**Summarized Transcript**

**Theme 1: Technical Debt Identification**

Question 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?

-In software prototype development, stages include planning, design, implementation, testing, and deployment.

- Challenges involve tool selection, requirements alignment, and adapting to changing standards.

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

-The bugs that show up that are difficult to track down

- When it requires a lot of steps to change a feature in the codebase

- Code review during programming with the senior developers

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

-When the user requests for a change and i’m unable to deliver the change on time or if i become hesitant to change the complex code.

- When the implemented solution works but I don’t know how it worked

**Theme 2: Technical Debt Measurement**

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

- Away of keeping track of changes in the code like github does for commit

- If there are some tools that code red flagging so that identifies TD in the project

5. Which current tools or measurements do you use to measure technical debt?

- Mainly rely on code reviews and feedback from supervisors.

- Using Git for version control

6. How do you prioritize which technical gaps to address first?

- Prioritize based on the criticality of affected parts of the application.

- Specific part of the software that is being affected so as to fix those first

- Those that are affecting the application from the users perspective

**Theme 3: Technical Debt Impact Evaluation**

7. How does unresolved software gaps affect the quality attributes of your software prototype?

- The user experience is affected and it becomes unreliable to the users

- Slow application loading, thus low application performance

- A lot of time taken in trying to maintain a product with technical debt

8. Can you provide specific examples of how technical gaps have affected project outcomes or end-user experience?

- Example: Rushed integration of online payments led to convoluted code and user issues without documenting the issues that had to be fixed later so it became hard to trace where to fix.

**Theme 4: Early Debt Repayment**

9. Are there practices or strategies to encourage early repayment or fixing of technical gaps during development?

- Lack of specific practices in place; addressing the financial impact on clients and teams could motivate repayment.

- Unaware of technical debt until things break up in the code.

- The curriculum does not specifically introduce students to technical debt in their practices

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

- Motivation comes from financial implications for clients and users and the ease of fixing issues promptly.

- The ease of fixing the application in case it breaks down, it would motivate students.

**Insights into the framework**

* If the framework would guide students on the usage of tools to minimize TD in their development.
* How to start and set up a project so that students can see how the TD is accumulating(a checklist to guide students throughout the process).

**Full Transcript**

Interviewer: It's like 30 people, but none has heard about technical debt, especially students. It's amazing. So, what do you know about technical debt basically? I don't have a clear description of it, but the way I understand it is technical debt normally comes about with the way we build software in that we might introduce some, we might do some things whereby it seems like a good, it seems like a win at the moment, which is like a quick and easy fix for a certain problem, but then we'll end up having to pay the cost for that quick and easy fix we did much later on.

Student: Absolutely. Oh, thanks so much. By the way, I don't know how glad I am, but I'm even surprised that you know this. Maybe what we can do, you start by introducing your name, your course, your study and the role on the project maybe you're working on.

Student: Yeah, my name is Ekamata Sosolomon. I'm a computer science student here at GCU and in our group project, I'm working as a backend developer.

Interviewer: Great. So, just an overview of the project you're working on.

Student: Yeah, an overview. Our project is a mental health application. It's aiming to help connect students to mental health professionals.

Interviewer: Okay, that's great. Which main objectives do you have all and requirements for your project?

Student: Main objectives? Yeah. Could you explain a little more on that question? I mean, what do you set like the main objective that is supposed to be achieved by your system or the prototype?

Student: Oh, the main objectives. Well, the main objectives that we would like, that we're trying, that we'd like to achieve is to have an easy way for students to connect with mental health professionals, like something whereby they can get to choose a specific professional based on a problem they have, reach out to them and maybe be able to even schedule a session with them.

Interviewer: Great. So, like you were explaining technical debt, basically, like you said, consequences of taking shortcuts or compromises during software development, just like financial debt, it accumulates interest over time, making it harder to maintain, sometimes to update the software in the future. And sometimes this is represented by extra work, challenges that arise when code is not properly designed, documented or tested. And the end result is always it slows down development, it introduces in bugs, and then requires additional effort to fix and improve the software. Basically, that's, I think, the basic description of technical debt and its causes and the effects. Maybe in this interview, we're going to look at only four themes. First of all, we're going to look at technical debt identification, technical debt measurement, technical debt impact, then the repayment of technical debt.

Student: Yeah, so in your experience as you were developing software, what are some of the software development stages that you go through and the key challenges that you encounter may be related to the standards, frameworks, and programming languages that you use?

Student: Some of the steps I normally go through is the ideation phase, trying to come up with an idea for the project. Then the next one after that, we normally have, after the ideation, ideation normally takes a little bit of a while. But should I also stick to the problems I face in ideation as well? If you can. Hello? Yes. Hello. Hello. So can you hear I can hear you?

Interviewer: I think it was a network issue.

Student: Yeah, I was asking that. Where did you last hear me from? Like, what was the last thing you had?

Student: Yeah, you're on ideation, like, ideation of the project ID or something like that.

Student: Yeah. So ideation normally isn't that much of an issue. So maybe the problem that comes that they should come to what an ideation is maybe choosing a specific.

Student: Yes. And maybe trying to see whether it's actually a solvable problem because there could be a situation where I end up picking up a project that seems good on paper, but when the problem is not solvable, it's out of my, let's say my skill level. Yeah. Or maybe I just don't have the tools to solve that problem. So you say that's one of the problems. The ideation is first step, then how the next steps do you go about them?

Student: Then the next step would be trying to design a prototype. So from actually, after ideation, so within ideation, I when I pick a specific problem to solve, I try to see how I can solve that problem. Then if I find a way a specific way to solve that problem, then I go to the point of building a prototype to see if there's I can try to build out something or maybe just have a simple structure of what the solution I want to build could look like.

Interviewer: Okay. So what are those key challenges that you encounter using maybe if you're using a framework or standards or tools?

Student: Well, personally, I haven't been using any specific frameworks or tools. But because normally, I would do try to do like a simple wireframe of the application, maybe like a paper wireframe. Yeah, yeah, yeah. I expect the software application to look like. So yeah, so a simple wireframe, then from that I could move to a digital wireframe on something like Sigma. Okay, then I tried to build out a prototype from that. Great. Yeah, I think that the rest now comes in after the prototype, I think there you go now, design something like that, or you straight go for the implementation?

Student: No, no, no, there'll be obviously a little bit of design involved. I tried to plan out and see what tools I would be going to use, what would be the best option, maybe software wise or framework wise for the problem I'm trying to solve. Then from there, that's where I would go to get something out as quickly as possible. Yeah, so that you come up with an MVP.

Interviewer: Yes. Okay, great. So how do you normally identify or how do you become aware of technical debt in your project?

Student: Some of the ways where I identify technical

debt, I think the very first telltale is, should I say, the bugs that show up, which are normally quite difficult to track down. The bugs that show up and like you can see where they're coming from, but then there are others whereby I have to go down like a long list of errors to find the actual cause of the problem. And also another thing that where I can spot technical debt from is when I try to change something and I realize that I have to go through quite a number of steps to change that specific part of the application that I've built. That's another way to identify that there's a little bit of technical debt involved in that part.

Interviewer: Okay. So do you normally do like code reviews or pair programming, maybe contact stakeholder feedback, something like that?

Student: I've done that. I've done that when I was on my internship. I actually used to do that for almost every feature that I tried to push, we would be assigned a team member to work with. So there'd be normally two people on a project. So I would try to do a review with my, there'd be a supervisor, but also with my team member, try to see if the feature is actually doing what we want it to do. And if possible, also try to look into the implementation of that specific feature before the code is merged. So that's kind of the, should I say review that would be done. And during that review, you could work like identify some bugs, right?

Interviewer: Yes. Yeah. So maybe another question, actually, as if, as if you answered it, but let's go to it and see what are some of the indicators or red flags that suggest that there is a technical debt in your processes or your product?

Student: One indicator that I normally get of technical debt is I can't say that it's a telltale sign, but it's normally around when later when I try to, when I push a product out, and maybe the client asks for a change, and I'm normally pretty hesitant to change something in that specific area of the code. Or if I write something and all I know is that the solution works, but I don't know how it works. So it's normally something that tells me that I might encounter a technical debt later on.

Interviewer: Wow, that's awesome. So let's go to technical debt measurements, how would you like to measure technical gaps in your processes? Could you elaborate a little more on that question?

Student: Yeah, like, you know, like you remember, you remember now like you've, you have gotten maybe you've you've conducted maybe system reviews, and you've realized there's a technical debt. And maybe that red flag that showed you that is a technical debt is you try to add on a product, maybe a feature to your product, but it took you a lot of time and you couldn't fix it in time.

Interviewer: So you would like now to measure how to come up or how to fix this technical debt in your prototype. And maybe some of the ways could be using some tools. Maybe there are tools that you could be using to measure this technical debt. Maybe you're like, I need a tool that enables me to measure this technical debt and aggregate it. So I'm asking how would you like to measure that technical debt in your processes?

Student: Ah, it's actually a really good question. If I were to have a tool, and I would like it to measure my technical debt, I feel like maybe just jumping straight into measuring the date would not be a good thing. Probably if there was a way for me to keep tracking, okay, just like how GitHub normally has, like, just shows me commit history. Yeah, if maybe there's a way to know, maybe have like a warning for potential red flags before I actually get to the point of having the technical debt. Yes, that would be a really good thing. Like if there's a way for me to, if I'm writing my code, and if let's say I try to push out a feature, and there are a few checks that can be run to see whether there is something that might actually tend that might end up being a problem later on, which would actually then result into technical debt, that would be really good. So you'd love to have something like the one that the ones that does the code, maybe analyzing the code and then come up with the code issues, or maybe do code versioning as you're developing something like that.

Interviewer: Yes, something like that. So which are some of the current tools that you normally use to measure technical debt in your prototypes?

Student: Well, one of the tools, obviously GitHub, but what it has been for what I've been working on right now, it has mostly been up to the code reviews that I get most of the people who review my code, mainly my supervisor, he normally checks through and points out issues that could actually end up bringing a problem. Let's say if I'm working on a web application, and I decide to use something like local storage, which is really not advisable. He normally points it out that I think that this is not production ready, and it would end up bringing us problems later on. So that's, that's literally what I've been using, like the code reviews, I've been having code reviews and the people who do review my code, let me know that there might be that there is a potential red flag or something that might come back to bite me later on.

Interviewer: Okay, that's great. So have you ever heard about these tools, maybe step size, sonar cubes and code climate for technical data measurement?

Student: I have heard about code climate, I think I've, I've had a little bit of an interaction with it, but I didn't really look deep into to see what it does. But I've had an interaction with code climate. Yeah, actually, that one also does the measurement, it enables you to aggregate your the level of your technical data and then identifies you which technical data you should maybe resolve first or pay off first, something like that. And then also in VS code, they are what we call have you heard about S lint in VS code?

Interviewer: Yes, lint. Yeah, yes, lint, like in VS code, like the those that

identify, like the causes the potential causes of errors in your code. Yes, I have. Yeah, so that that's for students for starters, that could be one of those. And do you know my use Python, something like that? No, I mean, you work with TypeScript. TypeScript. Do you have you come up with an extension called maybe PTR?

Student: Yes, I have PTR installed. Yeah, you see how it identifies those, maybe you're going some maybe a class is some is structured in a wrong way, something like that enables you may for it doesn't aggregate but at least it flags that this could be a cause of some problem and then something like that. Yes. Yeah. So those are ways these solar cubes and code claimant, they could be maybe premium for maybe if you're working on a project and they are able to fund they can pay for them. But S lint I think and PTR for them, they are really free for free of charge and you can use by students.

Interviewer: Yeah, so let's go to the another bit where we say how do you prioritize the technique which technical data to?

Student: I'm saying technical data. Basically, in my previous interviews, I've not been able to use technical data because we never use the term technical data. Someone's like, what the hell is technical data? They don't understand. But for you, I'm using it because you have an idea on what technical data is.

Interviewer: So how do you prioritize your technical data or which one to address first?

Student: I think I normally check a normally check on the specific parts of the application that are being affected. If it's a very let's have a booking, like a flight booking system. Yes. And that the booking system itself is the one with an issue. That's a very integral part of my application. So I'd normally look at the parts of the application that are really integral to the to solving the problem I'm trying to work with. And try to work on those first because they'd be the ones that are really quite easy to actually be filled on the side of the user and to be seen on my side as well. So I feel my answer would be that I would look at the most integral parts of my application that are being affected and work backwards from there.

Interviewer: Great. So you always look at the criticality of the core of over let me solve the gap in the software, right?

Student: Yes. All right. But I believe it has no size that fits all you can say maybe I use I'll go with maybe I consider the stakeholders feedback yet maybe you have maybe an urgent one. Let me say you have a break in the database connection. So you would first fix the one of database before you go to maybe the stakeholders features that they may want you to implement in your in your in the application, right?

Interviewer: Yes. Okay, so let's go to the impact of technical debt. How do or how does the these unresolved software gaps affect the quality of your product or your prototype in terms of reliability, performance and maintainability?

Student: Well, in terms of reliability, the user experience itself is affected in that. Back to my example of a web app, maybe the booking process is really, really slow, that a user could click on a button and it takes around like 10 seconds for it to be to for them to be progressed to the next part of the application. And normally, by that point, the user has already lost the actually the ones to use your product. So it's one of the main things you look at would be the user experience. And because at that point, I use that if I use this experience, if I use this experience, your application is really bad, that they want to recommend it to somebody else. And really that that that just takes down the reliability a whole lot. Then, could you remind me of what the other parts you had asked something else? Your performance and maintainability? Performance and maintainability. Yeah, I'm sure there would actually be performance hits, like something that can be really seen, like in the in a page load. If I have a page load of around five seconds, that's still very, very, that's a lot of time, five seconds and above, that's a lot of time to wait. Yeah. And that's enough time for somebody to drop off your product there and then. And then maintainability, if it's, it's difficult for me to, so if let's say, I have a prototype and somebody asks for an extra feature, and it's very difficult for me to add it. That's a very good saying that there's a lot of technical data involved in that the product itself, or even that specific part. So when there's a lot of technical data to take you then be a lot of time to add that feature to the product, right?

Interviewer: Yes. Okay, so can you provide a specific example of how technical gap or technical data has affected the project outcome or the end user experience in your, along your way of developing the software product, something like that?

Student: Normally, the very first sign normally what I see is that there's always feedback from the client, because a user of your application, let's say somebody Maybe to interpret a bit, I was like, maybe if you can provide for a specific example where you felt the impact of technical data in your project.

Student: Okay, so a specific example, there's a web app that I worked on during my internship. And at the start, we weren't, we were moving quite fast, because we're trying to beat a few deadlines. So I did things, I just did things to make sure they worked. But then at the point of handing over the product to the client, we started getting good, get calls almost every two or three days about something that's not working the way they expect it. Or even during a demo, there's a time we tried to integrate online payments, and we were in a hurry. So when we integrated online payment, you actually forgot to get a for like normal cash on delivery payments. Yeah, yeah. And the system got really convoluted around that pattern, even finding the exact problem was an issue. Like all we knew, this part of the application is not working. But finding the exact point where the application was breaking from was a very, very, very big problem. Like it took us I think a day or two. And it was simply just one of the function calls. But the way we had written the code had made it really difficult to actually track down where the error was coming from. Oh, so that's one of the biggest ones I remember. That's great. That's a great example. Did you by that time, were you guys doing like documentation?

Student: No, we weren't doing any documentation. And you guys documentation could have saved you like an hour or so?

Student: Yeah, it really would have. All right. Thanks so much. And what are some of the incentives or mechanism that you think could that are in existence that can motivate maybe a team that is working on a project to fix their technical debt as early as possible?

Student: Well, for a team, obviously, the very first thing that would be there is the fact that as a team and

if we were on a team and we're building a software project, there would be there would be a client's money on the line and real actual users involved in it. And if something breaks, your client would be losing money and that would also be affecting the team that's working on the product. Yes. So that's normally the one incentive that I see. But also maybe even should I say that is or how easy it would be to fix something like to fix the application in case something broke down. If there's maybe something that lets you know that if you are to work on this thing and make and try to make sure that it is much easier to fix a problem. That's if let's say my application was in production and it broke down that I can fix the problem within like a maximum of 15 minutes. Yes, that would be a really good incentive.

Interviewer: Wow, that's absolutely wonderful. I don't know how long you've worked with technical but absolutely you've given me the answers that even I'm amazed about because I'm almost like why don't I just ditch the other previous interviews that I did and I just go with only yours because absolutely the others I was doing people are not responding the way you are responding you ask someone the incentive of a technical date you take something like 10 minutes to explain what you mean by the incentive of fixing a technical date. Yeah, but anyway, thanks so much for taking initiative and understanding what technical date is.

Student: Oh, you're welcome. You're welcome. I know, I know. I know. It's one of the things that you actually need to do. It's a very big problem. A lot of people in the software industry are actually having problems with it, but yeah, it's something that we all have to learn at some point.

Interviewer: Yes. So in with this guideline all this framework that I'm coming up with what could be some of the insights or any additional features that we can add in regarding software development by young teams such as students if there is any that you feel like sharing with me I'm already okay to take it in.

Student: Well, I think if there's a way that you could maybe, for example, for me, if I'm working on a web application. Yes. Normally they come with linting inbuilt. Yeah. Like I normally even forget to use the linting actually just to remember it when you're talking about it. Yes. But if there's a way that yours if your product regularly run the linting let's say on my push if I'm pushing my code to GitHub, it runs a lint check. Maybe an update. This is good. It's not going to be like a product. It's going to be just a set of guidelines, a framework, you know, you know, like maybe a framework of let me say a set of rules or standards or direction to guide a student when they're coming up with a product. Hmm. You've got in my head. Okay, then I've. Yes. Yeah. Then maybe if there's something that oh, yeah. So if I think about it, I feel like most times the start of the project affects a lot of the way that things should win the project, even the amount of technical data that you could accrue. So if the way if there's a set of guidelines for how to start and set up a project, yes, that ensure that they are best to make sure that you see technical dates as it comes or as it's growing. That would be really nice. That's like if there's a specific set of guidelines that lets me know that in case that when I'm starting out, I should set up these checks and these checks will help me later on to track and see if there's any technical data being built up. That would really be helpful. Like you have something like a checklist where you were like, maybe I didn't do enough, maybe data collection. Maybe I did the I never analysed my research. Oh, I have taken some shortcuts and I need to fix them. So you have some check checklist. Such by the by the time you maybe you midway, you are like, no, I did I have to fix these errors before I can even proceed to the next level. Yes, because technical what students do, they collect, maybe they have impartial data collection. And then maybe they skip analysis, then they go, they skip the prototyping, they straight away go to implementation. When they go to their supervisors, their supervisors are like, and they go there when maybe they're about even to complete the entire product. And the supervisor tells them, go back and collect data. And then you analyse I think that time they are taking is that they're paying back something that they would have already fixed in. Maybe if they are they had a checklist that shows them that by the time you reach this level, you must have completed this and this and this. Yes, because technical what students do, they collect, maybe they have impartial data collection. And then maybe they skip analysis, then they go, they skip the prototyping, they straight away go to implementation. When they go to their supervisors, their supervisors are like, and they go there when maybe they're about even to complete the entire product. And the supervisor tells them, go back and collect data. And then you analyse I think that time they are taking is that they're paying back something that they would have already fixed in. Maybe if they are they had a checklist that shows them that by the time you reach this level, you must have completed this and this and this. Yes. Yeah. So that's exactly the product, the framework we are coming up with to guide students basically, that by the time they reach maybe the presentation level where they have to present their project, they have a list of things that I've checked out that they have fixed their errors in their in their in their project. And at least you know, with the software development, you cannot have a product that is 100% free of bugs, but at least you've checked out and you somehow right.

Interviewer: All right. Thanks so much. So much for spending your time. I've really gotten some insights in what I'm going to do with this flow.

Student: Yeah, you're welcome. Thank you for even conducting the interview. It's really amazing that somebody is actually out here looking to see how they can actually solve the problems that are in the software industry, because it's a really big problem.

Interviewer: Yeah, thank you so much.