**Summary for coding**

Theme 1: Technical Debt Identification

4. Stages in prototype development: Inception, requirement description, development/testing, and user acceptance testing. Key challenge: Lack of well-documented requirements.

5. Identifying technical debt: Mainly through testing and user feedback.

6. Indicators of technical debt: Red flag is a change requiring significant code rewrite.

Theme 2: Technical Debt Measurement

7. Measuring technical gaps: Prioritize based on time and impact.

8. Tools for measurement: Git, VSCode with Prettier and ESLint. Aware of StepSize, SonarCube, and CodeClimate.

9. Prioritization criteria: Based on time and impact, with high-impact changes getting priority.

Theme 3: Technical Debt Impact Evaluation

10. Impact on quality attributes: Unresolved issues affect reliability and, to a lesser extent, performance.

11. Specific examples: Associating school fees with classes led to significant code rewrite, impacting user satisfaction and system performance.

Theme 4: Early Debt Repayment

12. Practices for early repayment: Often depends on the senior engineer's experience.

13. Motivation mechanisms: Understanding risks and time savings motivates early debt repayment.

Insight:

14. Additional insight: Having a checklist or guideline for development standards could benefit startups and young teams by ensuring consistent adherence to important steps.

**Summarized Transcript**

Mugoya Dihfahsih: Okay, so Job, you're welcome to this interview. And I'm glad you made it. My name is Mugoya Dihfahsih, a software engineer doing a master's in data communication and software engineering. And I'm carrying out a research in software architecture, basically looking at identifying,

measuring and as well as evaluating technical data in prototypes by young software practitioners. Yeah, so I would like maybe to bring you up to date of what technical data is.

Technical data is always explained as, oh, it was coined by Cunningham, what? Who said technical data basically is consequences of taking shortcuts or making some optimal solutions or compromises during software development.

Sometimes technical data is compared to a financial debt in that it accumulates interest over time, making it hard to maintain, maybe add a feature to that product.

Sometimes it represents the extra work or the challenges you go through as you're trying to fix the code that were maybe was due to poor design, poor documentation, poor testing.

Yeah. And as the result, the effects are it slows down the development, it introduces the bugs.

And there is always what we call an extra effort or requires more effort to fix and then improve the software like software product. So basically, that's that's an overview of of the technical data.

So I'm going to request you, Mr. Job, to introduce yourself and maybe the startup you're working on and your role in the startup. Yeah, so that we can proceed. Thank you very much, Mr. Difas.

Job: Thank you for that comprehensive explanation of technical debt. Currently, I'm at Extrovert Technologies, a startup focused on software. I'm working on a school management system that we plan to deploy to several schools in the coming months. So that's my role and the project I'm involved in.

Mugoya Dihfahsih: No, that's all that's all that I needed from you. So, yes, thank you. In this interview, we're going to look at major four themes. First theme is going to be about technical data identification. And then the second one will be technical data measurement.

Thirdly, it will be technical data impact. And lastly, it will be our technical data repayment. So, have you ever heard about the term technical data before?

Job: I've heard the term a few times, but your explanation has clarified it for me.

Mugoya Dihfahsih: Alright, so you'll be comfortable if I use technical data or I use some other term, maybe technical gaps or bugs or something like that?

Job: Those terms are familiar and suitable.

Mugoya Dihfahsih: Alright. So, in your experience as a software developer, what are some of the stages that you involve yourself in while developing the prototype for your startup? And what are some of the challenges that you normally encounter while using these tools, maybe standards or frameworks and programming languages?

Job: Our development stages are brief, starting with inception, moving to requirement description, then development and testing, and finally user acceptance testing with the client. The biggest challenge is often lack of well-documented requirements, leaving a lot open to interpretation.

Mugoya Dihfahsih: How do you normally identify or become aware of the bugs or the gaps in your project as your startup is concerned?

Job: Most bugs are identified through testing and user feedback, especially during the user acceptance testing phase.

Mugoya Dihfahsih: What are some of the indicators or red flags that suggest that there must be a technical gap in your project or a bug somewhere in your product?

Job: If a requested change seems like it will require significant rebuilding, that's a red flag. For example, changing how we associate school fees with classes led to a substantial code rewrite.

Mugoya Dihfahsih: How would you like to measure these technical gaps in your product? How do you like quantifying the errors?

Job: I mainly consider the time and impact of fixing the issue. If it's a high-impact change, it gets higher priority.

Mugoya Dihfahsih: What kind of tools do you normally use to measure technical debt in your products?

Job: I use Git for version control and VSCode with plugins like Prettier and ESLint for code quality. I haven't used specific tools for technical debt measurement yet, but I'm aware of options like StepSize, SonarCube, and CodeClimate.

Mugoya Dihfahsih: 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?

Job: Having a checklist or guideline for software development standards could be very helpful. It would ensure that important steps are consistently followed, especially for startups and young teams.

Mugoya Dihfahsih: Thank you for your insights. That's something we can consider incorporating into our framework or guidelines for addressing technical debt. We've come to the end of the interview, and I appreciate your time and input.

**Full Transcript**

Okay, so Job, you're welcome to this interview. And I'm glad you made it. My name is Mugoya Dihfahsih, a software engineer doing a master's in data communication and software engineering. And I'm carrying out a research in software architecture, basically looking at identifying,

measuring and as well as evaluating technical data in prototypes by young software practitioners. Yeah, so I would like maybe to bring you up to date of what technical data is.

Technical data is always explained as, oh, it was coined by Cunningham, what? Who said technical data basically is consequences of taking shortcuts or making some optimal solutions or compromises during software development.

Sometimes technical data is compared to a financial debt in that it accumulates interest over time, making it hard to maintain, maybe add a feature to that product.

Sometimes it represents the extra work or the challenges you go through as you're trying to fix the code that were maybe was due to poor design, poor documentation, poor testing.

Yeah. And as the result, the effects are it slows down the development, it introduces the bugs.

And there is always what we call an extra effort or requires more effort to fix and then improve the software like software product. So basically, that's that's an overview of of the technical data.

So I'm going to request you, Mr. Job, to introduce yourself and maybe the startup you're working on and your role in the startup. Yeah, so that we can proceed. Thank you very much, Mr. Difas.

And thank you for that, I think, very comprehensive description of technical data. Yes, currently, and extrovert technologies, that is a startup still for software.

And apparently what I'm working on is a school management system that we hope to be able to send out to several schools in the next few months. Yes, so I don't know if that covers it or maybe there's something more I might need to describe.

No, that's all that's all that I needed from you. So, yes, thank you. In this interview, we're going to look at major four themes. First theme is going to be about technical data identification. And then the second one will be technical data measurement.

Thirdly, it will be technical data impact. And lastly, it will be our technical data repayment. So, have you ever heard about the term technical data before? I think a few times, but at least with the explanation you've given, I think I'm getting somewhere.

All right, so you'll be comfortable if I use technical data or I use some other term, maybe technical gaps or bugs or something like that? Sure, those are familiar. All right. So that's great.

In your experience as a software developer, that's in the first stage you are in. What are some of the stages that you involve yourself while developing the prototype for your startup? And what are some of the challenges that you normally encounter while using these tools, maybe standards or frameworks and programming languages?

Okay, well, thank you. So I think first of all, the stages are very, very brief on our side. The first one is the inception, where we just meet and have a high-level discussion of the project that we are going to undertake.

Maybe someone gives you a highlight of what it's about. The second part is description of requirements, where we meet and go through requirements. And then, of course, we go into the development, which is coupled with testing. And finally, we now go to what I would call testing, but this time maybe UAT with the client.

And yeah, we sign off. So maybe, of course, to mention, you asked about challenges. The biggest challenge is that most of our requirements are not necessarily documented. You will meet, you discuss, okay, I want a school management system.

I need to be able to pick results. I need to be able to register students. I need to be able to draft payments. But that is as far as it goes. The requirements do not cover how will the UI look, what kind of behavior will I have. So it's more like a verbal thing between you and the client. Yes, but it's also very ambiguous because you can, there's nothing to say, okay, it's based on impression or not.

I'm not sure what to say. But you have to go and decide how you bring it out. And that is also now you don't know whether it will work for the client or not. There's really no standard, but yeah, that's probably the challenge. That is the biggest. Okay, that's great.

So how do you normally identify or become aware of the bugs or the gaps in your project as your startup is concerned? I think most of the bugs are from testing and user complaints. That's all I can tell you. Absolutely, because you said your testing is always at that part when you're getting the feedback from the user.

Yes. Yeah, okay. So what are some of the indicators or red flags that suggest that there must be a technical gap in your project or a bug somewhere or else in your product?

You're saying a gap in the product? Yeah, what are some of the red flags or indicators that show that there is a gap in your product?

Yeah, of course, if someone asks for something, and I know you're in reasonable doubt that this might require a rebuild. That is why I worry. For example, if like recently, we have been attaching school fees to a class.

Like when you're creating a class, you immediately have a field for school fees. But then later you realize, okay, wait, this school fees needs to be captured in a separate table, which can be tracked per term. Now that means a lot of changes have to happen there.

The form which was entering the class now needs to have that field removed. And then that field will be populated in a whole new form altogether for a new table. So I don't know if that covers it maybe. Yeah, basically, you're talking about what they call code churn, something like that.

It's a technical term that refers to code changes in your project. The moment you're doing a lot of changes to the code, that means you have a bug somewhere. Sure. Yeah, that's great. So we go to theme number two of technical measurements.

So how would you like to measure these technical gaps in your product? How would you like to aggregate maybe the bugs, maybe the errors that you have in your prototype? Do you mean in terms of classification?

Like the quantification, yeah. How do you like quantifying the errors? I think it's mostly in terms of time to develop or time to rework. Because if a product can... and also impact. A product can take maybe a day or two to... rather a feature can take a day or two to fix fine.

But also there's fixing a product which, for example, let's say like I was telling you earlier, it means introducing or removing a primary key. Now in that case, it's a high impact change and I'm not sure how to... So usually time and impact. Oh, the impact of the bug itself to the product.

Yes. Basically, that's criticality. Ah, okay. Yeah, okay. So, but also the best way would maybe to use a tool that enables you to aggregate or to measure the impact that you have on your code base.

Okay, now I didn't know there was a tool for that, but that would be helpful. Yeah, actually that was going to be my second question about that of which kind of tools do you normally use to measure technical data in your products? Ah, so unfortunately, apparently not.

But you do. I don't believe there's any software developer who doesn't use Git, right? Ah, okay, fine, yes. Git version control, yes. Yes, and then you see these formatters? Formatters, yes. The... VSCode.

The Prettier and the Eslin, something like that? Yes, yes, those plugins I use on VSCode. Yes, those are normally identifies for you that this is a red flag and this could lead into something that would be messing your code. But if you want to have a tool that normally aggregates and then gives you reports,

something like that, that this is the amount of technical data you have, we have some tools called StepSize. StepSize, we have what we call SonarCubes. And then we also have what we call CodeClimates. All those three, they are premium. They come at a price, and they really help you to identify technical data,

not only identifying it, measuring it, and they show you the rate of its criticality, that you're using a long class, and this is how it's going to impact on your code, something like that, such that by the time you are taking that shortcut,

you know that suboptimal solution of using a long class, you are aware that I have to fix this within this certain time limit. Because if it goes beyond, then that means it is going to incur a lot of interest or it is going to incur you a lot of resources in terms of reading or re-developing,

something like that. Yeah, so those are the tools. So maybe one of those other things could be, how would you like to prioritize which technical data to address first? I think you answered that, right?

I hope I answered you correctly, but yes, it's mostly in terms of the time to build and then impact, like you phrased it. Yeah, but you do it at one single phase of software development? I'm not sure if I've understood you correctly.

Like the measurement of technical data, you do it at a single, like you said for you, basically it's at that level when you're getting the feedback from the user. Is that the only stage you do the measurement? Yeah, mostly, because of course, like I was telling you, the requirements are assumed that you understood entirely,

so you develop according to what you know. It's now at the time when you meet the client and they're asking you, wait, this is what I need, then you have to evaluate, okay, okay, okay, what will this take? Something like that. Alright, then that's great. So we go to the third part of the impact. How would these unresolved software development, software gaps in your product affect the client attributes,

maybe in terms of reliability, performance, and maintainability of your codebase? Sorry, could you repeat that, please? Yeah, I'm saying how does the unresolved gaps in your prototype or product affect the client attributes

of your software in terms of reliability, performance, and maintainability of the codebase? Yeah, of course, I think reliability is greatly affected.

I mean, if there's something that goes unchecked and it could alter data or lead to undesired behavior, then definitely the reliability is affected. Okay. Yes. Then outside of performance? Now, performance, okay, it has rarely been the case, but I believe, of course,

if you have something that could be optimized but is not optimized, then, yeah, that could lead to, yeah. So we haven't killed so much of the performance issue as much as the one whereby somebody needs to have the efficiency of the system. Okay. To be able to, yeah, help somehow, yes.

So don't these bugs also impact on you, maintainability of the code? Pardon me? Because you talked about the documentation, doesn't it also impact on your maintainability of your application? Well, it should, yes. It should because, of course, maintenance becomes harder.

Yes, it does become harder. That's for a fact. Okay. So could you just give me, like, some specific examples in your development where you really believe and you're like, ah, technical debt or technical gap affected me somehow or somewhere in my product?

Yeah, I think, like I was telling you earlier, already we've had that whereby we associated school fees with a class instead of having a separate table for it. So now we have to do a rebuild of those two features.

One, correct the way the class behaves. Two, to introduce the correct way to store the school fees for every class, the fee structure. I just pray that you get a solution for it. But anyway, you told me you just started on it so the code base is not that big so you can easily fix it, right?

Yeah, but now when coming back, like what you said, imagine this is being used by, like, seven schools already. Yes. And then you have to introduce that change. Then the impact is really high. It's huge, yeah, yeah, yeah, yeah. So that goes back to what we were talking about, that fixing it as fast as possible, as we are going to see in the next step, becomes so important for your prototype.

Because if it goes beyond and you can't fix it, maybe you have not fixed it early. Early on, when time goes on, when the code base has grown, it becomes too hard and the impact is really felt by the end users.

And then it also impacts on the other term we talked about of performance. So maybe the application is not performing as the users are expecting it to do, then maybe they lose the interest in the application.

And in the long run, you are losing maybe money and that is also paying back the technical debt. So you are paying the technical debt in the form of losing the clients.

Yes. So maybe we go to the last part of early debt repayment. Yes. So as a startup, are there any tools in the press or any policies in the press that encourage startups or young teams or young team developers to pay technical debt as early as possible?

Just please pardon me on that part.

I'm saying are there some practices or strategies in the press that you know of that encourage maybe young teams or maybe freshers from the university to always fix their technical debt or technical gaps as early as possible? Now, I don't think I am so much aware of that except of course it depends on who your senior engineer is, I guess.

Yeah, yeah, absolutely. Currently, right now, I think that's all I know as far as, because it now depends on the experience of the senior engineer you are working with. That goes on the other issue of maybe pair programming like the juniors pairing with the seniors, then they can easily identify which errors to fix first, right?

Yeah, yeah, yeah. Yeah, okay. So maybe to add on that as a follow-up question, what could be some of the incentives or mechanisms that would motivate a team to reduce their bugs in their code bases as early as possible?

I'm emphasizing as early as possible because you can always fix these gaps even after developing the prototype, but as early as possible, there's a way it motivates you to do these things and, you know, because the code base is still not so much so that you can easily identify where the bug is.

So I'm asking, are there some incentives, what are some of the incentives or mechanisms that motivate teams to actively reduce their technical gaps in their software applications?

Yeah, so I think following what we've been discussing, one of them is just knowing how dangerous it can be to have that technical debt.

So that can be one big, big way to be motivated to clear that. Yeah, but the other one is also probably just saving time because fine, whether you can deal with it later or not, if you realize why don't I just get this out of the way so that when I'm on a new product, I am completely on a new product.

Absolutely. So time is always maybe time of delivering the product is so crucial when you're handling the technical debt because sometimes you may be a user is telling you add this feature.

And because you never fix this date early as possible, you end up in you end up taking a lot of time adding a feature and it is impacting on you.

So it is an incentive to fix that technical data as early as possible, right? Because of time consciousness. Yeah. So as the last part, just the last question in in what we've discussed about this framework.

What is something else that we can incorporate into it or any additional insights that you know of regarding maybe fixing or repaying technical data as early as possible in young teams that we can include in such a framework or guide them to guide maybe people are just coming to software or don't have much knowledge into it, something like that.

Well, thank you. I think on a high level for me, of course, what counts is just knowing whether there's a standard in the first place or not, because if there was some sort of standard, you have like a checklist to refer to and say, okay, every time I'm developing software, I considering these things and those things and the ones like that.

Wow, that's great. You talk of checklist, the things that should be available is maybe during the data collection. This is what you someone should do during maybe the prototyping phase, they should do they should complete this, maybe the encouraging that have you done documentation throughout your SDLC, or even a testing level.

Sometimes maybe in development, you come up with a prototype, instead of you maybe you have maybe let me say you've, you've had maybe changes in your prototype, instead of going back to the prototype and changing the prototype, you just change the product straightforward and things mess up something like that.

So having like a checklist of those guidelines is like you said, it's so important and it will encourage developers to, to fix their technical data as possible.

Thank you. I think that surely works. Yeah. We've come to the end of the interview. And I really want to thank you so much for taking your time. And being part of this short interview. Has it been even short?

Yeah, I was looking forward to hearing a bit more. Yeah. Well, thank you for the insights. Also, I think it's something like, one of the things now I'm thinking about is just having some sort of checklist. Always make sure there's a reference somewhere.

Yes. Doing some of these tasks. Okay, so just in case we are done with the guidelines and that search is done, we can always provide this survey of this checklist and maybe provide because the search has to be validated, right?

Yeah, sure. Yeah, so we can provide such a survey, then you validate through which are maybe activities that are super amount to startups and they can be hopeful for, for young teams.

Ah, that would be nice. Yeah. All right, bro. Thanks so much for your time.