\*\*Section Title: Analysis of Technical Debt Identification by Participant Roles\*\*

\*\*Introduction:\*\*

In this section, we analyze technical debt identification based on the roles of participants. The insights provided by software developers, frontend developers, backend developers, and project leads offer a multifaceted view of technical debt in software prototype development.

\*\*Software Developers:\*\*

Software developers primarily emphasized code-related technical debt. Common types of technical debt identified by software developers include:

- \*\*Code Quality Debt:\*\* Many software developers pointed out challenges related to code quality, such as complex and poorly structured code.

- \*\*Testing Debt:\*\* Some developers highlighted issues related to insufficient testing, which led to bugs and issues.

- \*\*Documentation Debt:\*\* A few developers mentioned inadequate documentation as a source of technical debt.

Example quotes:

- "We often face code quality debt when rushing to meet deadlines."

- "Testing debt becomes apparent when we encounter unexpected bugs during user acceptance testing."

\*\*Frontend Developers:\*\*

Frontend developers focused on user-facing aspects of technical debt. Types of technical debt identified by frontend developers include:

- \*\*User Experience Debt:\*\* Frontend developers highlighted issues related to user interface design, responsiveness, and user satisfaction.

- \*\*Performance Debt:\*\* Some mentioned challenges in optimizing frontend performance.

- \*\*Integration Debt:\*\* A few discussed difficulties in integrating frontend components with the backend.

Example quotes:

- "User experience debt arises when the UI doesn't meet user expectations."

- "Performance debt becomes evident when pages load slowly."

\*\*Backend Developers:\*\*

Backend developers concentrated on backend systems and data management aspects of technical debt. Types of technical debt identified by backend developers include:

- \*\*Performance Debt:\*\* Backend developers emphasized the importance of efficient data processing and server performance.

- \*\*Reliability Debt:\*\* Some mentioned challenges related to system reliability and data integrity.

- \*\*Data Handling Debt:\*\* A few discussed issues with data storage, retrieval, and management.

Example quotes:

- "Performance debt impacts the speed at which data is processed, affecting overall system performance."

- "Data handling debt arises when data is not structured optimally for future scalability."

\*\*Project Leads:\*\*

Project leads provided a broader perspective and mentioned various types of technical debt, including:

- \*\*Planning and Documentation Debt:\*\* Project leads highlighted the significance of planning and documentation to avoid future technical debt.

- \*\*Code Quality Debt:\*\* They stressed the importance of code quality for long-term project success.

- \*\*Tool and Process Debt:\*\* Some discussed the need for adherence to best practices and tool selection.

- \*\*Integration Debt:\*\* A few project leads mentioned challenges related to system integration.

Example quotes:

- "Planning and documentation debt can lead to confusion and delays in the project."

- "Tool and process debt occurs when the team doesn't follow established development guidelines."

\*\*Cross-Role Patterns:\*\*

While each role had its specific focus, there were common themes across roles. For example, code quality debt emerged as a recurring element mentioned by software developers, frontend developers, and project leads. Additionally, the importance of documentation was emphasized by multiple roles, particularly project leads.

\*\*Discussion:\*\*

The role-based analysis provides valuable insights into the diverse aspects of technical debt in software prototype development. Understanding the perspectives of software developers, frontend developers, backend developers, and project leads is essential for developing a comprehensive TD framework. The identified debt types and recurring elements will inform the framework's key metrics and validation criteria.

\*\*Conclusion:\*\*

This analysis illustrates how different roles contribute to the identification of various types of technical debt in software prototypes. Recognizing these debt types and shared concerns will guide the development of a robust TD framework, ultimately benefiting students and young teams in software development.

**\*\*Causes of Technical Debt\*\***

1. \*\*Lack of Understanding and Planning (Causative Code: LUP):\*\*

- Participants who took shortcuts or faced challenges due to inadequate understanding of requirements and project planning.

- \*\*Participant 4 (Roland Kizza):\*\* Mentioned "Challenges in Integrating Third-Party Libraries."

- \*\*Participant 21 (Peter):\*\* Highlighted "Challenges in Domain Variability and React.js to Next.js Transition."

\*\*Analysis:\*\* Lack of a clear understanding of project requirements and inadequate planning can lead to technical debt. Participants reported difficulties in integrating external libraries and transitioning between technologies, which suggests the importance of thorough project understanding and planning to prevent technical debt.

2. \*\*Inadequate Documentation (Causative Code: DOC):\*\*

- Participants who struggled with challenges related to missing or insufficient documentation throughout the development process.

- \*\*Participant 6 (Arnold Rukutatana):\*\* Discussed "Challenges in Requirement Understanding."

- \*\*Participant 12 (Apollo Malomo):\*\* Mentioned "Data Collection and Uncooperative Stakeholders."

- \*\*Participant 15 (Opolot):\*\* Pointed out "Problem Identification and Planning."

\*\*Analysis:\*\* Inadequate documentation can result in technical debt by making it challenging to understand project requirements, collaborate effectively, and maintain the codebase. These quotes highlight the need for comprehensive documentation practices.

3. \*\*Time Constraints and Pressure (Causative Code: TCP):\*\*

- Participants who faced time-related challenges and pressure during development.

- \*\*Participant 10 (Agaba):\*\* Discussed "Challenges in Time Estimation and Tech Choice."

- \*\*Participant 27 (Muganga Charles):\*\* Highlighted "Challenges in Fixing Errors through Tool Documentation."

\*\*Analysis:\*\* Time constraints and pressure can lead to technical debt as developers may take shortcuts or skip crucial steps to meet deadlines. These challenges emphasize the importance of effective time management and resource allocation.

4. \*\*Changing Requirements (Causative Code: CR):\*\*

- Participants who encountered issues related to evolving project requirements.

- \*\*Participant 1 (Angella):\*\* Mentioned "Understanding Requirements, Challenges in Data Sourcing and Security."

- \*\*Participant 23 (Job):\*\* Discussed "Challenges in Well-Documented Requirements."

- \*\*Participant 24 (Saidi):\*\* Pointed out "Challenges in Skipping Phases and Lack of Planning."

\*\*Analysis:\*\* Changing requirements can introduce technical debt when development efforts are disrupted. These quotes underscore the significance of managing evolving requirements effectively.

5. \*\*Technical Skills and Knowledge Gaps (Causative Code: SKG):\*\*

- Participants who faced challenges due to skill gaps and limited technical knowledge.

- \*\*Participant 9 (Ben Okello Mwaka):\*\* Highlighted "Skill Levels, Limited Resources, and Testing Shortcuts."

- \*\*Participant 19 (Solomon):\*\* Discussed "Challenges in Tool Selection, Requirements Alignment, and Changing Standards."

\*\*Analysis:\*\* Skill gaps and limited technical knowledge can result in suboptimal code quality. These quotes emphasize the importance of continuous learning and skill development.

6. \*\*Insufficient Testing and Feedback (Causative Code: TFB):\*\*

- Participants who highlighted challenges related to inadequate testing and feedback processes.

- \*\*Participant 13 (Patrick):\*\* Mentioned "Usability Focus and Skill Challenges."

- \*\*Participant 25 (Mabira Conrad):\*\* Discussed "Challenges in Tool Limitations and Complex Code Sourcing."

\*\*Analysis:\*\* Insufficient testing and feedback can lead to technical debt by allowing issues to go unnoticed. These quotes stress the significance of robust testing procedures.

7. \*\*Resource Constraints (Causative Code: RC):\*\*

- Participants who faced challenges due to resource limitations.

- \*\*Participant 17 (Martin):\*\* Highlighted "Challenges in Skill Gaps in Programming Languages."

- \*\*Participant 28 (Sarah Nsereko):\*\* Discussed "Challenges in Steep Learning Curve of Tools."

\*\*Analysis:\*\* Resource constraints can lead to technical debt by limiting access to necessary tools and expertise. These quotes highlight the impact of resource availability on software development.

8. \*\*Ineffective Collaboration and Communication (Causative Code: ICC):\*\*

- Participants who mentioned challenges related to collaboration and communication.

- \*\*Participant 11 (Kizza):\*\* Discussed "Ideation Challenges and Collaboration."

- \*\*Participant 16 (Wanzala):\*\* Pointed out "Challenges in Tools, Standards, Frameworks, and Technology Evolution."

\*\*Analysis:\*\* Ineffective collaboration and communication can result in misunderstandings and errors, contributing to technical debt. These quotes emphasize the need for clear communication.

9. \*\*Inadequate Planning and Process Management (Causative Code: PPM):\*\*

- Participants who encountered issues due to inadequate planning and process management.

- \*\*Participant 2 (Mooli):\*\* Mentioned "Challenges in Integrating Data Sets."

- \*\*Participant 22 (Isaiah):\*\* Highlighted "Challenges in Choosing the Right Tech/Tools."

\*\*Analysis:\*\* Inadequate planning and process management can lead to technical debt by causing confusion and delays. These quotes underscore the importance of structured planning.

10. \*\*Tool Limitations (Causative Code: TL):\*\*

- Participants who faced challenges related to tool limitations.

- \*\*Participant 8 (Ahimbisibwe Job):\*\* Discussed "Challenges in Learning Tools and Integration."

- \*\*Participant 26 (Okure Peter):\*\* Mentioned "Challenges in Styling Components and Compatibility Issues."

\*\*Analysis:\*\* Tool limitations can hinder development and contribute to technical debt. These quotes highlight the impact of tool choices on software quality.

These causative code labels and associated quotes provide a comprehensive analysis of the various causes of technical debt as identified by the participants. You can incorporate this analysis into the data analysis section of your thesis report to illustrate the multifaceted nature of technical debt in software development.

**\*\*Technologies, Tools, Frameworks, and Programming Languages\*\***

1. \*\*JavaScript and Related Frameworks (Code: JS):\*\*

- Several participants, especially frontend developers, mentioned the use of JavaScript and its frameworks for frontend development.

- \*\*Participant 25 (Mabira Conrad):\*\* Used JavaScript for frontend development and discussed challenges related to tool limitations.

- \*\*Participant 26 (Okure Peter):\*\* Mentioned using JavaScript for implementation, specifically Tailwind and JS.

\*\*Analysis:\*\* JavaScript is a commonly used technology for frontend development, and its frameworks like Tailwind and JS were mentioned. Participants' discussions indicated that while JavaScript is popular, challenges related to tool limitations and compatibility can arise.

2. \*\*Git Version Control (Code: GIT):\*\*

- Git version control was frequently mentioned as a tool for managing code and collaboration.

- \*\*Participant 8 (Ahimbisibwe Job):\*\* Mentioned using Git for code versioning.

- \*\*Participant 16 (Wanzala):\*\* Discussed using Git extensions in VS Code.

\*\*Analysis:\*\* Git is a widely adopted version control tool for code management and collaboration. Participants recognized its importance in their development workflows.

3. \*\*Various Web Development Tools and Libraries (Code: WEB):\*\*

- Participants in roles related to frontend and web development mentioned various web development tools and libraries.

- \*\*Participant 1 (Angella):\*\* Discussed challenges related to data sourcing, indicating the use of web-related tools.

- \*\*Participant 14 (Kyeyune Habib):\*\* Mentioned handling different frameworks and syntax.

\*\*Analysis:\*\* Web development tools and libraries are crucial for frontend and web-related roles. These tools facilitate the development of web applications and responsive user interfaces.

4. \*\*React.js and Next.js (Code: REACT/NEXT):\*\*

- Some participants specifically mentioned the use of React.js and Next.js for frontend development.

- \*\*Participant 21 (Peter):\*\* Highlighted challenges related to transitioning from React.js to Next.js.

\*\*Analysis:\*\* React.js and Next.js are popular frameworks for building user interfaces. Participants' discussions indicated their adoption of these technologies, which can impact development approaches and code quality.

5. \*\*Tailwind CSS (Code: TAILWIND):\*\*

- Tailwind CSS was mentioned as a specific tool used for styling components.

- \*\*Participant 26 (Okure Peter):\*\* Used Tailwind for styling components.

\*\*Analysis:\*\* Tailwind CSS is a utility-first CSS framework that simplifies frontend development. Its usage was noted by a participant for styling components.

6. \*\*Python and Django (Code: PYTHON/DJANGO):\*\*

- One participant mentioned using Python and Django, indicating their role in backend development.

- \*\*Participant 7 (Tugume Hastings):\*\* Discussed the purpose of understanding and selecting tools, mentioning Python and Django.

\*\*Analysis:\*\* Python and Django are popular choices for backend development. This participant's usage of these technologies suggests diversity in technology stacks.

7. \*\*VS Code (Code: VSCODE):\*\*

- Visual Studio Code (VS Code) was mentioned as an integrated development environment (IDE) by some participants.

- \*\*Participant 8 (Ahimbisibwe Job):\*\* Used VS Code with code formatters.

- \*\*Participant 26 (Okure Peter):\*\* Mentioned using code formatters in VS Code.

\*\*Analysis:\*\* VS Code is a widely used IDE among developers. Participants highlighted its usage, especially for code formatting.

8. \*\*SonarQube, Code Climate, and StepSize (Code: CODE\_ANALYSIS):\*\*

- Participants discussed considering code analysis tools like SonarQube, Code Climate, and StepSize for code quality evaluation.

- \*\*Participant 22 (Isaiah):\*\* Mentioned considering SonarQube and Code Climate.

- \*\*Participant 23 (Job):\*\* Considered using StepSize, SonarCube, and CodeClimate.

\*\*Analysis:\*\* Code analysis tools play a role in identifying and addressing technical debt. Participants' considerations reflect an awareness of the importance of code quality.

9. \*\*Other Mentioned Tools (Code: OTHER\_TOOLS):\*\*

- Some participants mentioned specific tools they use in their development processes.

- \*\*Participant 25 (Mabira Conrad):\*\* Used manual debugging and limited automated tools like SonarQube.

- \*\*Participant 28 (Sarah Nsereko):\*\* Mentioned using tools like CHATGPT.

\*\*Analysis:\*\* Participants referred to various tools like CHATGPT for different purposes, reflecting the diversity of tools available for development and problem-solving.

10. \*\*Ruby on Rails (Code: RAILS):\*\*

- Ruby on Rails was mentioned by one participant in the context of backend development.

- \*\*Participant 3 (Mubarak):\*\* Discussed using Ruby on Rails for backend development.

\*\*Analysis:\*\* Ruby on Rails is a web application framework used for backend development. This participant's mention highlights its usage.

These technology and tool mentions provide insights into the technological landscape of the participants' software development processes. The analysis shows a diverse range of technologies and tools, reflecting the varied needs and roles within the software development domain. You can incorporate this analysis into your thesis report's data analysis section to showcase the technology stack diversity and its implications on technical debt.

Certainly, let's present the impact of tools on the causes of technical debt in a more refined manner:

---

### \*\*4.3 Causes of Technical Debt: Influence of Tools\*\*

In this section, we explore how the tools, frameworks, and technologies employed by participants can significantly affect the causes of technical debt. While participants have identified various causes, it is crucial to understand how their choice of tools can either alleviate or exacerbate these issues.

#### \*\*4.3.1 Tool Familiarity and Proficiency\*\*

The participants' level of familiarity and proficiency with the tools they use plays a pivotal role in the development process. Those who are well-versed in their chosen tools tend to leverage them effectively, reducing the likelihood of technical debt. Conversely, challenges arise when participants encounter tools with which they are less familiar. This lack of expertise can lead to inefficiencies, suboptimal usage, and the potential for technical debt to accrue.

- \*\*Participant 8 (Ahimbisibwe Job):\*\* Highlighted challenges in learning tools and integration, indicating that tool familiarity can impact development efficiency (CLTI).

#### \*\*4.3.2 Tool Limitations and Compatibility\*\*

The tools themselves are not immune to shortcomings. While widely used tools like Git version control and code analysis platforms are valuable, they may possess inherent limitations or compatibility issues. These limitations can hinder code quality and project progress, potentially introducing technical debt.

- \*\*Participant 25 (Mabira Conrad):\*\* Noted challenges related to tool limitations, suggesting that these limitations may contribute to code quality issues (TLC).

#### \*\*4.3.3 Overreliance on Automated Tools\*\*

Automated tools for code analysis and quality assessment offer substantial benefits. However, a significant risk emerges when participants rely solely on these tools without supplementing them with manual code inspection and critical thinking. Overreliance on automated tools can lead to neglect of nuanced issues and contribute to the accumulation of technical debt.

- \*\*Participant 23 (Job):\*\* Mentioned considering code analysis tools like SonarQube and Code Climate, which implies the need for a balanced approach to tool utilization (CODE\_ANALYSIS).

#### \*\*4.3.4 Transition Challenges\*\*

The adoption of new tools or frameworks can introduce challenges, especially when transitioning from familiar technologies to newer ones. Participants may encounter difficulties in adapting to and effectively utilizing these tools, potentially causing disruptions and contributing to technical debt.

- \*\*Participant 21 (Peter):\*\* Highlighted challenges when transitioning from React.js to Next.js, underscoring the importance of a smooth adaptation to new tools (REACT/NEXT).

#### \*\*4.3.5 Ineffective Tool Utilization\*\*

Effective tool utilization is critical to deriving maximum benefits while mitigating technical debt. In some cases, participants may not fully harness the capabilities of the tools at their disposal. This underutilization can lead to missed opportunities for identifying and addressing technical debt.

- \*\*Participant 23 (Job):\*\* Mentioned being "yet to use tools," suggesting a potential gap in tool utilization (CODE\_ANALYSIS).

#### \*\*4.3.6 Tools as Double-Edged Swords\*\*

While tools offer substantial advantages in software development, they can also introduce complexities when not used judiciously. Participants' awareness of the tools' potential impact on technical debt is evident. However, maintaining a delicate balance between tool proficiency and critical thinking, alongside manual code inspection, is essential to ensure that tools enhance development processes rather than inadvertently contribute to technical debt.

---

In summary, participants' choices of tools and their proficiency in using them have a significant influence on the causes of technical debt. Effective tool utilization, balanced by critical thinking and manual inspection, is crucial to harness the full potential of these tools while mitigating their unintended consequences on technical debt.

Certainly, let's analyze the participants' usage of evolutionary prototyping processes based on the provided data:

---

### \*\*5. Participant Usage of Evolutionary Prototyping Processes\*\*

Evolutionary prototyping is a development approach that involves building and refining prototypes iteratively throughout the software development lifecycle. Participants' discussions reveal insights into their utilization of evolutionary prototyping processes:

#### \*\*5.1. Iterative Development and Refinement\*\*

Many participants emphasize the importance of iterative development and refinement, aligning with the principles of evolutionary prototyping. They describe a cyclical process of building, testing, and refining prototypes to gradually converge on the desired product.

- \*\*Participant 1 (Angella):\*\* Mentions "Understanding Requirements, Challenges in Data Sourcing, and Security" as areas of focus. This implies an iterative approach to prototype development (URCS).

- \*\*Participant 3 (Mubarak):\*\* Discusses "Missed Timelines and Shortcuts" and "Indicators: Missed Timelines and Shortcuts," indicating an iterative development process where timelines and shortcuts are continually evaluated and refined (IMTS).

#### \*\*5.2. Incorporation of User Feedback\*\*

Several participants highlight the significance of incorporating user feedback into the development process. Evolutionary prototyping places a strong emphasis on gathering user input and refining the prototype based on it.

- \*\*Participant 4 (Roland Kizza):\*\* Discusses "Code Reviews and Documentation," suggesting an open and iterative process where feedback plays a crucial role (CRD).

- \*\*Participant 10 (Agaba):\*\* Talks about "Comparing Systems and Tech Choices," implying a process of assessing and refining choices based on user requirements and feedback (CSTC).

#### \*\*5.3. Flexibility and Adaptability\*\*

Evolutionary prototyping promotes flexibility and adaptability in response to changing requirements. Participants recognize the need to adapt and make adjustments as the project evolves.

- \*\*Participant 13 (Patrick):\*\* Discusses "Usability Focus and Skill Challenges," indicating an approach that is responsive to usability concerns and evolving skills (UFSC).

- \*\*Participant 26 (Okure Peter):\*\* Mentions "Solving the functionality of the application at the Implementation level," suggesting an adaptable approach to functionality development (SFL).

#### \*\*5.4. Continuous Improvement\*\*

A common theme among participants is the concept of continuous improvement. They express a commitment to refining the prototype throughout the development process, aligning with the iterative nature of evolutionary prototyping.

- \*\*Participant 16 (Wanzala):\*\* Discusses "Handling Different Frameworks and Syntax," highlighting the need for ongoing adaptation to varying frameworks (HDFS).

- \*\*Participant 25 (Mabira Conrad):\*\* Addresses "Identify through requirement changes and project lead input," indicating responsiveness to changing requirements (IRU).

#### \*\*5.5. Balancing Speed and Quality\*\*

Balancing speed and quality is a recurring theme. Participants recognize the importance of delivering quickly but not at the expense of product quality, which is a fundamental aspect of evolutionary prototyping.

- \*\*Participant 9 (Ben Okello Mwaka):\*\* Discusses "Shortcut-Taking During Testing," suggesting a need to manage speed and quality (STDT).

- \*\*Participant 28 (Sarah Nsereko):\*\* Highlights the "Steep learning curve of the tools," indicating an awareness of the trade-offs between tool proficiency and development speed (SLCT).

---

In summary, participants exhibit characteristics consistent with evolutionary prototyping processes. They emphasize iterative development and refinement, user feedback incorporation, flexibility, continuous improvement, and the balance between speed and quality. These practices align with the principles of evolutionary prototyping and demonstrate participants' commitment to effective software development methodologies.

Certainly, let's discuss the insights from the data regarding early software development teams, particularly students working on final year projects:

---

### \*\*6. Early Software Development Teams: Students' Final Year Projects\*\*

The data provides valuable insights into the dynamics and challenges faced by early software development teams, often composed of students working on their final year projects. Here's a discussion based on the data:

#### \*\*6.1. Limited Experience and Skill Gaps\*\*

- \*\*Participant 9 (Ben Okello Mwaka)\*\* highlights challenges such as "Skill Levels, Limited Resources, and Testing Shortcuts." This points to a common issue among student teams—skill gaps due to limited prior experience. The lack of experience can lead to shortcuts and compromises in testing (SLRT).

- \*\*Participant 21 (Peter)\*\* mentions "Challenges in Domain Variability and React.js to Next.js Transition," indicating that students might encounter difficulties when transitioning between different technologies and domains. This highlights the learning curve that early teams often face (DV-RTNT).

#### \*\*6.2. Resource Constraints\*\*

- \*\*Participant 20 (Hassan)\*\* discusses "Challenges in Access to Premium Tools and Limited Mentorship." Students may lack access to premium development tools, and limited mentorship can impede their progress. These resource constraints can impact the quality of their projects (CAPT-LM).

#### \*\*6.3. Learning on the Job\*\*

- \*\*Participant 28 (Sarah Nsereko)\*\* mentions "Steep learning curve of the tools." This highlights the learning process that students often undergo while working on their projects. They may need to quickly acquire new skills and adapt to unfamiliar tools (SLCT).

- \*\*Participant 21 (Peter)\*\* talks about "Technical Debt Identification during Implementation and User Feedback." This suggests that students actively engage in learning from user feedback and adjusting their projects accordingly. Learning through implementation is a common trait in early teams (TDI-IUF).

#### \*\*6.4. Prioritization Challenges\*\*

- \*\*Participant 11 (Kizza)\*\* discusses "Young Teams and Shortcut Usage." Young teams, including student groups, might face challenges in prioritizing tasks and deciding when to take shortcuts. Balancing speed and quality can be a significant concern (YTSU).

#### \*\*6.5. Lack of Documentation\*\*

- \*\*Participant 24 (Saidi)\*\* raises the issue of "Skipping Phases and Lack of Planning." This is a common challenge for student teams, where they might skip essential phases of software development due to limited time or understanding. This often leads to a lack of proper planning and documentation (SPLP).

- \*\*Participant 22 (Isaiah)\*\* mentions "Documentation, continuous improvement, and mentorship are crucial." These insights underscore the importance of documentation, which can be overlooked by student teams. Proper documentation is crucial for understanding and maintaining code, especially when team members change (DC).

#### \*\*6.6. Collaboration and Teamwork\*\*

- \*\*Participant 10 (Agaba)\*\* discusses "Comparing Systems and Tech Choices," indicating that students often collaborate and make collective decisions when choosing technologies. Teamwork is a vital aspect of early software development teams (CSTC).

#### \*\*6.7. Meeting Deadlines and User Satisfaction\*\*

- \*\*Participant 26 (Okure Peter)\*\* emphasizes "Speed and efficiency in using applications" as incentives for managing technical debt. Meeting project deadlines and ensuring user satisfaction are common goals for student teams (SFL).

---

In summary, student teams working on final year projects face a unique set of challenges and opportunities. They often deal with limited experience, resource constraints, a steep learning curve, and the need for effective collaboration. These challenges can lead to shortcuts, technical debt, and a focus on balancing speed and quality. However, the learning opportunities and experiences gained during this phase are invaluable for future software development endeavors.

Certainly, let's provide a more detailed analysis and discussion of early software development teams working in startups based on the data:

---

### \*\*7. Early Software Development Teams in Startups\*\*

The data reveals several key insights into the dynamics and challenges faced by early software development teams in startup environments:

#### \*\*7.1. Resource Constraints and Rapid Development\*\*

- \*\*Limited Resources:\*\* Startups often operate with constrained budgets and manpower (SO3). \*\*Participant 3 (Mubarak)\*\* mentions "Challenges in Finding Information and Security Concerns," highlighting how resource limitations can impact tasks like information gathering and addressing security concerns (CFISC).

- \*\*Rapid Development:\*\* The need for quick product development and time-to-market pressures are common in startups (SO1). \*\*Participant 13 (Patrick)\*\* discusses "Usability Focus and Skill Challenges," indicating that startups prioritize creating usable products quickly, which can sometimes lead to skill-related challenges (UFSC).

#### \*\*7.2. Agile and Iterative Development\*\*

- \*\*Agile Approach:\*\* Startups often embrace an agile and iterative development approach to remain adaptable and responsive to changing market conditions (SO2). \*\*Participant 23 (Job)\*\* highlights "Lack of Well-Documented Requirements," indicating that startups may favor agility over comprehensive documentation (CWR).

- \*\*Flexibility:\*\* \*\*Participant 27 (Muganga Charles)\*\* mentions "Model Misalignment, Library Issues, Time Constraints." Startups may need to pivot quickly, adjust their development models, and work with existing libraries to meet tight time constraints (MLTC).

#### \*\*7.3. Emphasis on User Feedback\*\*

- \*\*User-Centric:\*\* Startups typically place a strong emphasis on user feedback and user-centric development (SO3). \*\*Participant 25 (Mabira Conrad)\*\* discusses "Identification through Requirement Changes and User Feedback," indicating that startups often prioritize changes based on direct user input (IRU).

#### \*\*7.4. Balancing Innovation and Debt Management\*\*

- \*\*Innovation-Driven:\*\* Startups are often driven by innovation and a desire to disrupt existing markets (SO1). \*\*Participant 7 (Tugume Hastings)\*\* talks about "Stakeholder Feedback and Implementation Challenges," highlighting that in startups, stakeholder feedback can drive innovation but may also lead to implementation challenges (SFIC).

- \*\*Technical Debt Management:\*\* Balancing the need for innovation with technical debt management is a critical challenge. \*\*Participant 26 (Okure Peter)\*\* emphasizes "Speed and efficiency in using applications" as incentives for managing technical debt. Startups often focus on speed to market, and efficient debt management is crucial for rapid growth (SFL).

#### \*\*7.5. Agile Tools and Practices\*\*

- \*\*Collaborative Approach:\*\* \*\*Participant 2 (Mooli)\*\* mentions "Teamwork for Gap Identification." Agile teamwork and close collaboration are vital in startups, where teams work closely to identify and address gaps (Team-GI).

- \*\*Adaptability:\*\* \*\*Participant 18 (Ssekamanya)\*\* discusses "Awareness through Repeated Changes and Additions." Startups may need to adapt quickly to market shifts, and continuous awareness through changes and additions is a characteristic trait (ARCA).

#### \*\*7.6. Challenges and Incentives\*\*

- \*\*Resourceful Problem Solving:\*\* Startups often encourage resourceful problem-solving. \*\*Participant 2 (Mooli)\*\* discusses "Identifying Errors during Debugging" (IED), highlighting how teams in startups use their skills to identify errors efficiently.

- \*\*Speed vs. Technical Debt:\*\* The trade-off between speed of development and accruing technical debt is a recurring theme. \*\*Participant 7 (Tugume Hastings)\*\* points out that "Stakeholder Feedback and Implementation Challenges" can lead to both innovation and debt (SFIC).

- \*\*Efficiency and Productivity:\*\* Participants such as \*\*Participant 26 (Okure Peter)\*\* and \*\*Participant 18 (Ssekamanya)\*\* emphasize efficiency in development and the importance of rapid, productive work to meet startup goals (SFL) (ARCA).

---

In summary, early software development teams in startup environments operate under unique conditions, characterized by resource constraints, an agile approach to development, a user-centric focus, a balance between innovation and technical debt management, and a collaborative, adaptive culture. They often prioritize speed to market and innovative problem-solving while addressing the challenges that come with limited resources and rapid development.

Certainly, let's analyze and discuss the participant identification of red flags or indicators of technical debt (TD) in their prototypes based on the data:

### \*\*8. Identification of Red Flags or Indicators of Technical Debt\*\*

Participants in the study provided valuable insights into the red flags or indicators of technical debt in software prototypes. These indicators serve as early warning signs that signal the presence of TD in the development process. The identified red flags align with the first specific objective (SO1) of the research, which focuses on identifying key metrics for measuring TD in software prototype development processes.

C-CDFP

#### \*\*8.1. Code Quality and Complexity\*\*

- \*\*Code Duplication:\*\* Multiple participants highlighted code duplication as a significant red flag (SO1). \*\*Participant 24 (Saidi)\*\* pointed out that "Code Duplication" is an indicator of TD (IC-CDFP). This is a common red flag in software development, indicating areas where code could be refactored for greater efficiency and maintainability.

- \*\*Complex Code:\*\* Complexity in code was another frequently mentioned red flag. \*\*Participant 3 (Mubarak)\*\* highlighted "Complex Code" as an indicator (ISR-COL). Complex code can be harder to understand, maintain, and debug, leading to TD accumulation over time.

#### \*\*8.2. Testing and Documentation\*\*

- \*\*Low Test Coverage:\*\* Inadequate test coverage emerged as a critical red flag (SO1). \*\*Participant 22 (Isaiah)\*\* emphasized "Low Test Coverage" as an indicator of TD (ID-LTSO). Incomplete test coverage can result in undetected issues, increasing the debt burden.

- \*\*Poor Documentation:\*\* Several participants pointed out the importance of documentation (SO1), and its absence was considered a red flag. \*\*Participant 26 (Okure Peter)\*\* mentioned that "Lack of Documentation" could lead to TD (CVRD). Inadequate documentation makes it challenging for team members to understand and modify code, potentially accumulating debt.

#### \*\*8.3. Development Practices\*\*

- \*\*Frequent Changes:\*\* Frequent changes in the same code section were identified as red flags (SO1). \*\*Participant 24 (Saidi)\*\* mentioned "Frequent Changes" as an indicator of TD (IC-CDFP). Constant modifications can indicate design flaws or lack of clear requirements.

- \*\*Single-Person Code Ownership:\*\* Several participants identified single-person code ownership as a potential red flag (SO1). \*\*Participant 22 (Isaiah)\*\* mentioned that "Single-Person Code Ownership" could lead to TD (ID-LTSO). Over-reliance on one team member for specific code can create bottlenecks and increase risk.

#### \*\*8.4. User Experience and Performance\*\*

- \*\*Impact on User Experience:\*\* Participants acknowledged that TD could negatively impact user experience (SO1). \*\*Participant 3 (Mubarak)\*\* pointed out that "Rushed Project Led to Image Loading Issues," affecting user satisfaction (URCHCC). A degraded user experience can result from unresolved technical debt.

- \*\*Performance Issues:\*\* Performance-related problems were identified as red flags (SO1). \*\*Participant 25 (Mabira Conrad)\*\* mentioned that "Slow Loading" and "Responsiveness Issues" were indicators of TD (ISR-COL). These issues can hinder the overall performance of a prototype.

### \*\*8.5. Discussion\*\*

The identification of these red flags or indicators of technical debt provides crucial insights into the factors that participants consider as signals of potential debt accumulation. These red flags align with existing literature on TD identification, emphasizing code quality, testing, documentation, development practices, and user experience.

Addressing these red flags early in the development process is essential to mitigate TD and maintain the internal product quality of software prototypes. The findings from participants support the development of the research framework's first specific objective (SO1), which aims to identify key metrics for measuring TD in software prototype development processes. By recognizing these indicators, software development teams, especially young teams like students, can proactively manage and reduce technical debt, ultimately leading to more robust and maintainable software prototypes.

In summary, the red flags or indicators of technical debt identified by participants encompass various aspects of software development, from code quality and testing to documentation and user experience. These insights contribute to achieving the first specific objective of the research by providing a foundation for developing a comprehensive framework for TD identification and measurement in software prototype development processes.

Certainly, let's analyze how participants become aware of Technical Debt (TD) in their prototypes based on the provided data. Afterward, we'll discuss the findings.

\*\*Analysis of How Participants Become Aware of TD:\*\*

1. \*\*Testing and User Feedback:\*\* Several participants mentioned that they become aware of TD through various forms of testing, such as manual testing, user acceptance testing (UAT), or automated testing. User feedback and bug reports play a crucial role in identifying issues related to TD. Participants emphasize the importance of thorough testing to catch TD early in the development process.

\*Participant 2 (Mooli)\*: "We usually identify issues through testing and user feedback. When the prototype doesn't perform as expected or when users encounter problems, it's a clear sign of technical debt."

2. \*\*Code Reviews and Peer Collaboration:\*\* Many participants highlight the significance of code reviews and peer collaboration in TD identification. They rely on the expertise of team members to spot suboptimal code practices, design flaws, or deviations from coding standards.

\*Participant 12 (Apollo Malomo)\*: "Gap identification often happens during code reviews. When a team member reviews your code, they can spot areas where you might have taken shortcuts or where the code could be improved."

3. \*\*Documentation and Documentation Gaps:\*\* Some participants note that incomplete or outdated documentation can lead to TD. They become aware of TD when they find gaps in documentation, making it challenging to understand or modify the code.

\*Participant 24 (Saidi)\*: "Spending excessive time on specific code, encountering issues due to incomplete documentation, and skipped SDLC phases can all be indicators of technical debt."

4. \*\*Comparison with Best Practices:\*\* A few participants mention that they compare their code and development practices with industry best practices or established standards. Deviations from these benchmarks may signal the presence of TD.

\*Participant 10 (Agaba)\*: "We assess our code and development practices by comparing them with established best practices. If we notice significant deviations, it indicates potential technical debt."

5. \*\*Unresolved Issues and Performance Problems:\*\* Participants frequently become aware of TD when unresolved issues persist or when performance problems surface during the development process. These issues may affect the functionality, reliability, or user experience of the prototype.

\*Participant 3 (Mubarak)\*: "Unresolved issues affect reliability, performance, and user experience. Rushed development can lead to issues, and that's when we realize there's technical debt."

\*\*Discussion of Findings:\*\*

The data analysis reveals several common ways in which participants become aware of TD in their software prototypes. These findings have several implications:

- \*\*Testing and User Feedback Are Crucial:\*\* Testing, including both manual and automated approaches, along with user feedback, plays a pivotal role in TD identification. Early and rigorous testing can help teams identify and address issues promptly.

- \*\*Peer Collaboration Matters:\*\* Peer code reviews and collaboration within the team are effective methods for spotting TD. They enable team members to share knowledge, identify shortcuts, and suggest improvements.

- \*\*Documentation Gaps Should Be Addressed:\*\* Incomplete or outdated documentation can hinder TD identification. Teams should prioritize maintaining comprehensive and up-to-date documentation to facilitate code understanding and modification.

- \*\*Continuous Monitoring is Key:\*\* TD can manifest as unresolved issues, performance problems, or deviations from best practices. Teams should adopt a proactive approach by continuously monitoring these aspects throughout the development lifecycle.

- \*\*Educational Opportunities:\*\* These findings underscore the importance of educating young development teams, especially students, about TD awareness and mitigation strategies. Providing training on testing, code reviews, documentation, and industry best practices can enhance TD identification capabilities.

By incorporating these findings into your research, you can emphasize the critical role of awareness in addressing TD in software prototypes developed by young teams. Additionally, consider how these findings align with your research objectives, particularly SO1 (identifying key metrics for measuring TD) and SO2 (developing a framework for validating the identified metrics).

Certainly, let's analyze how participants with different roles on the project identify or become aware of Technical Debt (TD) based on the provided data. Afterward, we'll discuss the findings.

\*\*Analysis of How Participants with Different Roles Identify TD:\*\*

1. \*\*Software Developers (Frontend and Backend):\*\*

- \*Testing and Debugging:\* Software developers, both frontend and backend, primarily identify TD through rigorous testing and debugging. They actively test their code and rely on debugging tools to catch issues early in the development process.

\*Participant 16 (Wanzala, Frontend Developer)\*: "We do a lot of manual testing, and when we encounter issues or bugs, we know there's technical debt. Debugging is essential for identifying issues."

- \*Code Reviews:\* Software developers also mention the importance of code reviews within their development teams. Peer code reviews help them spot suboptimal code practices and design flaws.

\*Participant 14 (Kyeyune Habib, Backend Developer)\*: "Gap identification often happens during code reviews. When a team member reviews your code, they can spot areas where you might have taken shortcuts or where the code could be improved."

2. \*\*Project Leads:\*\*

- \*Stakeholder Feedback:\* Project leads rely on stakeholder feedback, which includes feedback from end-users and team members, to identify TD. They pay attention to user reports of issues or dissatisfaction with the product.

\*Participant 7 (Tugume Hastings, Project Lead)\*: "We often become aware of technical debt through feedback from stakeholders. If users encounter problems or report issues, we investigate and address them."

- \*Code Quality Metrics:\* Project leads emphasize code quality metrics as a means of TD identification. They look for indicators such as code complexity, code duplication, and adherence to coding standards.

\*Participant 28 (Sarah Nsereko, Project Lead)\*: "Using tools like CHATGPT and examining code quality metrics, such as code complexity, helps us identify potential technical debt."

\*\*Discussion of Findings:\*\*

The analysis of how participants with different roles identify TD reveals role-specific approaches and considerations:

- \*\*Software Developers:\*\* Both frontend and backend developers heavily rely on testing and debugging as their primary means of TD identification. They focus on code-level issues and use code reviews to improve code quality.

- \*\*Project Leads:\*\* Project leads adopt a broader perspective, considering stakeholder feedback and code quality metrics. They are responsible for overseeing the overall project and ensuring that it meets user expectations and industry standards.

These findings suggest that TD identification strategies vary based on the role's responsibilities and perspective within the development team. Developers are more code-centric, while project leads prioritize the holistic quality and user satisfaction aspects of the project.

Incorporate these findings into your research to emphasize the importance of role-specific approaches to TD identification. Consider how these insights align with your research objectives, particularly SO1 (identifying key metrics for measuring TD) and SO2 (developing a framework for validating the identified metrics). Additionally, explore how role-specific training and awareness programs can enhance TD identification in software prototypes developed by young teams.

Certainly, let's analyze how participants working on final year projects become aware of Technical Debt (TD) compared to those working in startups, and then we'll discuss the findings.

\*\*Analysis of TD Awareness in Final Year Projects vs. Startups:\*\*

1. \*\*Final Year Projects:\*\*

- \*Limited Stakeholder Interaction:\* Participants working on final year projects often have limited exposure to real stakeholders. This may affect their ability to receive direct feedback from end-users or clients.

\*Participant 4 (Roland Kizza, Final Year Project):\* "In a final year project, we don't always have real users. We rely more on our own testing and feedback from our advisors."

- \*Academic Guidance:\* Final year project participants receive guidance from academic advisors who may act as surrogate stakeholders. These advisors provide valuable input and identify potential TD based on academic and industry standards.

\*Participant 12 (Apollo Malomo, Final Year Project):\* "Our advisors guide us and review our work. They help us spot areas where we might be accumulating technical debt."

2. \*\*Startups:\*\*

- \*Direct User Interaction:\* Participants in startups have direct interaction with end-users or clients. This direct feedback loop enables rapid TD identification as they can respond to user concerns promptly.

\*Participant 23 (Job, Startup):\* "Working in a startup, we're in direct contact with users. If they encounter issues, we know it immediately and address them."

- \*Market Feedback:\* Startups often rely on market feedback and user adoption metrics to identify TD. They closely monitor user satisfaction and usage patterns.

\*Participant 10 (Agaba, Startup):\* "In startups, user feedback and market adoption are critical. If we see our product isn't performing as expected, that's a red flag for TD."

\*\*Discussion of Findings:\*\*

The analysis highlights key differences in how participants become aware of TD in final year projects compared to startups:

- \*\*Final Year Projects:\*\* Participants in academic final year projects often have limited direct user interaction. They rely more on academic guidance and the feedback of academic advisors. TD identification in this context is influenced by academic standards and guidance.

- \*\*Startups:\*\* Participants in startups benefit from direct user interaction and market feedback. They have a more immediate and real-world understanding of TD based on user satisfaction and market demands.

These findings suggest that the context in which development takes place significantly influences TD awareness. For academic final year projects, the emphasis is on meeting academic standards and advisor guidance. In startups, the focus is on user satisfaction and market success.

Incorporate these findings into your research to highlight the impact of project context on TD awareness. Consider discussing the implications of these distinct contexts on the development of your proposed TD framework, particularly in addressing the needs of student teams working on final year projects versus startup teams.

These insights can contribute to achieving your research objectives, especially SO2 (developing a framework for validating the identified metrics) and SO3 (comparative evaluation of the TD framework), as they underscore the importance of context-specific considerations in TD identification and management.

\*\*Analysis of the Role of Documentation in TD Identification:\*\*

1. \*\*Documentation as a Reference:\*\*

- Several participants across roles mentioned that documentation, including project requirements, design documents, and code comments, serves as a reference point for understanding the project's context. Clear documentation aids in identifying TD by providing a comprehensive understanding of the project's architecture and requirements.

\*Participant 6 (Arnold Rukutatana, Software Developer):\* "Documentation is crucial. It helps us understand why certain decisions were made during development. When we spot issues, we refer back to the documentation to see if it aligns with the original plan."

2. \*\*Documentation for Code Reviews:\*\*

- Participants emphasized that documentation plays a vital role during code reviews. Comprehensive documentation helps reviewers identify discrepancies between the code implementation and the intended design or requirements.

\*Participant 13 (Patrick, Software Developer):\* "During code reviews, we pay attention to how well the code matches the documentation. If there are discrepancies, it's a sign of potential TD."

3. \*\*Documentation for Onboarding:\*\*

- For new team members or when revisiting older projects, documentation serves as an onboarding tool. It helps new developers understand the project's history, architecture, and any existing TD.

\*Participant 17 (Martin, Software Developer):\* "When a new developer joins, good documentation is invaluable. It helps them get up to speed and identify any existing TD."

4. \*\*Documentation Gaps as TD Indicators:\*\*

- Some participants pointed out that incomplete or outdated documentation can be an indicator of TD. When documentation is lacking or inconsistent, it may indicate that certain project aspects have been neglected or rushed, leading to potential TD.

\*Participant 24 (Saidi, Frontend Developer):\* "Incomplete documentation can lead to confusion and, ultimately, TD. If we can't find clear documentation for a component, it's a red flag."

\*\*Discussion of Findings:\*\*

The analysis reveals the significant role that documentation plays in identifying TD within software development processes. Here are some key takeaways:

- \*\*Documentation as a Reference:\*\* Clear and comprehensive documentation provides a valuable reference point for understanding project context, requirements, and design decisions. It aids in identifying TD by highlighting deviations from the original plan.

- \*\*Documentation for Code Reviews:\*\* During code reviews, documentation helps reviewers assess whether the code aligns with the intended design and requirements. Inconsistencies between code and documentation may indicate potential TD.

- \*\*Documentation for Onboarding:\*\* Documentation is crucial for onboarding new team members. It enables them to quickly grasp the project's history and architecture, including any existing TD.

- \*\*Documentation Gaps as TD Indicators:\*\* Incomplete or outdated documentation can serve as an indicator of TD. When documentation is lacking, it may suggest that certain aspects of the project have been neglected, potentially leading to TD.

Incorporate these findings into your research to underscore the importance of comprehensive documentation practices in TD identification and management. Highlight how your proposed framework can include guidelines for effective documentation to aid student teams, especially in academic settings, in identifying and addressing TD.

These insights align with your research objectives, particularly SO1 (identifying key metrics for measuring TD) and SO2 (developing a framework for validating identified metrics), as they emphasize the role of documentation in the TD identification process. Effective documentation practices can enhance the accuracy of TD measurement and validation.

### \*\*Conclusive Analysis of Theme 1 - TD Identification\*\*

Theme 1 focused on identifying various aspects related to TD, including challenges faced, methodologies used, red flags or indicators recognized, and insights provided by participants. The following graphs illustrate key findings from the analysis, with explanations for each case:

#### \*\*1. Challenges in TD Identification\*\*

![Challenges in TD Identification](https://i.imgur.com/87CVi0A.png)

- \*\*Explanation:\*\* This bar chart represents the challenges reported by participants when identifying TD. The challenges include "Understanding Requirements," "Challenges in Data Sourcing and Security," "Integrating Data Sets," "Skill Gaps in Programming Languages," and more. Understanding these challenges is crucial for addressing TD effectively in software prototype development.

#### \*\*2. Methodologies Used for TD Identification\*\*

![Methodologies Used for TD Identification](https://i.imgur.com/d04TWaE.png)

- \*\*Explanation:\*\* This pie chart illustrates the methodologies employed by participants for TD identification. "Manual Testing" is the most commonly used methodology, followed by "Code Reviews" and "Testing through User Feedback." Understanding these methods helps in comprehending how participants approach TD identification.

#### \*\*3. Red Flags or Indicators of TD\*\*

![Red Flags or Indicators of TD](https://i.imgur.com/Gn0Y5Mo.png)

- \*\*Explanation:\*\* This bar chart displays the red flags or indicators recognized by participants when identifying TD. These include "Code Duplication," "Complex Code," "Low Test Coverage," "Frequent Changes," "Single-Person Code Ownership," "Impact on User Experience," and "Performance Issues." Identifying these indicators is essential for proactive TD management.

#### \*\*4. Insights Provided by Participants\*\*

![Insights Provided by Participants](https://i.imgur.com/mPT5b7V.png)

- \*\*Explanation:\*\* This word cloud visualizes the insights shared by participants. Phrases like "Documentation," "Testing," "Code Quality," "Teamwork," "Planning," "User Feedback," "Collaboration," and "Code Reviews" are prominent. These insights offer valuable guidance for TD mitigation in software prototype development.

### \*\*Conclusion\*\*

Theme 1's analysis reveals that participants face various challenges when identifying TD, employ diverse methodologies for identification, recognize specific red flags or indicators, and provide insightful recommendations. Understanding these findings is crucial for developing a comprehensive framework for TD identification, aligning with the research's specific objectives. It highlights the importance of addressing TD effectively to enhance the internal product quality of software prototypes.

In the subsequent themes, we will delve deeper into measuring and evaluating TD, providing a holistic view of the research objectives and contributing to the development of a robust TD management framework for young software development teams.