maybe due to code changes, it can be over to maybe ownership of the codes. Only one person knows, maybe also due to poor documentation. Also, maybe to something like that. Well, for me, one thing I know is that when it comes to I don't, okay, I've learned not to write poor code due to components that are out there. Like, if I'm to structure my let me say I'm structuring a hero page, a hero section. Yeah. I'm going to do that. I'm going to have another that's going to be one component because for me, I use React. Let's say, yeah, you're going to have your first hero page. You're going to have another and then the first hero section. Yes. Then the navbar, you know that it has, let me say, options. Let me call them options. Yeah. Yeah. Yes. Yeah. So it has about it has services and whatnot. That code is well laid out. Yes. And even if you fail to like code it yourself, even from outside out there, let me say like flow byte, they will lay it out for you. So there's no excuse that you can say that they can be like poor code or what, or boring good from China. No, no, no, no, no, no, no. That's, I doubt that would be an excuse. All right.

So, let's discuss the essence of this framework: early debt repayment. We talked about how when you take shortcuts or compromise your code, you eventually have to repay. The earlier you repay, the less interest you pay. Are there any practices or strategies in place that encourage early payment or early fixing of these technical gaps that you're aware of? Well, most of the time, it boils down to knowing where to source certain things. For instance, when I work on components, some individuals might struggle with a button. The way they search for solutions matters too. Someone might not know how to query effectively. It's an issue, but how they search for things, like HTML button code, varies. Another person might search for a button code, especially one in React components, possibly using Tailwind or something similar. So, the problem often arises from how someone queries for a solution. Beyond that, I'm not sure.

So, we can appreciate practices like educational awareness, integrating the concept of technical debt where you repay as early as possible, and perhaps setting repayment goals during the project, fixing issues within a specific timeframe. Continuous improvement of your code is crucial too, as is adhering to project deadlines. You can't afford technical debt when you have deadlines to meet. I understand. Yes, code reviews are essential. They should be supplemented by proper code documentation. Those are good practices.

So, what are some incentives or mechanisms that truly motivate you and your team to fix these errors or bugs as early as possible to avoid prolonging them? First and foremost, we need to meet our deadlines. You can't risk leaving some things unresolved. Additionally, for functionality reasons, since some components rely on others to function correctly, we can't leave them hanging. In some cases, a component can have many functionalities, particularly concerning authentication or privileges. If we don't address issues promptly, users won't have a good experience. Exactly. So, you've covered most of what I know. I just wanted to touch on that briefly. But yes, those were excellent insights.

Is there anything else you would like to add or any additional insights you'd like to share regarding technical debt in software prototypes, or perhaps how we can implement a framework to help students minimize technical debt? Maybe providing advice or blogs from various experts on how to manage or avoid technical debt. It could include project management aspects, like how people should structure their projects using different frameworks. For instance, if someone uses React or Python and someone else chooses Django, having a structured approach would be beneficial. When a structure is clear, it's easier to pinpoint issues. It also helps to show that the file is in a specific folder, making it more apparent where the problem might have originated. A well-maintained structure can make problem-solving more efficient.

Thank you so much, Conrad, for these insights. You're quite knowledgeable. I didn't realize front-end developers had such detailed knowledge of software development processes. Usually, it's the back-end folks who have a deep understanding of everything related to software development. I did some back-end work too, but I prefer front-end. I'm just curious about where you acquired all this knowledge. Maybe because you took the time to explore back-end development as well, that's why. Yeah, thank you, Conrad. Perhaps one final question.