# A Framework for identifying and evaluating process and product technical debt in early software prototypes: A case of early-stage software development teams

Area of study: Technical Debt, Software Development Processes

Supervisors: Dr. Grace Kamulegeya and Dr. Benjamin Kanagwa

By Mugoya Dihfahsih
Draft Research Proposal

## 1 Introduction and background of the study

Technical Debt (Technical gap) is a metaphor coined by Cunningham [5] to represent sub-optimal design or implementation solutions that yield a benefit in the short term but make changes more costly or even impossible in the medium to long term [2]. It describes what results when development teams take actions to expedite the delivery of a piece of functionality or a project which later needs to be refactored.
The technical debt originates solely from the requirements analysis through software implementation [6] up to the last phase of the software development life cycle. There exists risks in student projects due to poor requirements analysis [10], lack of skills in using technologies and poor documentation, all these and other factors lead into a technical debt of projects by these young teams. It is very common for software engineering student projects to incur technical debt during the development process because of many reasons but mainly due to prevalent [20]shortcuts. However, its presence in student projects can lead to risks [2] [1] that degrade the internal quality of the product which affects the future rework on the project making it so costly . The effects of the technical debt can be identified in different stages of software development due to different types of debt. Low quality, delivery delay, low maintainability, rework and financial loss are among the 10 most commonly effects of Technical debt [32] [34] .

Identifying the likelihood of Technical debt as early as possible [3] helps in managing the risks that could cost the project. The exploratory study [1] reports that developer code smells or anti-patterns contribute a lot to the occurrence of technical debt and these behaviours are found mostly in young teams that are experimenting with different technologies. The systematic literature review on technical debt [10] presents a report on costs of not following proper development processes and strategies in software projects thus causing different types of technical debt which are costly [7] to manage if not dealt with in early project prototypes.

Software engineering students are required to develop innovative projects [26] as part of their final grading. Other students work on university projects that are assigned to them by their lecturers. These projects most times end up with gaps [36] that are likely to cause project failure because of many reasons but one of them being student amateur knowledge about software development processes taking shortcuts to get the project done in the scheduled time frame. Without knowing the repercussions of these shortcuts, if not tamed early, student practitioners are likely to extend such vices to the industrial projects which can be costly for any business. The main contribution of this research is to develop a framework that enables students and their supervisors to identify and evaluate these gaps in software development processes of these young teams as early as possible.

## 1.1 Early-stage Software Development teams

These are considered to be the young teams, amateur software practitioners, ones with limited programming knowledge; they are also referred to as novice developers [8], sophomores or juniors who are specifically university students for the purpose of this study that work on innovative final year projects [24]. Software engineering students are supposed to learn engineering methods, processes, techniques, and measurement. They benefit from the use of tools for[17] managing software development, analysing and modelling software artefacts, assessing and controlling quality, and for ensuring a disciplined, controlled approach to software evolution and reuse. These junior developers are supposed to identify ideas for final year projects whose product is a [29] Real time software, Embedded software, System Software, Business software, Scientific and engineering software.

Universities and other higher institutions of learning require software engineering students to demonstrate independent skills in implementing non-trivial software engineering or research projects by pursuing lengthy supervised projects to introduce concepts of software development processes such as documenting, software designing, testing and deploying well engineered solutions.The purpose. These projects are to prepare students for absorption into mainstream industrial projects Professors, lecturers or advisors or supervisors assist to guide these students to deliver projects using state of the art technologies. The projects have a [15]short time frame in which students should deliver working prototypes as a result the students who are at early stages of software development take shortcuts to complete projects on time and graduate thus delivering projects that are costly to refactor when adding new features. These shortcuts affect the [15] [29] [2] quality of software products as students plan tests, schedule refactoring, code reviews which they do not implement. Therefore students working on these projects[14] that utilise software development processes must be aware of the effects of taking shortcuts. This study will be advantageous to the students as it proposes a framework that will enable both students and their supervisors to identify and evaluate the occurrence of technical debt and when to pay off the debts in such projects as early as possible to minimise the costs that are incurred in late repayment.

## 1.2 Early software prototypes

A software prototype is a simulation of how the actual project will look, work, and feel and would not have an exact logic involved in the software development phase. Young software development teams, especially the students, use it for [11] right guidance while working on their projects. Student prototypes are the proof of concepts that enables the supervisors to verify that their design for the project will work as intended.

Early software prototypes are meant to enable students to get the projects the right way at inception, to reduce risks, allow detecting errors and save project time. Prototyping becomes a challenge as most students since they are novice developers choose the wrong prototyping process, instead of throw-away prototypes, some choose evolutionary or incremental prototypes that are time consuming, this reduces project time that results in taking shortcuts to meet the project deadline.

Prototypes have to be refined after tests, feedback from the supervisors and other product stakeholders, students always ignore this and implement the suggested changes on the real product instead thus rendering original prototypes prone to errors if used as a baseline for developing the product. Students also lack enough requirements analysis knowledge to develop correct prototypes for the project, this costs the project in future since there has to be rework to capture features that were not included in the prototypes. Not knowing the importance of early prototypes, students after presenting prototypes to supervisors and getting approval, most times they don't keep the original prototype, they go ahead to modify the prototype into a product and thus they have no baseline for their projects which is so costly in evaluating the product features.

Prototypes are intended to minimise project risks such as technical debt, students and other novice developers create vague prototypes or spend less on them as they think they are time consuming and rather spend much lots of time on designing the real product, this makes the project prone to taking sub-optimal decisions like use of untested apis, libraries, wrong system architecture which all end up in technical debt.

## 1.3 Technical debt in student project processes

From student software development processes, the following are some of the sources of technical debt [16];

- Code debt - Usually related to poor or speedy coding or not having applied appropriate design patterns.
- Defect debt - Existing bugs, issues, and defects in the product.
- Design debt - It is all the good design concepts or solutions that were skipped in order to reach a short-time goal quicker.
- Requirement debt - Incurred during the identification, formalisation, and implementation of requirements
- Test debt - This happens when the team starts to spend more time on test maintenance and less on creating tests to ensure the software is bug-free.
- Architecture debt - Some architectural outline decisions incur technical debt, which can negatively impact the quality of a software project.
- Documentation debt - This type of technical debt describes problems in documentation such as missing, inadequate, or incomplete papers.
- Process debt - Usually generated when processes are poor or lacking altogether to handle defects, documentation, or even tests.

Process Technical Debt is also as a result of the inefficiencies, wastes, and redundancies that accumulate in student projects in [7] processes or workflows over a given period of time. The university software engineering students' final outcome or product of the project is always a system developed using tools such as UML case tools, CASE, CAD, IPSE, Eclipse, User modelling language, IDE, .Net framework , Python, PhP, JAVA and other technologies. A combination of these tools are used to develop a working system that can be tested and verified by the supervisor for grading. Students are supposed to follow Software Development Life Cycle processes in developing a product. These processes include Planning, requirement analysis, design, implementation, testing and integration and maintenance.

There are several other processes that software teams have to follow and software practitioners including experienced developers, university students, or novice developers have to rely on information repositories such as documents, standards, and policies. Proper following of these processes[8] results into a [10] quality product that has less flaws or bugs on contrary, if students do not adhere to proper workflows or processes such as testing, documenting, understanding of codebase and instead prioritise speedy patchwork fixes[9], the result is always a product with poor internal qualities that are costly to refactor in future.

Approximately 50 percent of product defects originate from requirements. And 80 percent of future project rework can be traced to requirements defects. According to one report on the impact of project [26] requirements, developers spend over 41.5 percent of their new project development resources on unnecessary or poorly specified requirements which can gravely appear in downstream activities, including architecture, design, implementation and testing. Poor software requirements analysis can also create further technical problems resulting in poor user responsiveness, long delivery times, late deliveries, defects, rising development costs, and low developer morale. Process debt causes project failures and a lot of time is spent on rework if the debt is not paid off in a stipulated and well defined time frame in the life of the project.

### 1.3.1 Causes of technical debt in student projects

Technical debt is always incurred in any project however professional the developers can be as it does not only originate from the implementation or the coding but in several software development processes. Experienced developers who have faced the consequences of taking shortcuts have testified about how costly it can be to resolve a prolonged unpaid debt.

Novice developers who are in their early stages of development are not aware of such ramifications and they enjoy taking shortcuts in many of their software development processes thus causing technical debt. There are several causes of technical debt[30] [10] but the following are those that pertain to student projects.

- Aggressive project schedules with pressure from supervisor to have the changes implemented soon.
- Insufficient requirements gathering and analysis Overly complex technical design especially for those working on start up projects.
- Inadequate technical skills or talent skill gap.
- Sub optimal choices like last minute changes to design and specification documents.
- Lack of collaboration between young teams especially those working in groups.
- Insufficient testing or quality assurance and control which leads to quick bug fixes.

After some time, the above factors, if not dealt with by students, accumulate into technical inefficiencies that have to be serviced in future. This makes the project more expensive to change thus meeting its death point.

### 1.3.2 Conventional ways of how students amateurly handle technical Debt

Students find it extremely hard to minimise technical issues due to lack of knowledge about it as well as lack of a designed framework to enable them to identify these gaps in their processes. Some of the observable ways students who have gained experienced minimise technical debt; pair programming with more senior developers, backlogging any item they suspect to compromise the projects and communicate with the team, using version control tools such as git to monitor code quality and easily rollback in case of any challenges with the project as well as carrying out simple unit tests mostly when students are about to deploy their products.

## 2 Significance of this research

The main contribution of this study is to build a framework using available [35]technical management models that will enable the young software teams to deal with technical debt in their project processes. Unlike industrial models that manage technical debt in a project when it has already occurred.

- This research proposes a model that will enable students to identify the likely activities that can cause technical debt in early prototype processes to minimise the costly refactoring and making the future rework so hard.
- This study provides a framework that can influence young teams' styles of implementing projects such as properly analysing user requirements, carrying out activities that help to ensure internal quality of the project product like reviews, timely refactoring and testing.
- The proposed framework will ensure that students' innovative project products see a day of light as they can be modified easily using well structured documentation, tested code thus making the product more maintainable.
- The study further provides a foundation on future studies on managing student technical debt such as developing automated tools that help students develop software products with minimal technical gaps.

## 3 Statement of the Problem

Young team practitioners, especially students, work on projects that have aggressive schedules to produce a working system or product and as a result, the practitioners tend to learn new technologies as they implement them on the project. This cripples on implementation time, consequently they opt for shortcuts by removing anything that is not code-able from their project plan such as code reviews, quality assurance and testing. This affects the internal quality of

the product making it hard to spot the occurrence of technical debt as early as possible. As the development process progresses, the technical debt takes the project hostage and the young practitioners take ages to complete features and fix bugs.
There is a need for a framework that can be used to identify these gaps in processes in project prototypes and evaluate them as early as possible to minimise the occurrence of technical debt on such projects. Such a framework can guide the software practitioners to be aware of the costs incurred on paying technical debt and the right time to pay this debt in the project.

Most studies about technical debt are focused on minimising the costs incurred in refactoring industrial projects [11] and there is little literature or no research about evaluating technical debt in university projects which are developed by young practitioners. These developers write partial project documentation and at times none, untested code methods and practise many anti-patterns such as using wrong architecture compromising the internal product quality which makes it hard to change in future leading to technical debt.

Technical debt in young team projects is like a virus which is hard to eradicate once it takes hold of these innovative projects [3] that students or junior practitioners work on. Without a framework to enable students to treat or deal with technical debt at early stages, it spreads rapidly across the entire codebase. Consequently, however innovative the project can be, the clean slate is always the only option as it is so costly to rework and fix the bugs in these compromised young team projects.

## 4 Literature review

The risks in the students or amateur software practitioners' projects [6] is due to lack of enough research conducted in evaluating amateur practitioners' processes which lead to project failure. The existing studies have proposed models [9] to manage the technical debt in projects.

There exists risks in student projects due to poor requirements analysis [21], lack of skills in using technologies and poor documentation, all these and other factors lead into a technical debt of a project by these young teams. Identifying the likelihood of Technical debt as early as possible [6] helps in managing the risks that could cost the project. The exploratory study [3] reports that developer code smells or anti-patterns contribute a lot to the occurrence of technical debt and these behaviours are found mostly in young teams that are experimenting with different technologies. The systematic literature review on technical debt [21] presents a report on costs of not following proper development processes and strategies in software projects thus causing different types of technical debt which are costly [18] to manage if not dealt with in early project prototypes.

Lenarduzzi at el propose the use of SonarQube [22] as a model to deal with bad code by junior developers. This model enables the novice practitioners to understand code quality and to learn coding skills with minimal consultation on code refactoring. The model does not address the entire software development process such as documentation. the causes of technical debt [5] are process decisions and not only the code, these decisions can be made before project implementation such as choice of architecture.

Fernandez et al. introduces a framework [14] that classifies the elements of technical debt into groups and then systematically maps them to stakeholders' point of view about technical debt. The study did not look at the experience of developers as one of items of technical debt and also the model proposed does not help to identify the processes that would lead to technical debt.

Marques et al. propose a model [23] enables students test their project code to improve the internal quality of the software by identifying the bugs that would cost the project in refactoring or adding of a new feature. However the framework only suits skillful developers designing algorithms that can be quantified through series of tests. Tests alone can not be enough to sustain code quality, this necessitates designing of a framework that will enable students choose processes that lead to speedy project implementation but do not make their projects complex in terms of paying off technical debt

The Strategic Technical Debt Management Model(STDMM) [9] was proposed to manage technical debt in a more structural way. The model is designed on three dimensions that is quality, time and budget. This model can be suitable for experienced practitioners, however it does not consider the project processes used by novice developers and also the excomment model [13].

Several studies have been pursued to identify and evaluate the elements required to manage technical debt. These studies have proposed taxonomies that are useful in understand the issues while dealing with technical debt, however a taxonomy is not more representative and elaborated as a model. The proposed frameworks by the studies are industrial [38] and do not address the case for students working on research or university projects.

This study examines different activities, techniques, strategies, standards, and policies used or applied by young software development teams to develop a framework that will identify gaps in what is done or missing to reduce technical debt in early prototype software projects by young or novice practitioners.

## 5 Objectives of the Study

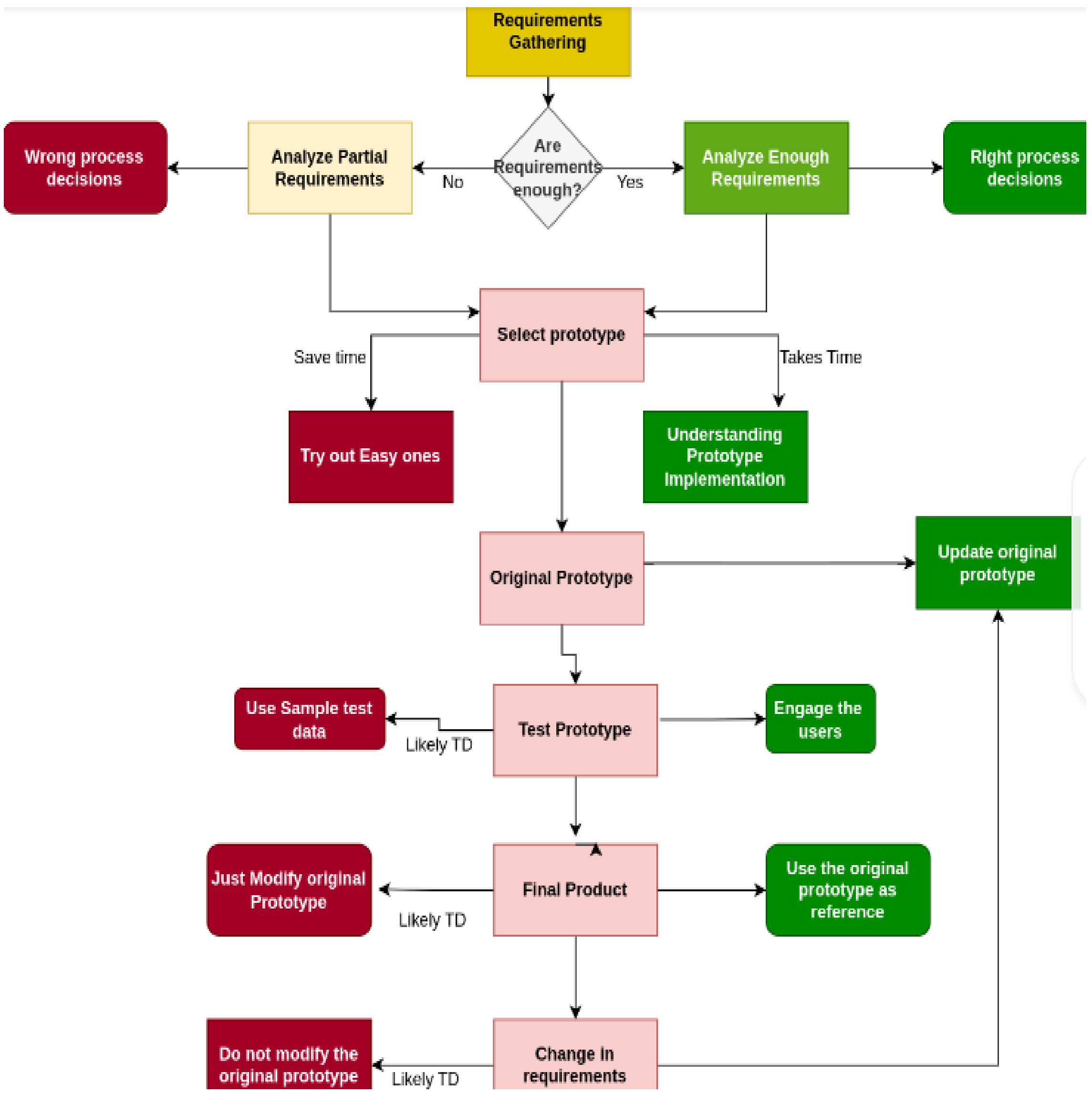
The main objective of this study is to build a model to minimize technical debt in software projects and reduce the costs on project refactoring.

### 5.1 Specific objectives

- To identify gaps in policies, processes or activities that student software practitioners use in order to get their project finished or graded.
- To enable the mitigation of the occurrence of the technical debt as early as possible in project prototypes and to be able to identify when to address the debt and the resources required in the repayment of the interest.
- To identify the relationship between the gaps in young practitioners' processes and the occurrence of technical debt.

## 6 Conceptual Framework

Many technical debt management studies are based on minimising the costs in industrial software projects[4] and most of these are as a result of young teams that are not aware of the effects of taking shortcuts in projects. This study evaluates the processes and technological gaps in projects by young teams to help mitigate the occurrence of technical debt in early software development stages. The figure1, shows the proposed framework to particularly identify the technical debt as early as prototypical inception by focusing on project requirements.



*F*igure1, A proposed model of identifying Technical debt in during early prototyping

## 7 Research Methodology

### 7.1 The Research Design

The research design for this research shall be a descriptive survey by using a qualitative approach [12] [25] in which interviews shall be conducted by a focused team of young software practitioners to collect data that is not available in the existing requirements documents, software development documents and other records. The interviews shall be structured in a way that they are unstructured or semi structured to allow collection of unexpected answers from the students as well as allowing the posing of open-ended questions and following the response lead. The responses shall be comprehensively used in interpreting results about the individual awareness of technical debt in their software development processes by comparing them to those who do not know about technical debt.

## 7.2 Research Questions to guide this study

Recent reports about TD [26] [19] indicate that TD is a well known terminology by industrial software practitioners, on contrary, there is [1] lack of enough research about technical debt in software projects by young teams that are transitioning into industrial practitioners. This knowledge gap can be closed by carrying out a study that looks at how [26] novices perceive and handle technical debt in their projects. The research design of this project is based on the previous literature about models that identify and manage TD [4] [27] [37] [33] [9] [31] [28] to develop a framework that will aid the students and supervisor spot TD elements in projects as early as prototype inception stage. This study seeks to answer the following research questions(RQ) as a way of examining the existence and evaluation of technical debt in processes of software development by young teams. This study seeks to answer the following research questions(RQ) as a way of understanding the existence and evaluation of technical debt in processes of software development by young teams.

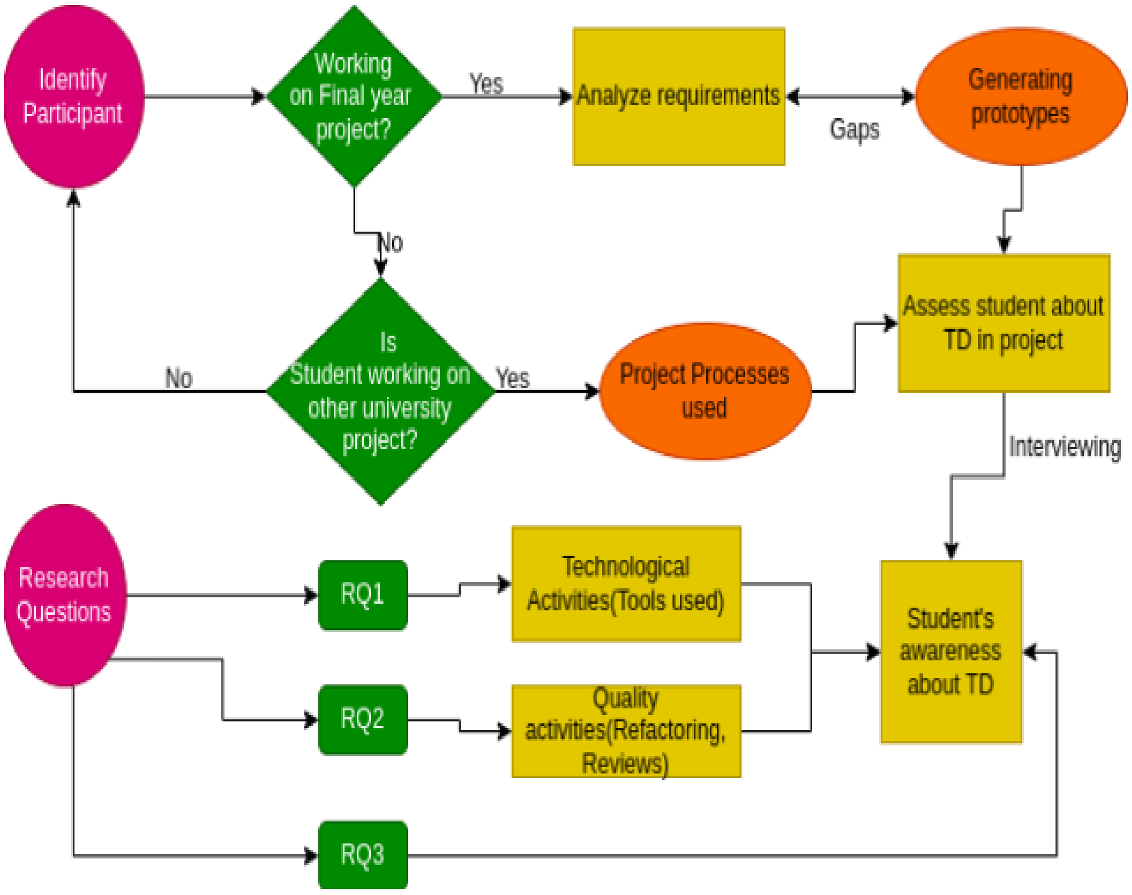
- RQ1: Are there gaps in processes or technological activities that software engineering students use on their projects?
- RQ2: When should the technical debt be repaid by students in project development?
- RQ3: What is the relationship between the techniques young teams use on software projects and occurrence of technical debt?

## 7.3 Data Collection

To answer the research questions RQ1, RQ2 and RQ3, a qualitative strategy [25] of investigation will be carried out to interpret the patterns, processes and activities of young teams on projects. Interview method will be used as a technique to collect data from the university students or novice practitioners. To get the student practitioner's views, interviews shall be administered to four leading information technology universities; Makerere University, Kampala University, Kyambogo University and Uganda Christian University. From each of the four universities at least 40 percent of the final year students shall be interviewed. This sampling population will be enough to aggregate the results and evaluate technical debt in university projects. The participants to be interviewed shall

be final year students working on their projects and also students working on on-going university research projects or university research start ups.
This qualitative research will also investigate the requirements processes to technological tools the students use such as editors, versioning tools, testing tools. Software development documents such as requirements, design, software specification documents shall be investigated to identify gaps that are likely to cause technical debt in projects. The participants shall be classified in groups of those who know about technical debt and those who do not know. This research shall be in person rather than sending numerous questionnaires to large numbers of respondents to ensure that the participants understand what is being investigated and for them to have a feeling of interacting with a human researcher rather than with an easily-discarded form.

### 7.3.1 The data collection process



Identify Participant
Working on Final year project?
Yes
Analyze requirements
Gaps
Generating prototypes
No
Assess student about TD in project
No
Is Student working on other university project?
Yes
Project Processes used
Interviewing
Research Questions
RQ1
Technological Activities(Tools used)
RQ2
Quality activities(Refactoring, Reviews)
Student's awareness about TD
RQ3

*F*igure2, A research process of how the study will be carried out

Figure2 conceptualises how this research shall be carried out, after identifying the geographical locations or universities where participants in the study will be found, then the participants will be chosen on the criteria that either they are working on a final year project or they are working on university projects known as innovative start ups.
Project requirements are the likely source of the Technical debt in student projects, therefore an emphasis on how requirements were collected and analysed will be pivotal in data collection. The data collection method shall mainly be interviewing project teams or individual developers on their role on the project. For cases where students are using the university start up projects as their final year projects, requirements processes and other documents shall be reviewed and assessed using the RQs of the study.
To investigate each of the research questions, students will be required to provide specific responses, these responses will later be analysed to formulate a framework that will provide a solution to the technical debt in student projects.

### 7.3.2 Identifying gaps in students' projects processes(RQ1)

RQ1 about the gaps in student development processes shall collect data that includes all activities such as requirements analysis for example what do students do when they realise requirements are not enough, do they progress with prototypes, do they recollect them or do they engage the users about. This will help to establish whether there is likelihood of technical debt in requirements analysis. Students will also be required to provide response on how do they come up with prototypes, do they always keep the original prototype or modify the original prototype into final product, do they always engage the supervisors about the right type of prototype for the project, what do the students do when the requirements change, do they document them and update the prototype or they simply update the final product with the changes. This will enable the identification of gaps in prototypes and requirements as early as possible before the implementation phase.

### 7.3.3 Student awareness about repayment of technical debt (RQ2)

To achieve this objective, students shall be required to provide information they know about the technical debt. Their response to this question shall trigger inform the which follow up questions, for example if the students are not aware about the effects of technical debt, then the follow up questions shall be whether they find it hard to fix errors or even identify bugs in the code, what do they do if they suspect that a certain decision is likely to negatively impact the project in the long run. Students will also be required to provide information about when they carry out code reviews, code refactoring and code testing. The students will provide a response to whether they willfully take shortcuts, and at what stage do they try to rectify these quick project fixes. This information shall help to determine whether students try to manage technical debt in their implementation. Students shall be asked if they would like to have a framework that enables them to minimise these technical gaps as early as possible.

#### 7.3.4 Relationship between technical debt and students activities (RQ3)

This objective is meant to verify that technical gaps in students' projects ultimately cause project failures. This will require students to provide responses on questions about the difficulties they face when adding new features to the product using partial project documentation, untested and reviewed code.

## 7.4 Data Analysis

In order to investigate the three research questions (RQ1, RQ2 and RQ3) mentioned in the hypothesis section, the data collected using the described methodology in data collection section shall be analysed so that the results of the analysis will enable the formulation of the framework to reduce the occurrence of technical debt in student projects in the following steps;

- In achieving this analysis association metrics shall be computed to measure the prevalence of technical debt in the student project processes.
- Correlating the students' awareness about dangers of taking shortcuts and the occurrence of technical debt.
- Analysing students' activities such as requirement analysis, code reviews, code refactoring, team collaboration and testing in ensuring the quality control of the software product. These quality attributes shall be ranked on a scale of 1 to 10 to identify which of the activities should be done first and when it should be used in the project.
- Visualise the analysed data through use of tables, charts and graphs to provide an insightful representation of the data.

#### 7.4.1 RQ1, Are there gaps in processes or technological activities that software engineering students use on their projects?

These gaps in students' processes shall be ranked in the order of criticality to inform the formulation of the framework to include those gaps that are most likely to cause a mayhem to the project. These gaps will be analysed by identifying the student's follow up questions on the shortcuts they take in order to complete the project. Activities such as requirements analysis, documentation and quality assurance controls of the project shall be ranked higher than any other attributes in this research.

#### 7.4.2 RQ2, When should the technical debt be repaid by students in project development?

This shall be analysed with ranking of the students' awareness about the effects of taking shortcuts on the projects. Students' response will be given scores with those that are not aware of consequences of shortcuts given higher scores compared to those that are aware. This is done such that a framework that includes students' awareness about technical debt can be modelled in consideration of technical debt novices. This shall be correlated with the student's ability to spot and manage the technical debt as well as the stage in the implementation phase where they can identify the occurrence of the debt.

#### 7.4.3 RQ3, What is the relationship between the techniques young teams use on software projects and occurrence of technical debt?

This research objective shall be analysed by indicators of mapping the RQ1 and RQ2, this will include analysing the technical debt elements in student projects to be used as a baseline for evaluating and correlating the relationship of students' processes and risks they pose to the project. This will be done with the help of visualisation tools such as graphs where technical debt elements will be plotted against the technical debt that was caused.

## 7.5 The scope of the study

Technical debt is incurred across all software projects from min projects to industrialised ones, this research will be focused on identifying and evaluating students' software development processes that are likely to cause technical debt in final year projects.

- Students are part of novice practitioners transitioning into senior level, therefore, the study will carry out the investigations in four universities in Uganda as mentioned in the research methodology. The participants shall be strictly those offering Bachelors degree in fields of Information systems, computer science and Software engineering.
- The sample space for each university shall be 40 percent of the desired population.
- Software projects have many processes from which technical debt can originate, this research will be constrained to examining only the requirement documents, prototypes, implementation and quality assurance activities of the project.
- The study shall only investigate students working on the final year projects, this will help to provide enough information on the proposed framework.
- The research will examine the individual efforts of the students to the project as well as the entire team contribution and assess the likely cause of technical. Data collection from the four universities shall take one month and data analysis will take one month to verify the research objectives

## References

[1] Tero Ahtee and Timo Poranen. Risks in students' software projects. In *2009 22nd Conference on Software Engineering Education and Training*, pages 154–157. IEEE, 2009.

[2] Mohammed Abdullah Hassan Al-Hagery. Problems discovery of final graduation projects during the software development processes. *International Journal of Computer Applications*, 975:8887, 2012.

[3] Reem Alfayez, Pooyan Behnamghader, Kamonphop Srisopha, and Barry Boehm. An exploratory study on the influence of developers in technical debt. In *Proceedings of the 2018 international conference on technical debt*, pages 1–10, 2018.

[4] Abdulaziz Alhefdhi, Hoa Khanh Dam, Yusuf Sulistyo Nugroho, Hideaki Hata, Takashi Ishio, and Aditya Ghose. A framework for conditional statement technical debt identification and description. *Automated Software Engineering*, 29(2):1–36, 2022.

[5] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[6] Barry W. Boehm. Software risk management: principles and practices. *IEEE software*, 8(1):32–41, 1991.

[7] Gustavo Caiza, Fernando Ibarra Torres, and Marcelo V Garcia. Identification of patterns in the involvement of novice software developers in software testing processes. In *2021 IEEE international conference on artificial intelligence and computer applications (ICAICA)*, pages 378–382. IEEE, 2021.

[8] Woei-Kae Chen, Chien-Hung Liu, and Hong-Ming Huang. Software debugging patterns for novice programmers. 2017.

[9] Paolo Ciancarini and Daniel Russo. The strategic technical debt management model: an empirical proposal. In *IFIP International Conference on Open Source Systems*, pages 131–140. Springer, 2020.

[10] Zadia Codabux, Byron J Williams, and Nan Niu. A quality assurance approach to technical debt. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2014.

[11] Yania Crespo, Arturo Gonzalez-Escribano, and Mario Piattini. Carrot and stick approaches revisited when managing technical debt in an educational context. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 99–108. IEEE, 2021.

[12] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage publications, 2017.

[13] Mário André de Freitas Farias, Manoel Gomes de Mendonça Neto, André Batista da Silva, and Rodrigo Oliveira Spínola. A contextualized vocabulary model for identifying technical debt on code comments. In *2015 IEEE 7th international workshop on managing technical debt (MTD)*, pages 25–32. IEEE, 2015.

[14] Carlos Fernández-Sánchez, Juan Garbajosa, and Agustín Yagüe. A framework to aid in decision making for technical debt management. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 69–76. IEEE, 2015.

[15] Kresimir Fertalj, B Milašinović, and I Nižetić. Problems and experiences with student projects based on real-world problems: a case study. *Technics Technologies Education Management*, 8(1):176–186, 2013.

[16] Martin Fowler. Technical debt quadrant. *Martin Fowler*, pages 14–0, 2009.

[17] Lakshmi Ganesh. Board game as a tool to teach software engineering concept–technical debt. In *2014 IEEE sixth international conference on technology for education*, pages 44–47. IEEE, 2014.

[18] Yuepu Guo, Rodrigo Oliveira Spínola, and Carolyn Seaman. Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, 21(1):159–182, 2016.

[19] Johannes Holvitie, Sherlock A Licorish, Rodrigo O Spínola, Sami Hyrynsalmi, Stephen G MacDonell, Thiago S Mendes, Jim Buchan, and Ville Leppänen. Technical debt and agile software development practices and processes: An industry practitioner survey. *Information and Software Technology*, 96:141–160, 2018.

[20] Tim Klinger, Peri Tarr, Patrick Wagstrom, and Clay Williams. An enterprise perspective on technical debt. In *Proceedings of the 2nd Workshop on managing technical debt*, pages 35–38, 2011.

[21] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, and Francesca Arcelli Fontana. A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, 171:110827, 2021.

[22] Valentina Lenarduzzi, Vladimir Mandić, Andrej Katin, and Davide Taibi. How long do junior developers take to remove technical debt items? In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2020.

[23] Filipe Marques, António Morgado, José Fragoso Santos, and Mikoláš Janota. Testselector: Automatic test suite selection for student projects–extended version. *arXiv preprint arXiv:2207.09509*, 2022.

[24] Maira Marques, Sergio F Ochoa, Maria Cecilia Bastarrica, and Francisco J Gutierrez. Enhancing the student learning experience in software engineering project courses. *IEEE Transactions on Education*, 61(1):63–73, 2017.

[25] Martin N Marshall. Sampling for qualitative research. *Family practice*, 13(6):522–526, 1996.

[26] William W McMillan and Sudha Rajaprabhakaran. What leading practitioners say should be emphasized in students' software engineering projects. In *Proceedings 12th Conference on Software Engineering Education and Training (Cat. No. PR00131)*, pages 177–185. IEEE, 1999.

[27] Ariadi Nugroho, Joost Visser, and Tobias Kuipers. An empirical model of technical debt and interest. In *Proceedings of the 2nd workshop on managing technical debt*, pages 1–8, 2011.

[28] Sameer S Paradkar. Framework and techniques for managing technical debt in software development lifecycle. *Global Journal of Enterprise Information System*, 13(1):71–77, 2021.

[29] Alex Radermacher, Gursimran Walia, and Dean Knudson. Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th international conference on software engineering*, pages 291–300, 2014.

[30] Robert Ramač, Vladimir Mandić, Nebojša Taušan, Nicolli Rios, Sávio Freire, Boris Pérez, Camilo Castellanos, Darío Correal, Alexia Pacheco, Gustavo Lopez, et al. Prevalence, common causes and effects of technical debt: Results from a family of surveys with the it industry. *Journal of Systems and Software*, 184:111114, 2022.

[31] Narayan Ramasubbu and Chris F Kemerer. Integrating technical debt management and software quality management processes: A normative framework and field tests. *IEEE Transactions on Software Engineering*, 45(3):285–300, 2017.

[32] Nicolli Rios, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology*, 102:117–145, 2018.

[33] Nicolli Rios, Savio Freire, Boris Perez, Camilo Castellanos, Dario Correal, Manoel Mendonca, Davide Falessi, Clemente Izurieta, Carolyn B Seaman, and Rodrigo Oliveira Spinola. On the relationship between technical debt management and process models. *IEEE Software*, 38(5):56–64, 2021.

[34] Nicolli Rios, Rodrigo Oliveira Spínola, Manoel Mendonça, and Carolyn Seaman. The most common causes and effects of technical debt: first results from a global family of industrial surveys. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2018.

[35] Vallary Singh, Will Snipes, and Nicholas A Kraft. A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. In *2014 Sixth International Workshop on Managing Technical Debt*, pages 27–30. IEEE, 2014.

[36] Mikel Villamañe, Begoña Ferrero, Ainhoa Álvarez, Mikel Larrañaga, Ana Arruarte, and Jon Ander Elorriaga. Dealing with common problems in engineering degrees' final year projects. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE, 2014.

[37] Marion Wiese, Paula Rachow, Matthias Riebisch, and Julian Schwarze. Preventing technical debt with the tap framework for technical debt aware management. *Information and Software Technology*, 148:106926, 2022.

[38] Jesse Yli-Huumo, Andrey Maglyas, and Kari Smolander. How do software development teams manage technical debt?–an empirical study. *Journal of Systems and Software*, 120:195–218, 2016.