Summarized trancript:

Certainly, here are the responses to the themed questions in reported speech and presented in a brief and concise manner:

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?

- Reported Response: "In his experience, the stages in software prototype development involve planning, team allocation, and budgeting.

- Key challenges include skill gaps in programming languages."

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

- Reported Response: "They plan to use regular meetings, code reviews, and GitHub for versioning to identify technical debt."

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

- Reported Response: "Indicators of technical debt include duplicated code, complexity, poor documentation, and code not easily understood by the team."

Theme 2: Technical Debt Measurement

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

- Reported Response: "They intend to use techniques like stakeholder feedback, code reviews, GitHub versioning, and tracking code issues for measurement."

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

- Reported Response: "They haven't used specific tools yet but are open to exploring tools like CodeClimate and SonarQube for identifying technical debt."

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

- Reported Response: "Prioritization would be based on criticality, performance, and key performance indicators. Focus may be on fixing issues during testing."

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)?

- Reported Response: "Unresolved gaps mainly affect performance and maintainability, potentially leading to slow loading times and making maintenance difficult."

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

- Reported Response: "Specific project examples aren't available yet, but it's expected that technical debt can lead to inefficient systems and a poor user experience."

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?

- Reported Response: "Currently, there aren't specific practices in place, but they consider mentorship and guidance to help students address technical debt early."

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

- Reported Response: "Incentives may arise from the desire to achieve project goals, such as improving system performance and meeting user expectations."

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?

- Reported Response: "They believe that guidance and mentorship can help students navigate technical debt, preventing poor coding practices in the long run."

Full Transcript:

Certainly, here is the entire conversation as a single block:

Mugoya Dihfahsih: "Okay, so for the purposes of the recording, my name is Mugoya Dihfahsih, and I'm doing a master's degree in software engineering and data communication from Makerere University.

And I'm carrying out research in software architecture. Basically, I'm looking at coming up with a framework or building a framework that enables students to identify technical gaps in their prototypes, not only identifying them but also measuring them.

After measuring them, evaluating those technical gaps as early as possible in their development life cycle. So, maybe to bring you to speed, what technical debt is, these are the consequences or repercussions of you taking the shortcuts in implementing your software products.

In relation to a financial debt, where it accumulates over time, you have to pay with interest. The more time you take to pay this debt, the more interest you pay. The less time you take to pay the interest, the less amount you are paying. The same applies to technical debt.

The moment you solve it as early as possible, the less time you take to fix bugs. But the more time you take to fix this technical debt, the more resources you will be using, and the more resources I'm talking in terms of maybe money, maybe time.

Yeah, so that's what a technical debt is. And this could be as a result of poor documentation, neglecting tests, maybe skipping some software development processes, as to expedite your program. Basically, the major cause of technical debt is what we call, have you heard about antipatterns?

Because you passed a degree, maybe a first class, they give you a project, but again you are doing the same stuff, copy paste, copy paste, you are not documenting, you are not testing, you are not, you know.

So, by the time the manager comes to check on the code you've done, it's really hard, they can't read it, they can't implement anything, so the best thing they can do is to maybe fire you and employ someone else.

Yeah, they fire you.

So, there you see you are paying for the debt, you are losing even the job, because you, instead of knowing that you are copying, yeah.

You are giving out your desk copy in the end.

And testing it. Okay, so, let me start by you asking what is your name, the course you are doing, your role on the project you are working on, and yeah.

Semba Martin: "My name is Sempa Martin.

Sempa?

Sempa Martin.

Okay.

I do BIS, that is Bachelor of Information Systems, here at Kyambogo University.

And this project, we have a project, I'm the project leader, and our project consists of five people.

I mean, project leader course, that is too high. I mean, maybe software development, maybe front end, I do back end, I do everything, or you just manage the project?

Actually, I just manage the project.

But, for us, we distributed ourselves in that project, that one, this one is going to be concerned with this technical part, this one is going to be concerned with marketing our project, this one will be talking with the executives like that.

Now, I'm the project leader.

Project leader.

Yeah.

Great.

And I'm the one who came up with the idea, and now we are working on it.

Okay, so, maybe not deep, because you don't want to go deep into your prototype and the project, just main objectives, the requirements of the prototype you are working on?"

Semba Martin: "Okay, our main objectives, it is just we are going to make this one, it is almost a software, it is going to be web-based.

Web-based.

And at a later stage, we can make an application according to the outcome from the users.

Right now, we are targeting different organizations, and our one is Rao, that is the room to read, and our project is for it, it is going to be in another language.

Have you ever seen it?

Do you know Grammar In Check?

Yeah, yeah, Grammar In Check.

Right now, I want to come up with a software that resembles Grammar In Check, but for it, it is going to edit it in the Ugandan language.

You give it sentences, then you transfer them to Uganda.

Okay.

Okay, that is Google Translate, it translates things to other different languages, and it can translate maybe English to Uganda.

But when it translates them to Uganda, they are not in the Grammar language.

Yeah, yeah, yeah.

Uganda Grammar.

Sometimes they are even in Nature.

Yeah, yeah, in Nature.

Now, I want to come up with that Grammar In Check in Uganda.

Okay, so that is great, that is a great idea.

So, our interview is going to be categorized in four themes.

We have technical data identification.

We have technical data measurement.

We have repayment.

Then we have technical data impact.

So, let's start with technical data identification.

In your experience, because you talked about the stages of software development, what are those stages that you involve yourself while developing a prototype?"

Semba Martin: "The stages that we involve when we are going to start with a project, for software development.

Yeah, the software development process.

Yeah, of course, we came up with the first one.

We need to do planning.

We do planning.

We plan for our project.

Identify planning.

It involves coming up with your team.

You sit down to identify the things.

How can I call it?

You make the budget.

You have the work, everything.

You distribute yourself.

This one is going to work on this, and this one is going to work on this.

Something like that.

So, what are some of the key challenges that you normally interface related to tools?

The tools that you use for development.

Maybe the programming languages.

Maybe the frameworks.

Maybe the tools that we encounter right now, they are related to skills.

You know, we are not equipped with all the skills that we need now.

We lack some skills in different programming languages.

Some of the members on the team, they know C and the others know some basics in Python.

Yet, the languages you need are different.

Now, you have to sit down and start training yourselves the language that you basically need.

That's the main problem.

But the other three, there's not so much a problem.

The other materials, because basically, how you deal with your laptop?

You start working on your project and even your time.

The skills, it is the main problem.

Okay, so, what are some, or how do you normally identify or become aware of technical gaps in your project?

In this case, I mean maybe in terms of stakeholder feedback, system reviews.

Because maybe you're doing pair programming or maybe you're doing code reviews.

Or maybe sometimes when code is not working or sometimes when you can't add the feature to your product.

So, how do you normally identify or become aware of technical gaps in your project?

Okay, now to become aware of technical gaps.

So far, we have not moved so much ahead in our project.

Programming indicators, KPIs, that will help us to identify.

Of course, we are going to make regular meetings.

Maybe after two weeks, to see.

Group A, all

of these, we come up with this.

We test it and we see.

That one will help us to identify.

So, that will come into when you're doing a system review.

Okay.

So far, we are going to use that.

And another one, we are going to get a technical person who is going to guide us in evaluation.

For him, he knows more about us.

He will say now, this and this, this should be done and this.

But we improve on this, so that we come up with somewhat of a perfect system.

So, what are some of the indicators that you can really see that these are leading to technical data?

All these are technical data in our product.

What you want to mean is that, what should be the indicators of success in our project?

I want some of, because generally I'm looking at the code.

The code, yeah.

Because technical data, for it, may come in other parts, but here, for architecture, basically I look at the code.

Here, I'm meaning the code matrix.

The code matrix, I can maybe bring you to speed.

You have duplication.

If you have duplicated code, maybe you have complexity in your code.

Your code cannot be easily understood.

Maybe your code is known by only one person.

It's only one person who owns the code.

The rest are just not so sure of what they're doing.

Maybe the code quality is not all that good.

Maybe you've not documented.

Maybe, let me say, maybe you're out of sync with the supervisors.

So, what are these indicators that for you really see that this one, when this happens in my code, I can really see that we're heading into a mess or...

Basically, in the code, if I told someone who comes up with the code, as required,

that we're going to design this and this, to be able to bring it to this.

Now, if that one works out very well, and if it works as intended, maybe you input and it gives the correct output.

And you are totally accurate.

Maybe we can set our measure to be like 85.

It can make 85% of the accuracy of what we need.

And it means that code is going to work for us.

So, you're meaning that the red file could be at a code quality?

Yeah.

Okay, that's great.

So, let's look at technical data measurement.

How do we measure the technical data?

So, how would you like to measure these technical gaps in your product?

I want to measure the technical gaps.

Actually, I need to make a thorough research on it.

I'm applying different techniques that you can use.

So, one of the techniques could be maybe contacting the stakeholders you're developing the system for.

Maybe you also contact, maybe you do what you call code reviews.

You know code reviews?

You give your code to an expert to go through what you've done, and then they identify the data for you.

Maybe you could also do what you call code versioning using GitHub.

Maybe you can also use code issues, like you sit down and you identify issues in your code.

So, there you can be able, those are the techniques.

So, on all those.

And which one?

The stakeholders.

Yeah, stakeholders.

So, you can use GitHub.

For versioning your code.

What else?

Yeah, so, this guy, I'm like, for you, which one?

You know my use, or you prefer?

So, you could maybe, okay, it's more like this.

But GitHub, you know, GitHub, it will save the other things according to their experience.

But this target customer, the stakeholders, they know what we want.

Of course, I may move on with that.

So, which of the current tools have you used to identify technical gaps in your code?

Gaps in our code?

Basically, we just move up the front, so the stakeholders to help you.

I mean, the tools.

You know the tools, maybe to identify technical data.

You have tools that identify technical data.

You have maybe step size.

You have code claimant.

You have SonarQube.

Those ones, they could be extensions in Visual Studio Code that you normally use to identify technical data.

So, you're not interacting with any of those.

Actually, it is even my first time to hear about technical data.

No, technical data, but to hear about those tools.

Actually, you know, we're just developing the experience.

Yeah, I understand.

But now that you are further about them, you can make research on them.

And then you incorporate them in your project.

They really enable you to identify technical data.

You're writing maybe a variable.

Maybe instead of writing, maybe you're developing maybe a login system.

You're saying maybe use a login, but for your writing, let me say xlogin.

So, this tool will tell you that you've written this variable.

Yes, but with time, you will not be able to understand what you meant by this.

So, it raises that flag for you to either fix it or you don't.

If you don't, you go on.

That will continue to be a technical data.

So, with time, you come back and what did I try to mean by xlogin?

You can't even remember.

You can't remember.

So, that is a technical data.

So, these tools are designed to help you identify these technical gaps as early as possible.

The moment you are coding, as you finish that line, it red flags that you have a technical data here.

So, you can easily say, let me fix this or let me continue coding.

I'll fix it later.

So, how would you like to prioritize technical gaps to address first?

How would you like to prioritize them?

I would try, maybe you again break from that equation.

Yeah, like prioritizing technical gaps to address first, maybe in SDLC.

Again, you go to SDLC phase.

Maybe you are at requirements.

That's where I should fix my technical data or maybe designing or maybe at the implementation,

maybe at the testing, maybe at the deployment.

So, how would you prioritize fixing the bugs in your code?

Okay, maybe now we may prioritize mainly in the testing phase.

Because it is very easy to come up with a code, but in testing, you see whether it has brought out exactly what you want.

So, it fixes and you go back to your code and you fix that.

Okay, so, you would like maybe to consider the criticality.

Maybe I'll fix my errors depending on how critical it is, whether high or low.

Because

maybe some errors, they don't even matter.

You just ignore them.

Yeah, I just ignore them.

You don't look back at them.

And again, maybe how it can affect maybe the other.

Maybe the other errors or how it can affect the system.

Okay, so, in terms of repayment, what do you think would be a fair way of repaying technical data in your code?

A fair way of repaying technical data.

Because like, maybe in future, you'll be in charge of a project, you'll be a project manager.

You'll have a software architect under you.

Maybe that software architect will be making those decisions of, "Let's fix these technical gaps."

Now, how would you like, or what would be a fair way for the team or for the developers to repay that technical data?

And yet, they are still continuing to work on the project.

Because as you repay, you are also building.

So, I would suggest maybe we can go back to our phases in the SDLC.

That is the software development lifecycle.

Maybe in the planning phase, that is when you're coming up with your project.

If you identify technical data, maybe at that point, you could maybe decide whether you are going to spend a little bit more time or maybe a little bit more resources to go back and do a thorough research.

Because that technical debt is going to pay in future.

You're going to pay it with resources.

So, at that stage, maybe you could make a decision whether to go back and do a thorough research and maybe address the technical debt there.

At that phase.

So, you mean you could also fix it in the coding phase?

You know, it is very easy to identify technical data while coding.

Maybe you write something that doesn't make any sense.

Or maybe you could not agree with your team.

Yeah, yeah, yeah.

Because if you're doing maybe a pair programming or code review, somebody will definitely tell you, "You've written this code and it doesn't make any sense."

So, we have to rewrite that code.

So, that is one way of paying for the technical debt.

And then the testing phase, again, it is more like it is very technical, but you can still identify technical debt in testing.

Testing, yeah.

Because maybe your code is not accurate. Maybe it is giving, it is not giving the exact results that you intended.

The intended output.

Yeah, yeah.

So, from there, again, you could go back to your code and do some coding.

So, what are some of the risks of not repaying technical debt in your project?

So, what are the risks that we encounter?

Maybe in the future.

In future, in case we don't repay it.

Yeah, maybe it keeps on accumulating and accumulating.

I think the risks are so much.

Maybe you can't be able to make any changes to the code.

To the code.

Even if the customer comes and they're like, "No, you know what?

I wanted this and this and this in your code."

And again, because of the data, it's very hard for you to add a feature to your code.

It's not flexible.

It's not flexible.

Again, it could also be if the code is so complex that you can't easily understand.

Maybe someone wrote the code.

Maybe they left.

And now you have a new developer who has come on board.

They can't even understand your code.

They can't understand it.

So, they will have to rewrite.

So, for us, we may spend a little bit more time rewriting the code.

The same code that you had written.

But maybe because of the data, now it is so hard to understand.

So, they have to rewrite that code.

So, that again is going to be a risk.

And maybe your code could also be breaking.

It could also be breaking.

You make a little change, you add a feature, it is breaking.

So, that's some of the risks.

And it could also be that maybe you're not making profit.

The code is not giving you profit.

Because you're spending more time, more resources on fixing the technical debt.

So, at the end of the day, maybe you're making losses.

So, those are some of the risks.

So, why do you think some developers might be reluctant to repay technical debt in their code?

For me, I think one of the reasons could be a bit of, maybe they are a little bit lazy.

They are a bit lazy.

Because maybe it is so easy, you write the code, you submit.

Yeah, I'm done.

It is done.

But they don't know the consequences of what they are doing.

They don't know what they are doing. They don't know that later they are going to pay for it.

Maybe they are afraid that maybe you write the code and it is giving the correct results.

Yeah, you see, because again, maybe the customer is okay with the code.

But again, in case they come up with maybe a feature or maybe they come up with a new requirement, maybe it is going to be very hard for you to do.

Yeah, to fit in that.

Because maybe your code is not flexible.

So, it is very important for you to have a flexible code.

Maybe to take maybe a little bit more time, maybe a little bit more resources.

You do your research, you come up with a code that is going to be flexible, that is going to be easy to understand.

So, it is very important for you to maybe put aside that.

Maybe for you to have a code that is going to be working for you, maybe for the future. Yeah, not just the current situation, but for the future. So, as a developer, for me, I think that's the approach I would take.

So, in your project, have you encountered any technical debt?

In your project, have you encountered any technical debt?

Actually, for us, we are just in the planning stage.

Yeah, you are still in the planning stage.

We are still in the planning stage, but we have not done anything.

Yeah, we have not done anything, and we have not written any code.

So, you haven't encountered any technical debt yet?

Yeah, we have not.

But now that you know about it, I think you can take measures to prevent it.

Yeah, that is very true.

So, in your experience, what are some of the strategies that you think could help developers avoid or minimize technical debt in their projects?

In their projects, maybe some of the strategies. I think for us, we can start with, let me say, good documentation. Maybe you need to document your code very well. Maybe that is one of the strategies.

That's true.

Another strategy, maybe you need to test your code as you write it. You write one line of code, you test it. You make sure that it is working as expected. Another strategy, you could do code reviews. Maybe you give your code to an expert. You tell them, "Please go through my code and tell me what you think about my code." Maybe they could give you suggestions. Maybe you could also incorporate some of those suggestions. Another strategy, maybe you could also have regular meetings with your team. You sit down, you brainstorm. Maybe you come up with new ideas on how to improve your code. So, those are some of the strategies that I think could

help developers to avoid or minimize technical debt in their code.

Yeah, so, from what I'm getting is that it is very important for you to come up with a code that is going to be flexible.

Yeah, that's true.

Because maybe today you're working on it, maybe on it you're working on one project, and in the future, maybe that project could evolve, or maybe there could be some new requirements, or maybe there could be some new features that the customer wants to add.

Yeah, that's true.

So, you need to come up with a code that is going to be flexible.

Flexible, that's true.

Yeah, I totally agree with you.

So, maybe you could also have regular meetings with your team.

With your team, yeah.

You sit down, you brainstorm, you come up with new ideas.

New ideas, yeah.

Yeah, I totally agree with you.

Yeah, so, I think those are some of the strategies that could help developers to avoid or minimize technical debt in their code.

Yeah.

Okay, so, do you have any questions for me?

Actually, for me, I don't have any questions.

Okay, okay, that's fine.

Yeah.

Okay, so, thank you so much for your time.

You're welcome.

Thank you for coming.

You're welcome. Thank you.

Okay, thank you.

Thank you.

Okay, so, this is the end of the interview. Thank you for participating.

Yeah, thank you so much for having me.

You're welcome.

I enjoyed the interview.

Yeah, thank you.

Thank you.

Thank you. Have a great day.

You too. Thank you.

Thank you. Goodbye.

Goodbye.

Goodbye.

Okay, so, that's the end of the interview. Thank you for participating.

Yeah, thank you so much for having me.

You're welcome.

I enjoyed the interview.

Yeah, thank you.

Thank you.

Thank you. Have a great day.

You too. Thank you.

Thank you. Goodbye.

Goodbye.

Goodbye.

Okay, so, that's the end of the interview. Thank you for participating.

Yeah, thank you so much for having me.

You're welcome.

I enjoyed the interview.

Yeah, thank you.

Thank you.

Thank you. Have a great day.

You too. Thank you.

Thank you. Goodbye.

Goodbye.