**Part 1: Theme 1 - Technical Identification**

- Participant 1 (Angella):

- T1Q1: Understanding Requirements, Challenges in Data Sourcing and Security (URCS)

- T1Q2: Gaps Identification through Manual Testing (GIT)

- Participant 2 (Mooli):

- T1Q1: Challenges in Integrating Data Sets (CIDS)

- T1Q2: Teamwork for Gap Identification (Team-GI)

- T1Q3: Indicators of Technical Debt in Project (ITD)

- Participant 3 (Mubarak):

- T1Q1: Challenges in Finding Information and Security Concerns (CFISC)

- T1Q2: Manual Testing and Missed Timelines (MTMT)

- T1Q3: Indicators: Missed Timelines and Shortcuts (IMTS)

- Participant 4 (Roland Kizza):

- T1Q1: Challenges in Integrating Third-Party Libraries (CITL)

- T1Q2: Code Reviews and Documentation (CRD)

- T1Q3: Missed Deadlines and Shortcuts (MDS)

- Participant 5 (Richard):

- T1Q1: Feasibility Survey and Data Acquisition (FSDA)

- T1Q2: Adding Features as Specified by Users (AFSU)

- T1Q3: Time Spent on Development and Documentation (TDD)

- Participant 6 (Arnold Rukutatana):

- T1Q1: Challenges in Requirement Understanding (CRU)

- T1Q2: System Not Working as Intended (SNWI)

- T1Q3: Non-Compliance with Development Patterns (NCDP)

- Participant 7 (Tugume Hastings):

- T1Q1: Model Misalignment, Library Issues, Time Constraints (MLTC)

- T1Q2: Issues during Documentation (ID)

- T1Q3: Stakeholder Feedback and Implementation Challenges (SFIC)

Part 2: Theme 1 - Technical Identification

- Participant 8 (Ahimbisibwe Job):

- T1Q1: Challenges in Learning Tools and Integration (CLTI)

- T1Q2: Identifying Errors during Debugging (IED)

- T1Q3: Code Understanding and Communication Channel (CUCC)

- Participant 9 (Ben Okello Mwaka):

- T1Q1: Skill Levels, Limited Resources, and Testing Shortcuts (SLRT)

- T1Q2: Shortcut-Taking During Testing (STDT)

- T1Q3: Delays in Delivery and Undocumented Features (DDUF)

- Participant 10 (Agaba):

- T1Q1: Challenges in Time Estimation and Tech Choice (CTET)

- T1Q2: Comparing Systems and Tech Choices (CSTC)

- T1Q3: Technical Debt Impact on Aspects (TDIA)

- Participant 11 (Kizza):

- T1Q1: Ideation Challenges and Collaboration (ICC)

- T1Q2: Young Teams and Shortcut Usage (YTSU)

- T1Q3: Consequences of Technical Debt (CTD)

- Participant 12 (Apollo Malomo):

- T1Q1: Data Collection and Uncooperative Stakeholders (DCUS)

- T1Q2: Gap Identification through Peer Testing (GIPT)

- T1Q3: Code Stops and Starts Indicators (CSSI)

- Participant 13 (Patrick):

- T1Q1: Usability Focus and Skill Challenges (UFSC)

- T1Q2: Stakeholder Involvement and Testing (SIT)

- T1Q3: Missing Features and Documentation (MFD)

- Participant 14 (Kyeyune Habib):

- T1Q1: Handling Different Frameworks and Syntax (HDFS)

- T1Q2: Debugging at the Implementation Level (DIL)

- T1Q3: Frontend to Backend Discrepancies (FBBD)

- Participant 15 (Opolot):

- T1Q1: Problem Identification and Planning (PIP)

- T1Q2: Skipping Steps and Software Development Practices (SSSD)

- T1Q3: Testing and Code Review Indicators (TCRI)

Participant 16 (Wanzala):

- T1Q1: Challenges in Tools, Standards, Frameworks, and Technology Evolution (TSTFTE)

- T1Q2: Evaluating Code Understanding and External Code Use (ECU-ECU)

- T1Q3: Indicators: Code Duplication, Complexity, and Inconsistent Standards (ICDCIS)

Participant 17 (Martin):

- T1Q1: Challenges in Skill Gaps in Programming Languages (SGPL)

- T1Q2: Planning Meetings, Code Reviews, and GitHub for TD Identification (PMCG-TDI)

- T1Q3: Indicators: Duplicated Code, Complexity, Poor Documentation, and Code Understanding (IDCPCU)

Participant 18 (Ssekamanya):

- T1Q1: Challenges in Evolving Tools, Standards, Frameworks, Languages, and Conventions (ESTSFLC)

- T1Q2: Awareness through Repeated Changes and Additions (ARCA)

- T1Q3: Indicators: Code Complexity, Duplication, Quality, and Lack of Modularity (ICCDQLM)

Participant 19 (Solomon):

- T1Q1: Challenges in Tool Selection, Requirements Alignment, and Changing Standards (TSRACS)

- T1Q2: Bugs, Complex Feature Changes, and Code Review by Senior Developers (BCFC-CRSD)

- T1Q3: User-Requested Changes and Hesitation in Complex Code (URCHCC)

Participant 20 (Hassan):

- T1Q1: Challenges in Access to Premium Tools and Limited Mentorship (CAPT-LM)

- T1Q2: Strong Typing and Unit Testing for Bug Discovery (STUT-BD)

- T1Q3: Indicators: Failing Tests, Poor Documentation, Code Duplication, and Complex Code (IFPDC)

Participant 21 (Peter):

- T1Q1: Challenges in Domain Variability and React.js to Next.js Transition (DV-RTNT)

- T1Q2: Technical Debt Identification during Implementation and User Feedback (TDI-IUF)

- T1Q3: Error Tools, Learning on the Job, and Project Delivery Delays (ET-LOJ-PDD)

Based on the provided data for Part 4 of Theme 1 - Technical Identification, I've labeled each participant's responses with descriptive labels. Similar responses have been assigned similar labels, following the same criteria as in Part 1, Part 2, and Part 3 to capture common themes and ideas. Here are the labels for Part 4:

Participant 22 (Isaiah):

- T1Q1: Challenges in Choosing the Right Tech/Tools (CRTC)

- T1Q2: Using Tests and Evaluating Code Complexity, Ownership, and Documentation (UT-ECOD)

- T1Q3: Indicators: Code Duplication, Low Test Coverage, and Single-Person Code Ownership (ID-LTSO)

Participant 23 (Job):

- T1Q1: Challenges in Well-Documented Requirements (CWR)

- T1Q2: Identification through Testing and User Feedback (ITU)

- T1Q3: Red Flags: Changes Requiring Significant Code Rewrite (RF-CRSR)

Participant 24 (Saidi):

- T1Q1: Challenges in Skipping Phases and Lack of Planning (SPLP)

- T1Q2: Spending Excessive Time, Incomplete Documentation, and Skipped SDLC Phases (SETID-SP)

- T1Q3: Indicators: Code Duplication, Complex Code, Frequent Changes, and Poor Code Quality (IC-CDFP)

Participant 25 (Mabira Conrad):

- T1Q1: Challenges in Tool Limitations and Complex Code Sourcing (TLC)

- T1Q2: Identification through Requirement Changes and User Feedback (IRU)

- T1Q3: Indicators: Slow Loading, Responsiveness Issues, Complex Code, Ownership Problems, and Lack of Documentation (ISR-COL)

Participant 26 (Okure Peter):

- T1Q1: Challenges in Styling Components and Compatibility Issues (SCCI)

- T1Q2: Testing, Code Logic Issues, User Feedback, and Code Reviews (TCUC)

- T1Q3: Difficulty Understanding Code during Testing/Documentation and User Feedback (DU-CDU)

Participant 27 (Muganga Charles):

- T1Q1: Challenges in Fixing Errors through Tool Documentation (FETD)

- T1Q2: The Results of the Application (TRA)

- T1Q3: Negative Feedback from Stakeholders (NFSS)

Participant 28 (Sarah Nsereko):

- T1Q1: Challenges in Steep Learning Curve of Tools (SLCT)

- T1Q2: Using Tools like CHATGPT and Red Lines in Code (TUCR)

- T1Q3: Code Complexity, Version Control, and Rare Documentation (CVRD)

Related or similar themes into a more concise list with associated code labels:

1. URCS (Understanding Requirements, Challenges in Data Sourcing, and Security)

2. CIDS (Challenges in Integrating Data Sets)

3. ITD (Indicators of Technical Debt in Project)

4. Team-GI (Teamwork for Gap Identification)

5. MTMT (Manual Testing and Missed Timelines)

6. IMTS (Indicators: Missed Timelines and Shortcuts)

7. CRD (Code Reviews and Documentation)

8. MDS (Missed Deadlines and Shortcuts)

9. FSDA (Feasibility Survey and Data Acquisition)

10. AFSU (Adding Features as Specified by Users)

11. TDD (Time Spent on Development and Documentation)

12. CRU (Challenges in Requirement Understanding)

13. SNWI (System Not Working as Intended)

14. NCDP (Non-Compliance with Development Patterns)

15. MLTC (Model Misalignment, Library Issues, Time Constraints)

16. ID (Issues during Documentation)

17. SFIC (Stakeholder Feedback and Implementation Challenges)

18. CLTI (Challenges in Learning Tools and Integration)

19. IED (Identifying Errors during Debugging)

20. CUCC (Code Understanding and Communication Channel)

21. SLRT (Skill Levels, Limited Resources, and Testing Shortcuts)

22. STDT (Shortcut-Taking During Testing)

23. DDUF (Delays in Delivery and Undocumented Features)

24. CTET (Challenges in Time Estimation and Tech Choice)

25. CSTC (Comparing Systems and Tech Choices)

26. TDIA (Technical Debt Impact on Aspects)

27. ICC (Ideation Challenges and Collaboration)

28. YTSU (Young Teams and Shortcut Usage)

29. CTD (Consequences of Technical Debt)

30. DCUS (Data Collection and Uncooperative Stakeholders)

31. GIPT (Gap Identification through Peer Testing)

32. CSSI (Code Stops and Starts Indicators)

33. UFSC (Usability Focus and Skill Challenges)

34. SIT (Stakeholder Involvement and Testing)

35. MFD (Missing Features and Documentation)

36. HDFS (Handling Different Frameworks and Syntax)

37. DIL (Debugging at the Implementation Level)

38. FBBD (Frontend to Backend Discrepancies)

39. PIP (Problem Identification and Planning)

40. SSSD (Skipping Steps and Software Development Practices)

41. TCRI (Testing and Code Review Indicators)

42. TSTFTE (Challenges in Tools, Standards, Frameworks, and Technology Evolution)

43. ECU-ECU (Evaluating Code Understanding and External Code Use)

44. ICDCIS (Indicators: Code Duplication, Complexity, and Inconsistent Standards)

45. SGPL (Challenges in Skill Gaps in Programming Languages)

46. PMCG-TDI (Planning Meetings, Code Reviews, and GitHub for TD Identification)

47. IDCPCU (Indicators: Duplicated Code, Complexity, Poor Documentation, and Code Understanding)

48. ESTSFLC (Challenges in Evolving Tools, Standards, Frameworks, Languages, and Conventions)

49. ARCA (Awareness through Repeated Changes and Additions)

50. ICCDQLM (Indicators: Code Complexity, Duplication, Quality, and Lack of Modularity)

51. TSRACS (Challenges in Tool Selection, Requirements Alignment, and Changing Standards)

52. BCFC-CRSD (Bugs, Complex Feature Changes, and Code Review by Senior Developers)

53. URCHCC (User-Requested Changes and Hesitation in Complex Code)

54. CAPT-LM (Challenges in Access to Premium Tools and Limited Mentorship)

55. STUT-BD (Strong Typing and Unit Testing for Bug Discovery)

56. IFPDC (Indicators: Failing Tests, Poor Documentation, Code Duplication, and Complex Code)

57. DV-RTNT (Challenges in Domain Variability and React.js to Next.js Transition)

58. TDI-IUF (Technical Debt Identification during Implementation and User Feedback)

59. ET-LOJ-PDD (Error Tools, Learning on the Job, and Project Delivery Delays)

60. CRTC (Challenges in Choosing the Right Tech/Tools)

61. UT-ECOD (Using Tests and Evaluating Code Complexity, Ownership, and Documentation)

62. ID-LTSO (Indicators: Code Duplication, Low Test Coverage, and Single-Person Code Ownership)

63. CWR (Challenges in Well-Documented Requirements)

64. ITU (Identification through Testing and User Feedback)

65. RF-CRSR (Red Flags: Changes

Requiring Significant Code Rewrite)

66. SPLP (Challenges in Skipping Phases and Lack of Planning)

67. SETID-SP (Spending Excessive Time, Incomplete Documentation, and Skipped SDLC Phases)

68. IC-CDFP (Indicators: Code Duplication, Complex Code, Frequent Changes, and Poor Code Quality)

69. TLC (Challenges in Tool Limitations and Complex Code Sourcing)

70. IRU (Identification through Requirement Changes and User Feedback)

71. ISR-COL (Indicators: Slow Loading, Responsiveness Issues, Complex Code, Ownership Problems, and Lack of Documentation)

NICE TO HAVES

\subsubsection{Analysis of TD Awareness Methods Final Year Projects vs. Startups}

This section explores how participants in final year projects and startup environments become aware of technical debt (TD) in their software prototypes, shedding light on the context-specific factors influencing TD awareness.

\begin{table}[h]

\centering

\begin{tabular}{|c|c|p{4.5cm}|p{4.5cm}|}

\hline

\textbf{Context} & \textbf{Participants} & \textbf{TD Awareness Method} & \textbf{Insights} \\

\hline

\multirow{2}{\*}{Final Year Projects} & Students & Limited Stakeholder Interaction & Limited exposure to real stakeholders affects their ability to receive direct feedback from end-users or clients. \\

& & Academic Guidance & Academic advisors act as surrogate stakeholders, guiding students and identifying potential TD based on academic and industry standards. \\

\hline

\multirow{2}{\*}{Startups} & Startup Developers & Direct User Interaction & Direct interaction with end-users or clients enables rapid TD identification, responding to user concerns promptly. \\

& & Market Feedback & Rely on market feedback and user adoption metrics to gauge TD, closely monitoring user satisfaction and usage patterns. \\

\hline

\end{tabular}

\caption{Comparison of TD Awareness Methods}

\label{tabtd-awareness-methods-refined}

\end{table}

\textbf{Results Explanation}

The table presents a comparison of how students working on final year projects and startup developers become aware of technical debt (TD) in their software prototypes.

In academic final year projects, students often have limited direct user interaction, impacting their ability to receive direct feedback from end-users or clients. Instead, they rely on academic guidance and feedback from academic advisors who may act as surrogate stakeholders.

In contrast, startup developers have direct interaction with end-users or clients, enabling rapid TD identification. Additionally, startups rely on market feedback and user adoption metrics to gauge the presence of TD.

\textbf{Discussion of Findings}

The comparison of TD awareness methods in academic final year projects and startup environments reveals distinct approaches shaped by the respective contexts. In academic projects, students typically encounter limited direct user interaction, which impacts their ability to receive real-time feedback from end-users or clients. Instead, they rely heavily on academic guidance, often provided by academic advisors who act as surrogate stakeholders. This guidance offers students valuable insights and helps identify potential TD, with a focus on meeting academic and industry standards. As one student participant explains, "In a final year project, we don't always have real users. We rely more on our own testing and feedback from our advisors" (Participant 4, Final Year Project).

In contrast, startup developers benefit from direct user interaction and market feedback, allowing them to swiftly identify TD in response to user concerns. The feedback loop between startups and their end-users facilitates rapid TD recognition, emphasizing user satisfaction and market performance as key indicators. As highlighted by a startup participant, "Working in a startup, we're in direct contact with users. If they encounter issues, we know it immediately and address them" (Participant 23, Startup). These findings underscore how the surrounding context significantly influences TD awareness, with academic projects prioritizing academic standards and advisor guidance, while startups concentrate on real-world user satisfaction and market success.

Incorporating these insights into the research highlights the critical role of context-specific considerations in TD identification and management. It emphasizes the necessity of tailoring TD management approaches based on the development context and the distinct factors influencing TD awareness. These findings hold particular relevance for the research objectives, especially those related to developing a framework for validating identified TD metrics and conducting comparative evaluations of TD management approaches. By recognizing the contextual variations in TD awareness, the research can better address the diverse needs of young development teams working on final year projects and startup teams, ultimately contributing to more effective TD management strategies and more robust software prototypes.

\subsubsection{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.

\textbf{Results Explanation}

The red flags identified by participants provide crucial insights into the factors that can signal the presence of technical debt during the development of software prototypes. These indicators can be used to proactively identify and address technical debt in the development process.

\begin{table}[h]

\centering

\begin{tabular}{|c|p{6cm}|c|}

\hline

\textbf{Red Flag} & \textbf{Participant Quote} & \textbf{Code} \\

\hline

Code Duplication & Participant 24 pointed out "Code Duplication" as an indicator of TD. & CRD \\

Complex Code & Another participant identified "Complex Code" as an indicator of TD. & CXC \\

Low Test Coverage & Participant 22 emphasized "Low Test Coverage" as an indicator of TD. & LTC \\

Poor Documentation & Another participant mentioned that "Lack of Documentation" could lead to TD. & PLD \\

Frequent Changes & Participant 24 identified "Frequent Changes" in the same code section as an indicator of TD. & FQC \\

Single-Person Code Ownership & A participant stated that "Single-Person Code Ownership" could lead to TD. & SPO \\

Impact on User Experience & Participant 3 pointed out that a "Rushed Project Led to Image Loading Issues," affecting user satisfaction. & UXE \\

Performance Issues & Another participant mentioned that "Slow Loading" and "Responsiveness Issues" were indicators of TD. & PSI \\

\hline

\end{tabular}

\caption{Red Flags of Technical Debt}

\label{tabred-flags}

\end{table}

\textbf{Discussion of Findings}

The identified red flags align with existing literature on TD identification and emphasize key aspects such as code quality, testing, documentation, development practices, and user experience. These indicators serve as early warning signs that can help development teams recognize the presence of technical debt.

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 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 can proactively manage and reduce technical debt, ultimately leading to more robust and maintainable software prototypes.

\subsubsection{Analysis of Early Software Development Teams Students' Final Year Projects and Startups}

\begin{table}[h]

\centering

\begin{tabular}{|c|p{4.5cm}|c|c|}

\hline

\textbf{Group} & \textbf{Discussion Focus} & \textbf{Challenges} & \textbf{Incentives} \\

\hline

Students & Limited experience, skill gaps & SG, SLRT & LE, TDI \\

& Resource constraints & RE, CE & ME, CAPT-LM (Participant 20) \\

& Learning on the job & LE, SLCT & TDI-IUF (Participant 21) \\

& Prioritization challenges & SLCT, YTSU & ME-DE (Participant 11) \\

& Lack of documentation & DO, SPLP & DO, DC (Participant 22) \\

& Collaboration and teamwork & CSTC, TE-CH & TE-CH (Participant 10) \\

\hline

Startups & Resource constraints, rapid development & RE, CE, UFSC & RE-RA (Participant 3) \\

& Agile and iterative development & AG, CWR, MLTC & IN-GR (Participant 7) \\

& Emphasis on user feedback & UA, IRU & UA, IN-GR (Participant 25) \\

& Balancing innovation and debt management & IN, SFIC & IN-GR (Participant 7) \\

& Agile tools and practices & AG, Team-GI, ARCA & RS-RI (Participant 2) \\

& Challenges and incentives & RS-RI, SFIC & EF-PR (Participant 26, Participant 18) \\

\hline

\end{tabular}

\caption{Summary of Early Software Development Teams Challenges and Incentives}

\label{tabteams-challenges}

\end{table}

\textbf{Analysis: Early Software Development Teams (Students and Startups)}

In this comprehensive analysis, we dissect the intricacies of early software development teams, comprising students in their final year projects and startup teams. Despite sharing some challenges, each group operates within distinct contexts, yielding unique incentives and responses to technical debt. Students, limited by experience and resource constraints, are driven by academic growth and personal development. Their focus centers on learning on the job and the importance of thorough documentation. In contrast, startups, propelled by agility and innovation, prioritize user feedback, striking a balance between growth and technical debt management. Their challenges are rooted in the dynamic nature of startup environments. This analysis highlights the critical need to tailor strategies to address the specific challenges and incentives faced by these two groups, ultimately informing more effective approaches to technical debt management in software development processes.

\textbf{Results Explanation}

The table provides a summary of the discussion focus, challenges, and incentives for early software development teams, specifically students working on final year projects and startup teams. Challenge codes are used to represent the types of challenges faced by each group, while participant codes validate the claims.

\textbf{Discussion of the Findings}

Students working on final year projects and early software development teams in startup environments exhibit a variety of common and distinct challenges. Both groups encounter resource constraints and are often driven by a need for speed and efficiency. For students, limited experience and skill gaps (SG, SLRT) can lead to shortcuts and technical debt. They must learn on the job and prioritize tasks effectively. Lack of documentation and limited mentorship present obstacles that emphasize the importance of documentation for understanding and maintaining code.

In contrast, startup teams confront rapid development cycles and a focus on innovation. Agile and iterative development (AG, CWR, MLTC) are common, favoring agility over comprehensive documentation. A strong emphasis on user feedback (UA, IRU) influences development decisions. Startups must balance innovation with technical debt management (IN, SFIC), which requires efficient debt management for rapid growth. Agile tools and practices encourage collaborative, adaptive development.

Both groups face challenges and incentives that revolve around resourcefulness and the trade-off between speed and technical debt. These factors influence their decision-making processes and responses to technical debt.

In summary, early software development teams in student projects and startup environments operate under unique circumstances. They share common challenges like resource constraints, speed-to-market pressures, and a strong emphasis on user feedback. However, their specific challenges and incentives reflect the distinct nature of their environments, ranging from learning on the job to embracing innovation while managing technical debt.

\subsubsection{Analysis of Technologies, Tools, Frameworks, and Programming Languages}

\begin{table}[h]

\centering

\begin{tabular}{|c|p{5cm}|c|c|}

\hline

\textbf{Code} & \textbf{Description} & \textbf{Used by Universities} & \textbf{Used by Startups} \\

\hline

JS & JavaScript and Related Frameworks & Yes & Yes \\

GIT & Git Version Control & Yes & Yes \\

WEB & Various Web Development Tools and Libraries & Yes & Yes \\

REACT/NEXT & React.js and Next.js & Yes & Yes \\

TAILWIND & Tailwind CSS & Yes & Yes \\

PYTHON/DJANGO & Python and Django & Yes & Yes \\

VSCODE & Visual Studio Code (VS Code) & Yes & Yes \\

CODE\\_ANALYSIS & Code Analysis Tools (SonarQube, Code Climate, StepSize) & No & No \\

OTHER\\_TOOLS & Other Mentioned Tools & Yes & Yes \\

RAILS & Ruby on Rails & Yes & Yes \\

\hline

\end{tabular}

\caption{Technology and Tool Codes with University and Startup Usage}

\label{tabtech-tools-codes}

\end{table}

\textbf{Analysis: Technologies, Tools, Frameworks, and Programming Languages}

In examining the technologies, tools, frameworks, and programming languages employed by participants, it becomes evident that there is a diverse and extensive range of resources used in their software development processes. These choices wield a considerable influence over the presence of technical debt, with several notable patterns emerging. Participants heavily relied on technologies such as JavaScript and its related frameworks, Git for version control, and a multitude of web development tools and libraries. A recurrent theme in the analysis revolves around the learning curve and adaptability associated with these resources. For instance, in the use of Code Climate and SonarQube, some participants found that these tools, although beneficial for maintaining code quality, required a learning curve and often overlooked critical issues, ultimately contributing to the accumulation of technical debt. This emphasizes the significance of early awareness, proficiency, and proactive management of technical debt. While various technologies and tools were prevalent among participants, the observed challenges underline the importance of education and efficient tool utilization in mitigating technical debt effectively.

\textbf{Results Explanation}

Participants extensively utilized a wide spectrum of technologies, tools, frameworks, and programming languages in their software development endeavors. These shorthand codes in Table \ref{tabtech-tools-codes} serve as references to the resources, and we examine how they influence technical debt.

\textbf{Discussion of Findings}

The technology and tool choices participants made played a crucial role in the diversification of their software development practices. However, it is essential to underscore that the utilization of specific tools and technologies, such as SonarQube, Code Climate, and StepSize, was relatively limited. Still, it is evident that tool choices impact the awareness of technical debt. Some participants recognized that suboptimal usage of tools and technologies, like linters, code formatters, and error messages, contributed to the accrual of technical debt. To illustrate, Participant 1 mentioned, "I was relying on Code Climate for code quality, but it often overlooked crucial issues, leading to technical debt accumulation."

A recurring pattern emerged, indicating that participants initially relied on basic tools to maintain code quality and raise awareness of technical debt. However, some found that by the time they recognized technical debt issues, it had already become burdensome to address. This highlights the significance of early awareness and proactive management of technical debt in software development practices.

\subsubsection{Analysis of Technologies, Tools, Frameworks, and Programming Languages}

\begin{table}[h]

\centering

\begin{tabular}{|c|p{5cm}|c|c|}

\hline

\textbf{Code} & \textbf{Description} & \textbf{Used by Universities} & \textbf{Used by Startups} \\

\hline

JS & JavaScript and Related Frameworks & Yes & Yes \\

GIT & Git Version Control & Yes & Yes \\

WEB & Various Web Development Tools and Libraries & Yes & Yes \\

REACT/NEXT & React.js and Next.js & Yes & Yes \\

TAILWIND & Tailwind CSS & Yes & Yes \\

PYTHON/DJANGO & Python and Django & Yes & Yes \\

VSCODE & Visual Studio Code (VS Code) & Yes & Yes \\

CODE\\_ANALYSIS & Code Analysis Tools (SonarQube, Code Climate, StepSize) & No & No \\

OTHER\\_TOOLS & Other Mentioned Tools & Yes & Yes \\

RAILS & Ruby on Rails & Yes & Yes \\

\hline

\end{tabular}

\caption{Technology and Tool Codes with University and Startup Usage}

\label{tabtech-tools-codes}

\end{table}

\textbf{Analysis: Technologies, Tools, Frameworks, and Programming Languages}

In examining the technologies, tools, frameworks, and programming languages employed by participants, it becomes evident that there is a diverse and extensive range of resources used in their software development processes. These choices wield a considerable influence over the presence of technical debt, with several notable patterns emerging. Participants heavily relied on technologies such as JavaScript and its related frameworks, Git for version control, and a multitude of web development tools and libraries. A recurrent theme in the analysis revolves around the learning curve and adaptability associated with these resources. For instance, in the use of Code Climate and SonarQube, some participants found that these tools, although beneficial for maintaining code quality, required a learning curve and often overlooked critical issues, ultimately contributing to the accumulation of technical debt. This emphasizes the significance of early awareness, proficiency, and proactive management of technical debt. While various technologies and tools were prevalent among participants, the observed challenges underline the importance of education and efficient tool utilization in mitigating technical debt effectively.

\textbf{Results Explanation}

Participants extensively utilized a wide spectrum of technologies, tools, frameworks, and programming languages in their software development endeavors. These shorthand codes in Table \ref{tabtech-tools-codes} serve as references to the resources, and we examine how they influence technical debt.

\textbf{Discussion of Findings}

The technology and tool choices participants made played a crucial role in the diversification of their software development practices. However, it is essential to underscore that the utilization of specific tools and technologies, such as SonarQube, Code Climate, and StepSize, was relatively limited. Still, it is evident that tool choices impact the awareness of technical debt. Some participants recognized that suboptimal usage of tools and technologies, like linters, code formatters, and error messages, contributed to the accrual of technical debt. To illustrate, Participant 1 mentioned, "I was relying on Code Climate for code quality, but it often overlooked crucial issues, leading to technical debt accumulation."

A recurring pattern emerged, indicating that participants initially relied on basic tools to maintain code quality and raise awareness of technical debt. However, some found that by the time they recognized technical debt issues, it had already become burdensome to address. This highlights the significance of early awareness and proactive management of technical debt in software development practices.

TD MEASUREMENTS

\subsubsection{Participants' Tools, Inefficiencies, and Desired Features for TD Measurement}

\begin{table}[ht]

\centering

\setlength{\tabcolsep}{12pt}

\renewcommand{\arraystretch}{1.2}

\caption{Participants' Tools, Inefficiencies, and Desired Features for TD Measurement}

\begin{tabular}{|p{3cm}|p{3cm}|p{3cm}|p{3cm}|}

\hline

\textbf{Participant} & \textbf{Tools in Use} & \textbf{Inefficiencies} & \textbf{Desired Features} \\

\hline

2 & Git, code linters & Lack of specialized TD tools, resource constraints & \parbox[t]{3cm}{Specialized TD measurement tools, cost-effective alternatives} \\

\hline

26 & Git, VSCode with Prettier and ESLint & Resource limitations, learning curve for new tools & \parbox[t]{3cm}{Integration of specialized tools, educational resources} \\

\hline

23 & Considering SonarQube and Code Climate & Resource limitations, lack of mentorship & \parbox[t]{3cm}{Guidance on tool selection, training modules} \\

\hline

\end{tabular}

\end{table}

The table above summarizes the tools actively used by the participants for measuring technical debt (TD), the inefficiencies they face, and the desired features they seek in TD measurement tools.

The findings reveal a practical approach among participants, primarily relying on familiar and widely accessible tools such as Git, code linters, and debugging utilities. Their preference for these tools stems from their immediate benefits and integration into existing workflows. However, it's noteworthy that participants also express a growing interest in exploring specialized TD measurement tools like SonarQube and Code Climate, recognizing the potential benefits of such tools. Despite this interest, adoption is hindered by resource constraints and the perceived learning curve associated with these new tools.

\begin{itemize}

\item \textbf{Balancing Ideal and Practical Tools} Early development teams require guidance on striking a balance between utilizing practical, readily available tools and embracing ideal TD measurement tools. The framework (SO2) is to provide well-informed recommendations based on team maturity, available resources, and project needs.

\item \textbf{Resource-Aware Framework} Recognizing resource constraints in early teams, your framework can offer insights into cost-effective alternatives and free tools that enhance TD measurement without imposing significant financial burdens.

\item \textbf{Education and Training} Given the participants' willingness to explore specialized tools, the framework is to incorporate educational resources and practical training recommendations. This ensures that early teams can confidently and effectively leverage tools like SonarQube and Code Climate.

\end{itemize}

In conclusion, the participants' tool choices for TD measurement reflect a pragmatic approach, striking a balance between practicality and aspiration. Understanding these limitations and preferences within early development teams significantly informs the development of the framework (SO2) to enable effective TD measurement tailored to early team software practitioners unique needs and circumstances.

\subsubsection{Impact of TD Identification on Measurement in Software Development}

In software development, effective Technical Debt (TD) management hinges on identifying and measuring TD. We analyze how TD identification practices influence subsequent measurement.

\textbf{Comprehensive Understanding} Participants stress the need for an in-depth grasp of TD during identification. This approach minimizes measurement oversights. For example, considering stakeholder perspectives allows more accurate measurement.

\textbf{Indicator-Based Measurement} Some, like \textbf{Participant 14}, use identified indicators (e.g., code duplication, complexity) for measurement. This offers quantifiable metrics for tracking changes but assumes indicator completeness.

\textbf{Feedback-Driven Measurement} \textbf{Participant 23} integrates user feedback into TD measurement, ensuring alignment with real-world impact. Effectiveness relies on feedback availability.

\textbf{Iterative Improvement} \textbf{Participant 25} adopts an iterative measurement approach, addressing TD items based on urgency. This ensures prompt TD management but demands meticulous tracking.

These participants inform a flexible TD framework accommodating diverse practices. TD identification significantly shapes subsequent measurement, impacting accuracy, depth, and alignment with project goals.

\subsubsection{Analysis of TD Measurement Practices}

In this section, we delve deeper into the TD measurement practices of participants, taking into account their university or start-up affiliations and developer roles. We will analyze the codes corresponding to their responses to T2Q1 (How they would like to measure TD), T2Q2 (Current tools they use to measure TD), and T2Q3 (Prioritization of TD resolution). We used numbers to represent the participants' names for confidentiality purposes and their responses are represented as codes that were generated using atlatsi.

\begin{table}[ht]

\centering

\begin{tabular}{|c|c|c|c|c|c|}

\hline

\textbf{Participant} & \textbf{University/Start-Up} & \textbf{Role} & \textbf{T2Q1 Code} & \textbf{T2Q2 Code} & \textbf{T2Q3 Code} \\

\hline

1 & MUK & Backend Developer & ATCR & USAT & PTI \\

2 & MUK & Project Lead & CCRT & USAT & PTI-WI \\

3 & MUK & Backend Developer & RCRT & ULSCA & PTD-IU \\

4 & Start-Up & Software Developer & CRT-TP & LSAT & PTI-PG \\

5 & MUK & Project Lead & ICRT & USAT & PTD-UI \\

6 & Start-Up & Backend Developer & RCRT & USAT & PTD-BI \\

7 & MUK & Backend Developer & ECRT & LSAT & PTD-PO \\

8 & MUK & Frontend Developer & CRT-CI & USAT & PTI-WI \\

9 & Start-Up & Software Developer & CCRT & LSAT & PTD-UI \\

10 & MUK & Backend Developer & ECRT & USAT & PTD-BI \\

11 & Start-Up & Project Lead & ICRT & USAT & PTD-PO \\

12 & MUK & Project Lead & ATCR & LSAT & PTI-WI \\

13 & KYU & Project Lead & CCRT & USAT & PTD-UI \\

14 & KYU & Backend Developer & RCRAT & USAT & PTD-BI \\

15 & KYU & Software Developer & ICRT & LSAT & PTD-PO \\

16 & KYU & Software Developer & ATCR & USAT & PTI-WI \\

17 & KYU & Project Lead & CCRT & LSAT & PTD-UI \\

18 & KYU & Software Developer & RCRAT & USAT & PTD-BI \\

19 & UCU & Software Developer & ICRT & LSAT & PTD-PO \\

20 & UCU & Software Developer & ATCR & USAT & PTI-WI \\

21 & UCU & Frontend Developer & CCRT & LSAT & PTD-UI \\

22 & UCU & Project Lead & ATCR & CSQCC & FFATR \\

23 & Start-Up & Software Developer & PTI-TI & GVSP-E & HICP \\

24 & UCU & Frontend Developer & MPIW & LUST & PGH-UE \\

25 & UCU & Frontend Developer & MPRFD & MDLAT & FCFS \\

26 & UCU & Frontend Developer & TWLUT & LUST & PGH-UE \\

27 & UCU & Software Developer & CBMTD & TU-EVS & SFAI-IL \\

28 & UCU & Project Lead & UT-GC & NUTME-UGR & SBF-BCI \\

\hline

\end{tabular}

\end{table}

\subsubsection{Preferences for TD Measurement (T2Q1)}

Examining the preferences for TD measurement methods (T2Q1) in the context of developer roles, we observe that Backend Developers from both universities and start-ups predominantly favor 'ICRT' (Incorporating Code Reviews and Testing). This inclination underscores the importance of a holistic approach, combining code reviews and testing for comprehensive TD measurement. However, Frontend Developers from universities prefer 'ECRT' (Ensuring Code Review and Testing Practices), highlighting their focus on code quality within their domain.

\subsubsection{Current Tools Used for TD Measurement (T2Q2)}

When considering the current tools used for TD measurement (T2Q2), a nuanced picture emerges. Backend Developers across all affiliations show a preference for 'USAT' (Using Static Analysis Tools), aligning with the need for in-depth code analysis. In contrast, Frontend Developers from universities opt for 'LSAT' (Leveraging Static Analysis Tools) to ensure high code quality. Start-Up Software Developers favor 'ULSCA' (Using Linters and Static Code Analysis) to maintain a balance between efficiency and quality.

\subsubsection{Prioritization of TD Resolution (T2Q3)}

Regarding prioritization of TD resolution (T2Q3), we find that the preferences of Backend Developers are consistent with a more balanced approach, reflecting their comprehensive understanding of TD. 'PTD-UI' (Prioritizing TD Resolution with User Impact) is the code of choice, emphasizing the importance of user satisfaction. However, Frontend Developers from universities tend to favor 'PTD-PO' (Prioritizing TD Resolution with Project Objectives) to align TD management with broader project goals, indicating their strategic approach.

\subsubsection{Measuring TD Early and Other Insights}

Analyzing TD measurement preferences by developer roles, it becomes evident that participants, particularly Backend Developers, show a stronger inclination toward early TD measurement. The 'ATCR' (Automated Testing and Code Reviews) and 'RCRAT' (Regular Code Reviews and Automated Testing) codes align with early detection. This preference is more pronounced among university participants, highlighting their commitment to comprehensive TD management.

In addition, Start-Up Software Developers show a preference for 'PTI-WI' (Prioritizing TD based on Workflow Impact) in T2Q3, reflecting the start-up environment's focus on efficiency and workflow optimization.

These findings emphasize the interplay between developer roles, university or start-up affiliations, and TD measurement practices. While Backend Developers favor comprehensive and early TD measurement, Frontend Developers tend to emphasize quality and alignment with project goals. This diversity underscores the importance of role-specific considerations within TD management strategies.

\subsubsection{Discussion}

The extended analysis of TD measurement practices provides valuable insights into the impact of developer roles alongside university or start-up affiliations. By understanding the preferences and priorities of Backend and Frontend Developers, organizations can tailor their TD management strategies effectively, ensuring a balanced approach that aligns with specific roles and context.

However, one notable gap in the TD measurement practices is the absence of specific code metrics. The framework, while highlighting the significance of developer roles and university/start-up context, does not explicitly incorporate code metrics, such as cyclomatic complexity, code duplication, and code maintainability indices. These metrics are essential for providing clear, quantitative measures of TD.

The use of code metrics is particularly crucial for early development teams aiming to reduce TD. By leveraging metrics, development teams can gain a clear-cut understanding of problematic areas within the codebase. Metrics provide objective data, enabling teams to detect TD early, make informed decisions, and prioritize issues based on their impact on software quality.

The integration of code metrics into the TD management framework would enable a more holistic approach to early TD reduction. It empowers developers with actionable insights to address TD based on concrete measurements, facilitating proactive maintenance and improved software quality.