Makerere University College of Computing and Information Sciences

Department of Networks

Mugoya Dihfahsih

1 Title of the study

A Framework for identifying and evaluating process and product technical debt in early software Prototypes: A case of Early-stage Software Development teams

2 Area of Study

Technical Debt, Software Development Processes

3 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 at the software implementation [6] phase of the software development life cycle.

There are several processes [8] that software teams have to follow and software practitioners including experienced developers, university students, and young teams have to rely on information repositories such as documents, standards, and policies. Unfortunately due to costs, ever changing requirements, project time constraints [9] and other factors limit following these processes which results in costly future refactoring of such projects.

The risks in the students or amateur software practitioners' projects [3] is due to lack of enough research conducted in evaluating amateur practitioners' processes which lead to project failure. The existing studies have proposed models [4] to manage the technical debt in projects but this needs to be dealt with at early stages of project prototypes especially for student 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.

4 Preliminary Literature Review

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 a project by these young teams. 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.

5 Statement of the Problem

Most studies about technical debt are focused on minimising the costs incurred in refactoring industrial projects [11] and there is little literature or research about evaluating technical debt in university projects always developed by junior or amateur practitioners. These practitioners write partial project documents and at times none, untested code methods and carryout anti-patterns such as using wrong architecture which makes it hard to change in future leading to technical debt. There are many innovative projects [3] that students or junior practitioners work on but due to lack of a framework to identify the gaps in their techniques, most of these projects are costly to refactor when adding features to them and they end up being trashed by the project investors. There is a need for a framework that can be used identify these gaps in processes and products in project prototypes 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 resources and costs incurred on paying technical debt in future

6 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.

6.1 Specific objectives

- This study will evaluate policies, processes or activities that have been foregone or missing in the projects by the novice, junior or student practitioners in university projects to reduce project failure.
- 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.

7 Hypotheses of this study

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: To what extent have the young teams foregone policies, processes or technological activities in software projects?
- RQ2: How will awareness about technical debt by young teams minimise project failure as early as possible?
- RQ3: What is the relationship between the techniques young teams use on software projects and occurrence of technical debt?

8 Theoretical 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.

9 The Research Design

To answer the research questions RQ1, RQ2 and RQ3, a qualitative strategy 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. The results of this data analysis will enable the formulation of the framework to reduce the occurrence of technical debt.

9.1 The scope of the study

References

[1] 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.

[2] 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.

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

[4] 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.

[5] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1992.

[6] Carlos Fernández-Sánchez, Juan Garbajosa, Carlos Vidal, and Agustín Yagüe. An analysis of techniques and methods for technical debt management: a reflection from the architecture perspective. In *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*, pages 22–28. IEEE, 2015.

[7] 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.

[8] Watts S Humphrey. Characterizing the software process: a maturity framework. *IEEE software*, 5(2):73–79, 1988.

[9] Rupinder Kaur and Jyotsna Sengupta. Software process models and analysis on failure of software development projects. *arXiv preprint arXiv:1306.1068*, 2013.

[10] 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.

[11] Edith Tom, Aybüke Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, 2013.

Changelog @changelog · 17h

“Just keep coding we can always fix it later”

Show this thread