---

Interview Transcript: Bukirwa Angela and Eva Yawenzo (Participant 1)

T1Q1. 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?

- Participant A:

- Mentioned the importance of following SDLC (Software Development Lifecycle) stages, starting with requirements, design, implementation, and deployment.

- Highlighted challenges such as a lack of credible information sources, especially for projects dealing with critical data like health information.

- Addressed security concerns due to handling personal data.

- Discussed the importance of understanding user requirements, interfaces, expectations, and constraints.

- Emphasized the need for in-depth knowledge and credible data sources, particularly for projects dealing with health data.

- Mentioned security as a significant concern, especially when handling personal health data.

- Interviewee C (Eva Yawenzo):

- Echoed the importance of requirements gathering and understanding user expectations.

- Highlighted security as a key concern due to the sensitive nature of health data.

- Mentioned using C and MicroPython for coding, especially for an embedded system project.

T1Q2. How do you identify or become aware of technical debt in your project?

- Participant 1 (Bukirwa Angela):

- Described identifying technical debt through physical testing and error detection.

- Gave an example of identifying gaps when the prototype shows abnormal data like an excessively high heart rate.

T1Q3. What are the indicators/red flags that suggest that there is technical debt in your processes or product?

- Participant 1 (Bukirwa Angela):

- Discussed indicators like code complexity, code duplication, code quality, code ownership, and testing.

- Mentioned the use of intelligent IDEs to identify vulnerabilities and code duplication.

- Emphasized the importance of commenting code for team collaboration.

- Highlighted the use of version control tools like Git for sharing and reviewing code with teammates.

**Theme 2: Technical Debt Measurement based on the provided transcript:**

T2Q1. How would you like to measure technical gaps in your processes or product?

- Participant 1:

- Mentioned using Code Climate as a tool to measure technical gaps.

- Described Code Climate's functionality in identifying vulnerabilities and assisting in resolving issues.

T2Q2. Which are the current tools if any or measurements you would use to measure technical debt?

- Participant 1:

- Confirmed using Code Climate for measuring technical debt.

- Discussed the availability of other tools such as Sonar Cube and Step Size, emphasizing Step Size's capabilities in visualizing error criticality.

T2Q3. How would you prioritize which technical gaps to address first?

- Participant 1:

- Discussed prioritization based on immediate resolution, especially for merge conflicts or critical bugs.

- Focused on resolving issues in the implementation phase before merging into production or staging.

- Acknowledged the importance of addressing technical debt at all stages of development, from requirements collection to implementation and maintenance.

- Highlighted the need to incorporate tools like Code Climate from the early stages of development to avoid overlooking issues that may arise later.

**Theme 3: Technical Debt Impact Evaluation based on the provided transcript:**

T3Q1. How does un-resolved software gaps affect the quality attributes of your software prototype (e.g., reliability, performance, maintainability)?

- Participant 1:

- Mentioned that unresolved gaps in the software lead to backlogs, making it difficult to move forward.

- Discussed the impact on reliability, specifically in the context of a real-time software like the wearable wellness checker, where displaying incorrect data can be a health hazard.

- Emphasized the importance of identifying and addressing such bugs to maintain clean and reliable systems.

- Highlighted the concept of maintainability and how neglecting it can result in the need to revamp or redevelop systems entirely.

- Stressed the importance of using tools to identify bad practices and antipatterns in the code.

T3Q2. Can you provide a specific example of how technical gap has affected the project outcomes or the end-user experience?

- Participant 1:

- Provided a specific example related to the project, mentioning that they initially had separate code running for different parameters (e.g., temperature, heart rate).

- Described a situation where they needed to combine the code into a single class, but due to dependencies and the way the code was structured, certain methods and interactions stopped working.

- Acknowledged that this issue was not predicted during the design phase and equated it to technical debt, comparing it to choosing a loan (e.g., using monolithic architecture) when a better design option (e.g., microservices) was available.

**Theme 4: Early Debt Repayment based on the provided transcript:**

T4Q1. Are there any practices or strategies in place to encourage early repayment or fixing of technical gaps during the software prototype development process?

- Participant 1:

- Discussed strategies and practices to encourage student repayment of technical debt, including educational awareness, rewards, scholarships, and supervisor guidance.

- Emphasized the importance of incorporating technical debt awareness into projects and the need for increased sensitization and awareness among students about the consequences of technical debt.

T4Q2. What incentives or mechanisms exist to motivate the team to actively manage and reduce these gaps?

- Participant 1:

- Highlighted the absence of substantial incentives or mechanisms to encourage proactive management and reduction of technical debt among students, emphasizing the need for increased education and awareness about the importance of addressing technical debt rather than leaving it unattended.

Participant 1:believes that more awareness and education about technical debt should be integrated into the curriculum to help students understand the importance of addressing technical debt early in the development process. They highlighted the potential negative consequences of ignoring technical debt in the real world, where it can lead to low-quality, unmaintainable products.

**Insights:**

Participant 1's response provides valuable insight into their perspective on technical debt in software prototype development:

- Participant 1 acknowledges that the conversation has been helpful and that they have learned something new, specifically about the concept of technical debt.

- They express the importance of technical debt as an aspect that every software engineer or developer should be aware of, highlighting its significance in software development.

- Participant 1 emphasizes the need for more knowledge and resources related to technical debt, particularly for students. They suggest that access to resources explaining technical debt management and recovery should be available to students.

- The participant suggests that technical debt could be included as part of the education framework for students, indicating a desire for formalized education and awareness of technical debt within software engineering programs.

Overall, Participant 1's response underscores the importance of educating students and future software engineers about technical debt to ensure that they are well-prepared to manage and address it effectively in their software development projects.

FULL TRANSCRIPT

<A>My name is Mugoya Dihfahsih and I'm doing a Master's degree in Software Actuaries, Data Communication and Software Engineering. But I'm basically doing the other part of Software Engineering and that's where I'm carrying out research and the research is basically in Software Architecture.

In Software Architecture we are looking at metrics such as reliability, we are looking at maintainability, we are looking at performance of the software. So this research is basically going to look at how do students identify the key metrics in their software development

when they are carrying out their prototypes.

What are the key metrics that they normally follow or do they even identify technical gaps in their software development? So that's the main objective of this interview and maybe to bring you to speed about technical debt these are the consequences that you incur as an individual or as a project due to the sub-optimal solutions such as shortcuts, using the code that you don't understand. And technical debt is a reference to financial data. The more time you take to repay it, the more interest you pay. But in the short term you have to pay that debt. There will come a time when you have to pay that debt. It's not that when you don't pay that debt you never pay it, you will pay it. And to the students sometimes it slows their morale of software development,

it slows their productivity on working on a project. So basically that's what technical debt is. So I'm just going to request you to tell me your name because you are doing the year of your study and then a brief overview of the project you are working on.

<B>I'm Bukirwa Angela, I'm a software engineering student in my fourth year. My role on the project is basically a developer. So our project is wearable wellness checker. I'll just give a small introduction about giving a real world example. You might notice at home that maybe

your father takes like now one and a half spoons of sugar or like limit sodas and all. That means at that elder age they are conscious about their wellness in general. So the wearable wellness checker is a device that is designed to enable a person to keep track of their

wellness. We are going to capture parameters like heart rate, temperature, blood pressure and so on so that a person knows exactly that it's like to be providing first aid on a mobile application for a person just to keep track of their wellness generally.

<C> My name is Eva Yawenzo, I'm a software engineering student finalist working on the same project wearable wellness checker and we are developing a wearable wellness checker that will enable people, specifically we are targeting those that are advanced in age to capture the data from their body and then provide a personalised...

<A>So you work the same role? You are doing the development?

<C>Yes.

<A>So our technical debt is divided into four themes. Theme one is technical debt identification, then theme two is measurement, then theme three is about technical debt evaluation and lastly is technical debt repayment.

A>So in your experience as a software developer, what are the stages that you normally consider when you are developing a software prototype?

<B>Basically from what we've learned or experienced with our lecturers, especially we have the requirements part because before you come up with a solution to give to people you have to know what your system exactly will do. So as I've already described my project, that is not enough for me as a developer to be like let me get my hands dirty and I do this. You have to know how will the user interface with the system, what are the expectations

of the user from that particular system, what constraints, what's the scope of your project, will it be accessible by everyone, is there an administration role, is there a normal user role? <A>So basically it's following the normal SDLC lifecycle. You get requirements,

refine, design, then maybe later on you deploy. You implement and then deploy. <B>Yes. <A>Okay, so what are some of the challenges you normally encounter? During the whole process?

<B>Okay, so basically for my final year project, most cases we don't have credible sources of information. Okay, there is Google Scholar but there are restrictions to other papers. Most times we need that in-depth knowledge so that we totally get the understanding of what we are doing now. For example, for us we are dealing with health and wellness. We

need health data, we need more information about, we are literally dealing with people's lives so we have to be extremely critical with the data and how we shall be handling it. Also the other thing is security. Security as I am explaining the project to you right

now, I'm not sure yet which security constraints I'll be facing because this is personal data. You cannot have someone's health data, temperature, what, accessible to everyone. You never know people's intentions.

<A>So what are some of the tools like frameworks or programming languages and other conditions that you normally use in a project?

<B> So for this project, first of all we have the hardware part of it. It is an embedded system project. So basically what we are dealing with is C and MicroPython. So C, we are running

C in the Arduino IDE. Basically we are consuming libraries all over to help us determine the pulse rate and all whatever. And we are using some MicroPython for machine learning because some data has to be trained over time so that like sleeping, walking steadiness, sleeping patterns, you cannot just have a single code. You have to train your model to understand what happens for a long time.

<A> Okay, so how do you normally, because I believe you always have those gaps in your software prototype, so how do you normally identify them?

<B> So for gaps in software, so far we just do our own physical, we don't have like a, let

me say, we just do normal testing like you yourself, you get the device and you look at that. For example, let me give the example of a heart rate. You know that the normal heart rate is supposed to be between 60 and 100. So when you come up with a prototype

and maybe it's showing 2000 or something, that's when you actually identify that there is a gap. So you have to find a way to eliminate the noise and all.

<A>I mean like in the coding part? <B>Yeah, <A> how do you identify that in my code there is a gap somewhere?

<B> Error. Yeah, when you face the errors, right? Yeah, when I face the errors. It's not like, as long as it works, as I said. As long as it works, you just keep going. Like there is no technical way that we follow during

development. As long as we get the requirement satisfied, we move on.

<A> So what could be some of the indicators or red flags that your code is in a mess or you have a gap in your code? Maybe, this time looking at the code itself, there is code

complexity, maybe there is code duplication, there is the quality of the code itself, then there is code ownership, then you talked about testing. So what is one of those metrics that normally tells you that your code is in a mess or you have a technical gap in your project?

<B>Oh, so for that particular question, I'm just trying to understand it. <A> I've mentioned these metrics. I've mentioned that if there is code duplication, there is code complexity, you have a long line, like you have a long lines of code in one class or in one function, then you have code, the way you change your, how often you change your code, then you have the testing of your code, then you have also the ownership of the code. Maybe you are developing a system, but it's only you who understands the code, your fellow members don't understand the code. Maybe you fall sick or something happens, these guys cannot take on the project, so I'm asking what are those indicators that really…<B> Okay, before I forget, first I'll talk about the advanced technologies that we have. We have intelligent IDEs that help us when you write code, it can easily identify a vulnerability or a code duplication, so with that flag you're able to act upon it. At this point, at this point or this point, there is duplicated code, so you act accordingly. Also, the second part for work collaboration within our teammates, mostly there is a good practice of, like as software developers, you have to comment your code. So basically what I do for my fellow group members to understand what code is being developed is to provide as many comments as possible like if this section is for temperature, I comment it that way. If a certain section is for heart rate, I do the same. <A>Oh, that's good. <B>And maybe lastly, we have version control tools.<A>That's Git. <B>Yeah, Git. You just push, share with your friends, they clone, and they have a view of what you do, which is easier. So they can easily identify that there are errors somewhere, sometime.

<A>So we go to the second theme of technical debt measurement, because after identifying that there is a problem in my code, you have to measure it. It's criticality in a project. So what would you like to, like which tools would you like to use to measure such a technical gap in your code?<B> So aside from the final project, because for the final projects, they've not tailored that testing version. Okay, on different projects I've worked on, there is something called Code Climate. Yeah, Code Climate, I've tried it, it's the one I have here. It's one that comes with... Yeah, there are very many testing tools. I've ever interfaced with it, and it made life easy, in that if you have, you like to commit something with vulnerabilities, it immediately updates you and makes you, helps you resolve those things that could cause merging issues.

<A>So for you, normally you use Code Climate. <B>Yeah, <A>but there is also Sonar Cube, and then there is also Step Size. These are just extensions you can put in Visual Studio Code. They also do the same, but Step Size is much better because actually it can even generate for you the graphics that show that at this level

you had such errors. <B>It actually helps you identify the criticality of your error. <A> The duplication? <B>Yeah, yeah. So all those tools are normally... <A>So that means for you, you are aware of a technical debt, right? Because if you've been using Code Climate, that means you are aware that... <B>But it's not common, so like with our fellow students, they should get more knowledge about it. <A>Yeah, so actually that one you've answered because I was going to ask you which current tools you normally use to measure technical debt. You've said you use Code Climate, yeah.

<A>So how do you normally prioritize these bugs in fixing them?

<B> Still, Code Climate provides the overview of everything. Prioritizing, okay. I'll just give an overview of the side projects I work on. If there is a merge conflict

or if there is a bug in the code, it cannot be merged production or staging if you don't resolve it. So you have to resolve it immediately so that that commit is approved by your fellow workmen. <A>So for you, you are looking at basically the implementation phase.

<B>Yeah, yeah, yeah. <A>That's where you measure, you prioritize your data. <B>Yes, yes. <A>But you also believe that the same bug or the same problem could be due to maybe insufficient data collection or the designs that you made. Maybe you decided to use a monolithic architecture

instead of using microservices. <B>Sure, sure, sure. <A>So do you believe that technical debt cuts across, it needs to be resolved at all the stages, maybe at requirements. You take time and after collecting, you ask yourself, are these requirements enough? You look at your project and you compare. If they're not enough,

you correct and you contact the stakeholders. <B> Sure, sure. Maybe what I can say about that is that, okay, basing on our exposure and the things we interface with, maybe we've been limited to using these tools at the latervstage of implementation. Yet initially they are supposed to be part of the whole development phase that we overlook and we'll be like, ah, skip that for implementation, try using them. Yet we have issues right from the start. Maybe we should learn how to incorporate them just from the requirements collection to the implementation. Implementation and maintenance.

<A> So, you know, because you talked about contacting the stakeholders, we talked about considering the risks that you have in your project, in the long-term effect of your challenges. So those could be one of the things that you consider when you are prioritizing your technical data, right? <B>Yes. <A> All right.

<A>**So let's go to the third theme of impact.**  Okay. Yeah. So how do these unresolved gaps in your software impact your prototype in terms of reliability, performance and maintenance? <B>Before we talk about reliability, performance and maintenance, first of all it leads to backlogs. So if you have those issues unresolved, sometimes it's always hard for you to move forward if you have that particular error and the whole thing is supposed to be like

continuous or there is a certain modularity. It's always hard to move forward. For reliability, for example, you're working on a real-time software like a wearable wellness checker. Someone has to know the heart rate and maybe there is that little glitch and maybe it's

displaying a wrong value. <A>Like that is a health hazard. <B>Yeah, yeah. It's a health hazard in that you provide advisory for a wrong value. So such bugs should be actually identified as the asset that we have clean and reliable systems.

<A>Yeah. Then the other part of maintainability. Maintainability, yes.

<B>If you pick garbage and place garbage like everywhere, it will always be hard for a person to start all over again. Sometimes what happens to such systems they have to fully revamp them or drop them completely. And re-develop. Yeah, and re-develop. So as engineers we have to have that setting that we have to always develop, maintain, enable code and if we use these tools, they can actually help us identify bad practices.

<A>Yeah, thank you so much. Those bad practices. Have you ever heard about antipatterns? <B>Maybe if you explain, I'll see if I have knowledge about it. <A> Antipatterns. You've studied design patterns. <B> Yeah. <A>So the opposite of design patterns, these are the antipatterns. These are like taking shortcuts in your code, just getting code from somewhere, you put in your... So those are the antipatterns. And those are the code smells. Your code is smelling. Someone cannot, maybe you've given this code to your fellow student but they cannot...

<B>Understand. Understand. So someone will just say that your code smells. True. So those are... The antipatterns. Antipatterns. Antipatterns. Yeah. <A>So can you provide a specific part in your project where you really felt like this is due to a technical debt in the prototype?

<B>Yeah, I actually can. Since we are dealing with four parameters, temperature, heart rate, so at a certain point I had code, individual code running separately in different instances. So it reached a point where I had to create a connection between these three parameters

and I had to combine the whole code. You notice if it's a class, it has a whole class structure but I had to create a single class and insert each and everything there. So if you had a dependency, let me say that temperature, heart temperature, something, depending on a certain method, on converting a site, that thing actually cannot work.

<A>So you never predicted that in design? <B>No, I didn't. <A>Yeah, so that's what they call technical debt. So if you choose a loan, maybe you had to use microservices and then you implement just using APIs. You say this app runs independently

from this app so we just connect using a microservice. Maybe you have an independent database that these applications connect to. Maybe then you decide to go with monolithic. The entire stuff is built in one block. So that's where the issue is. But if you understand what microservices are and what monolithic architecture is, they hope you to design the best feature for your application. <B>Yeah,

<A>so we go to fourth theme of repayment of a technical debt. Are there any strategies that have been set in place that encourage students to repay their technical debt? You understand repayment,right?

<B>Yeah, like just fixing. Yeah, fixing. Fixing the issues. Yeah, fixing the issues as fast as possible. Is it during the development or the final project development?

<A>Yeah, I mean like are there any practices in place that might encourage students to

settle their debts as fast as possible? Maybe I talk about these strategies. We have educational awareness. We have rewards. We have scholarships. We have maybe the election from the supervisor.

You sit with the supervisor. They look through your code. You explain to them. Then they guide you on what to do. Then we also incorporate technical debt in a project such that students are aware of technical debt. Someone, let me say, is getting code from GitHub. It's processing it in his code or they are getting it from CHATGPT and then they are using it in their code but they don't understand it. After the final year, maybe this project is superb and they have code for them to pitch. When the experts go through their code, it's really

full of technical debt they don't understand. Then the person loses out on the grant or something like that. So are there any strategies in place that encourage students to pay their technical debt?

<B>No. Currently, the only support we have is from the supervisor. So before you're okay,

We always check on them so that they look at our progress. So as experienced people, they are able to tell you this is not supposed to be this, do this, or please revamp this, reduce the modularity between these components. That's the only advice we have for now. But

still, on the student level, I feel like there should be more sensitization.

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

There should be more education awareness. Education and awareness, which is not there yet. Because currently, we just give up as long as it works. We don't touch it.

<A>So that was going to be my last question, which mechanism do you support that should

help students identify technical debt you said? More awareness. Awareness. So do you believe with me that when students are taught this, maybe a way of taking credit in their prototypes, it enables us to have developers that really know what they are doing. That even when they go in a production world, they produce high-performing products, maintainable. But because students are not aware of such products or such strategies or are not aware even of a technical debt, they end up taking the shortcuts. Provided

it works, don't touch it. But when they go in the real world, maybe after two months, they lose their jobs because they develop products that are not of quality, they are not maintainable, they are not documenting, something like that. But if we encourage students

right from day one, that though you are learning software engineering, you should be aware of these patterns. Even though you want marks, they have given you a coursework, the deadline is tomorrow, instead of you doing it to pass, you should also have that mindset of doing it to learn. Even if I copy this code, okay, you know, technical debt is somehow helpful. Let me give you an example. Imagine you have this product that you are developing, it's not market, and you are like four teams that are working on it. So instead of you understanding

what this code is doing, you just get a solution, you implement it, you roll it out to the users, they use it, you see you are ahead of these other teams. For them, they are trying to understand by the time they finish, you are already getting caught in the market. But

that one comes at an experience level if you know the cost of technical debt. If you have rolled it out in one month, then you come back and try. We took these shortcuts, how can we fix them as early as possible

Insights:

First of all this conversation has been of help to me because i have learnt something very new(Technical Debt) and i have come to understand that this is an important aspect that every software engineer or developer should know, I feel like we should have more knowledge about this aspect and if we could have access to such resources explaining more about technical debt and how we(students) can manage it or how we can recover(repay) in case of issues. Basically we can include it in education of students as a framework.