**Summarized Transcript**

Certainly, here are the responses with points that can be coded for analysis:

**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?

Points for Coding:

- Stages in software development: Ideation, design, implementation.

- Challenges: Skipping phases, leading to delays and confusion, lack of planning, and skipping database design.

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

Points for Coding:

- Identification: Spending excessive time on specific code, encountering issues due to incomplete documentation, and skipped SDLC phases.

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

Points for Coding:

- Indicators: Code duplication, complex code, frequent changes in the same code section, poor code quality (lack of documentation and testing), and code ownership limited to one person.

**Theme 2: Technical Debt Measurement**

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

Points for Coding:

- Measurement preference: Impact on workflow, higher priority for issues consuming more time.

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

Points for Coding:

- Mainly using Git for code versioning, limited use of specialized tools like StepSize, SonarQube, or CodeClimate.

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

Points for Coding:

- Prioritization: Based on impact on workflow, higher priority for issues causing delays.

**Theme 3: Technical Debt Impact Evaluation**

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

Points for Coding:

- Impact: Unresolved gaps lead to negative effects on performance, reliability, and maintainability, requiring extensive rework and causing delays.

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

Points for Coding:

- Example: Technical debt resulting in delays and confusion in a project, impacting end-user experience and necessitating catch-up work.

**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?

Points for Coding:

- Practices include sprint planning, project planning, awareness of technical debt, repayment goals, continuous code improvement, and providing incentives.

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

Points for Coding:

- Mechanisms include sprint planning, career advancement opportunities, recognition for fixing critical bugs, education and awareness programs, decision-making involving technical debt, and financial incentives.

**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.

Points for Coding:

- Importance of visual aids, practical examples, and accessibility in guidelines for minimizing technical debt to facilitate student understanding and application of best practices.

**FULL TRANSCRIPT**

Interviewer: My name is Mugoya Dihfahsih, and I'm pursuing a master's degree in software engineering at Makerere University. I'm conducting research on student projects, particularly starter projects related to programming, software engineering, or software development.

The main focus of this interview is to identify key metrics for measuring technical debt in software prototypes or software development processes. Have you heard about technical debt before?

Participant: No, this is the first time.

Interviewer: Alright, this is the first time. So, basically, technical debt, what it is, these are consequences of taking shortcuts or making compromises during software development processes. The shortcuts I'm talking about may include not documenting your code, not designing it well, or skipping testing. It might involve copying and pasting code without a full understanding or implementing long lines of code that only one person on the project can comprehend.

So, when that person is unavailable, the project comes to a standstill. All of these contribute to technical debt in projects. Technical debt is similar to financial debt; the longer you delay paying it, the more interest you accrue. But if you address it early, you spend fewer resources, less time, and find it easier to maintain and upgrade your software.

Technical debt can slow down development, introduce bugs, and require additional effort to improve the software. Before we proceed, could you please tell me your name, your course, the year of study, and the role you have in a current project, if any?

Participant: Good evening. Yes, maybe if you come closer to the microphone, I can't hear you well.

Interviewer: Can you hear me now?

Participant: Yeah, I can hear you. Yeah. I'm Mukasa Saidi, and I'm pursuing a Bachelor of Science in Architecture and Innovation at the university. I consider myself as someone involved in software development because I often work on development projects.

Interviewer: Great! So, you're a front-end developer?

Participant: Yeah, I always engage in development activities.

Interviewer: You always engage in development, alright. Have you understood what technical debt is?

Participant: Yeah, I'm trying to grasp what you're saying. It sounds familiar.

Interviewer: That's good to hear. It's quite common; these are everyday things that developers encounter. Yes, we all live with technical debt, and there's no way you can have a 100% perfect software project without any technical debt. The goal is to minimize its occurrence so that we can have a project that is understandable and implementable.

Now, we are going to explore four themes: Technical Debt Identification, Technical Debt Measurement, Technical Debt Impact, and Technical Debt Repayment. In your experience as a front-end developer, what are some of the stages you go through when creating a software prototype or project?

Participant: Firstly, we brainstorm ideas for the application's user interface. During this ideation process, I focus on understanding the user's needs and how the application can address them effectively.

I also consider the skill set available within the team, as different team members may have varying expertise. Technical factors come into play too, such as which frameworks and tools we should use, and whether we need additional team members to support us in building the full application.

Interviewer: That's great. You seem well-versed in software development, essentially following the Software Development Lifecycle (SDLC), right?

Participant: Yes, I'd say so.

Interviewer: What are some of the challenges you typically encounter when using tools, programming languages, and other aspects of software development?

Participant: As a learning student, I'm not yet highly proficient. Debugging may pose a challenge, and sometimes I might need assistance from more experienced individuals. Learning a new technology or language can also be time-consuming.

Interviewer: I see, the learning curve for these technologies can indeed be steep. You often need to put in extra hours to grasp them fully. Now, how do you normally identify or become aware of technical gaps or technical debt in your projects?

Participant: Often, it's through the explanation of a particular topic. For example, someone might explain a piece of code to me, and I'll realize that I've been working with it. When you work on a project for an extended period, you tend to become "blind" to certain issues.

Interviewer: So, it's like when you're pair programming or getting a senior developer to review your code, right?

Participant: Exactly, it's like a code review. They help identify gaps in the code.

Interviewer: Great! What are some indicators or red flags that suggest the presence of technical debt in your project or prototype?

Participant: One indicator is the inability to identify certain issues. Additionally, code ownership, where only one person understands a specific piece of code, and insufficient code documentation, such as comments, are also red flags. Commenting is important, especially when debugging, as it provides context for easy reviewing.

Interviewer: Commenting is indeed crucial for understanding code. It sounds like you have a good grasp of what technical debt entails.

Participant: Yes, technical debt is essentially about the time it takes to address issues. The longer you wait, the more resources it consumes, not only in terms of time but sometimes in marks or even job opportunities.

Interviewer: Those are the effects of technical debt. Thank you for sharing your insights. We'll continue with the second part shortly.

Interviewer: Alright, let's now delve into the topic of measuring technical debt. How do you propose to measure technical gaps in your processes or the product you're working on in phase two?

Participant: Well, in measuring technical gaps in our processes, I usually consider their impact on my progress. For instance, if I find that I'm spending a lot of time reviewing a specific code, I'll give it more priority, let's say, 15 percent compared to other issues. Because it's consuming more of my time.

Interviewer: That's a practical approach. Prioritizing based on the impact it has on your workflow. Have you been utilizing code versioning systems like Git?

Participant: Yes, I use Git for code versioning. It helps me keep track of changes and fixes.

Interviewer: Excellent! Git is indeed a useful tool for measuring technical debt. However, there are specialized tools like StepSize, SonarQube, and CodeClimate designed for this purpose. Have you ever used any of them?

Participant: No, I haven't used those tools before. They're new to me.

Interviewer: Not a problem. These tools can be quite beneficial for measuring and visualizing technical debt in your projects. They provide insights into the criticality of technical debt in your codebase.

Interviewer: Now, let's talk about prioritization. How would you decide which technical gaps to address first in your project?

Participant: Prioritization is not a one-size-fits-all approach. Technical debt can exist at various stages of a project. So, I prioritize based on the specific needs of the project. For me, code documentation is a top priority, particularly in the development phase. Proper documentation helps prevent technical debt from accumulating.

Interviewer: That's a solid choice, focusing on documentation to prevent technical debt. It's essential for understanding and maintaining code. Now, let's discuss how unresolved software gaps affect the quality attributes of your software, such as reliability, performance, and maintainability.

Participant: When we talk about documentation, it starts from the early stages, like ideation and design. If you skip these phases and jump into implementation, it can severely affect your project's progress. For example, neglecting proper database design or UI layout can lead to extensive rework and negatively impact performance and reliability.

Interviewer: I see, skipping critical phases can indeed have adverse effects. Now, could you share a specific example of how technical debt affected your project outcomes and the end-user experience, especially as a front-end developer?

Participant: Certainly. In one project, we were tasked with creating a self-medication application during the COVID-19 pandemic. However, due to a lack of proper planning and skipping essential phases of the Software Development Life Cycle (SDLC), we encountered significant delays and confusion in the project. Many different project ideas emerged within the class, causing further complications. It taught me that skipping critical development steps leads to time-consuming setbacks.

Interviewer: Thank you for sharing that real-world example. It highlights the importance of following proper development practices to avoid technical debt. Now, let's move on to the final phase, which is early repayment. Just like repaying a loan early reduces interest, addressing technical debt promptly reduces its cost. Are there any practices or strategies you'd recommend to encourage early repayment of technical debt in software projects?

Participant: To encourage early repayment, you can implement practices like setting up specific time-bound sprints to ensure that all phases are covered and not skipped. Proper project planning before the start is crucial. Additionally, awareness of technical debt should be part of project planning. Repayment goals can be established, and continuous code improvement should be encouraged. You could also provide incentives, financial or otherwise, for addressing technical debt. Lastly, regular code reviews can help identify and address issues early.

Interviewer: Those are excellent strategies for encouraging early repayment. You've clearly thought about this. Now, what mechanisms or incentives exist to motivate teams or individuals to actively manage and reduce these gaps in software development?

Participant: Mechanisms like sprint planning and proper project management are effective for motivating teams to manage and reduce technical debt. Additionally, career advancement opportunities and recognition sessions for those who fix critical bugs can serve as incentives. Continuous education and awareness about technical debt are essential. Decision-making should include considerations of technical debt. Additionally, offering financial incentives can be a strong motivator.

Interviewer: Those are comprehensive strategies to motivate teams and individuals to address technical debt effectively. Finally, if you were to develop a set of guidelines or a framework to help students minimize technical debt, what key elements would you include in it?

Participant: In such a framework, I would emphasize the importance of illustrations, as students often find visual aids more accessible than lengthy written guides. It could include step-by-step instructions, best practices, and common pitfalls to avoid. Tutorials and practical examples should be part of the framework to facilitate learning. Accessibility, either in print or as software, is essential to ensure students can easily access and apply the guidelines.

Interviewer: Great! Visual aids, practical examples, and accessibility are key elements to consider when developing such a framework. Thank you for your valuable insights, Mukasa Seidi. I appreciate your time and contributions to this discussion.

Participant: You're welcome. Thank you for having me. Just to clarify, my name is Mukasa Seidi, and I'm pursuing a Bachelor's degree in Science and Information Technology.