Summarized for codeable data:

**Theme 1: Technical Debt Identification**

Question 4: Stages in Software Prototypes & Key Challenges

- Stages in software development: Sketching, selecting tech stack, front-end and back-end implementation, testing.

- Key challenge: Styling components, compatibility issues with libraries.

Question 5: Identifying Technical Debt

- Identification: Testing, code logic issues, user feedback, code reviews.

- Red flags: Difficulty understanding code, CSS issues, code metrics for duplication, complexity, and lack of testing/documentation.

**Theme 2: Technical Debt Measurement**

Question 11: Measuring Technical Gaps

- Current measurement: Not measuring but aware of tools like Cypress for speed testing, VS Code extensions for code issues.

Question 12: Tools or Measurements for Technical Debt

- Current tools: GitHub for version control and change tracking.

- No specific measurements for technical debt.

Question 13: Prioritizing Technical Gaps

- Priority: Focus on issues affecting user experience the most.

---

Theme 3: Technical Debt Impact Evaluation

Question 14: Impact on Quality Attributes

- Impact: Unresolved gaps can affect performance, causing slow load times and resource-heavy processes with many users.

Question 15: Specific Impact Examples

- Example: Navigation issues in a community website project negatively impacted user experience.

Theme 4: Early Debt Repayment

Question 16: Practices for Early Repayment

- Practices: Collaboration, feedback collection during testing to identify and address issues early.

Question 17: Incentives for Managing Technical Debt

- Incentives: Speed and efficiency in using applications motivate active management of technical debt.

**Insight:**

Question 18: Additional Insights

- Requested framework for identifying resource-heavy code segments and tools for choosing less resource-intensive technologies.

- Suggested incentives: Career advancement, professional development.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Full Transcript**

Participant: Mugoya Dihfahsih

Course: Pursuing a master's degree in software engineering from Makerere University

Research Focus: Identifying key metrics for measuring technical debt in software development prototypes

—

Participant: Peter Akure

Course: Computer Science, Year Three

Project: Education platform for teaching English in local languages

Themed Questions:

**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?

Mugoya Dihfahsih: My name is Mugoya Dihfahsih, and I'm pursuing a master's degree in software engineering from Makerere University. I'm currently conducting research in identifying key metrics for measuring technical debt in software development prototypes. Technical debt is understood as the consequences of taking shortcuts or making compromises during software development processes, similar to financial debt. When you take shortcuts or don't properly document your code, you accumulate technical debt. It's crucial to repay it as early as possible because delaying the repayment can lead to significant resource drain and introduce more bugs.

Themed Questions:

**Theme 1: Technical Debt Identification (Continued)**

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

Peter Akure: I haven't heard of technical debt before, but after the explanation, I understand it relates to bugs and suboptimal solutions. For example, if I receive code from another source and later need to modify it, not understanding the code can lead to significant delays.

6. What are the stages that you involve yourself in when you're working on projects?

Peter Akure: I begin by sketching out the project and creating a wireframe on paper to visualize it. Then, I choose my front-end stack, create the web app, and start implementing the front end. If necessary, I work on the backend logic. Collaboration with others and testing are also essential steps.

---

Themed Questions:

**Theme 1: Technical Debt Identification (Continued)**

7. What are some of the tools you use, and what are some of the key challenges you face with these tools?

Peter Akure: I use React, DTAP framework, and JavaScript for coding. For styling, I use Tailwind CSS. Styling components can be challenging, especially when the libraries I'm using don't align with my requirements.

8. How do you normally document the challenges you face when working with these tools?

Peter Akure: I haven't been documenting the challenges, but I realize the importance of doing so.

9. How do you identify or become aware of technical gaps in your software?

Peter Akure: I identify technical gaps through testing, checking for issues in the code logic, and user experience feedback. Pair programming or code reviews can also help.

10. What are some indicators or red flags that suggest there is technical debt in your processes or product?

Peter Akure: Red flags include difficulty understanding previously written code, issues with CSS styling, and using code metrics to identify duplication, complexity, and lack of testing or documentation.

---

Themed Questions:

**Theme 2: Technical Debt Measurement**

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

Peter Akure: While I don't currently measure technical debt, tools like Cypress can be useful for testing speed and response time. Extensions in IDEs like VS Code can identify code issues.

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

Peter Akure: I don't currently measure technical debt, but I use GitHub for version control and tracking changes.

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

Peter Akure: I prioritize technical gaps that affect the user experience the most. If a bug makes the user experience terrible, I address it first.

---

Themed Questions:

**Theme 3: Technical Debt Impact Evaluation**

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

Peter Akure: Unresolved gaps can impact performance, causing slow load times or resource-heavy processes, especially when dealing with a large number of users.

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

Peter Akure: In a project to create a community website, navigation issues impacted the user experience. Users had difficulty moving between pages, negatively affecting their experience.

---

Themed Questions:

**Theme 4: Early Debt Repayment**

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

Peter Akure: Collaborating with others on the project and obtaining feedback during the testing phase can help identify and address issues early.

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

Peter Akure: Speed and efficiency in using the application are key motivators for actively managing and reducing gaps. Users prefer fast, efficient experiences.

---

**Insight:**

18. 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?

Peter Akure: I would like a framework to help identify resource-heavy parts of the code and tools to choose less resource-intensive technologies. Additionally, incentives like career advancement and professional development could motivate individuals and teams to manage technical debt effectively.