Building an approach for identifying and Mapping Developer Technical gaps/debt in early-Prototypical Architectures

MUGOYA DIHFAHSIH

May 2022

1 Background of the study

Technical Debt (Technical gap) is a metaphor introduced by Cunningham [1] 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. In other words, it's the result of prioritizing speedy delivery over perfect code, giving priority to client values or project constraints such as inadequate budget over technical implementation and design considerations. The technical debt originates solely at software implementation [13] phase of the software development life cycle. Software companies need to adapt frameworks that can help to identify such gaps as early as initial stages of software development.

Recently, there have been many studies carried out to describe and show the existence of Technical Debt which is as a result of ignoring the basic software development principles due to inadequate funds[9], demanding market needs thus lack of time[10], inadvertence as a human factor[11] such as choosing an optimal technologies like libraries or frameworks that become obsolete with time[12]. Most of these studies address the technical debt after the software has been implemented rather than finding a framework or an approach that minimises the occurrence of technical debt at early stages of development

1.1 Technical Debt in project documentation

There are a lot of systems that have project documents that have technical gaps which make the maintenance of the projects costly and time consuming. This is known as Documentation Technical Debt[14] which is due to concerns of non-existent, inadequate or incomplete software projects' documentation. Technical debt in documents comes from developers simply moving too fast and taking shortcuts and perfectionism that leads to deviation from project specification.

1.2 Early Prototypical Designs

Early prototype is a model version of a software product,it is used as an early and less expensive sample of a product that allows to test its features, gather feedback and identify defects[15]. It is aimed at validating customer demands, testing of technical feasibility, meeting customer standards and shower casing to investors in case of project funding. Early prototyping is always ideal for small business start ups due to the mentioned benefits but it comes with weakness that would lead into technical debt. The dangers of prototyping[16] include; hidden assumptions which might surface too late, obtaining feedback in the context of use is prohibitively expensive and rarely done and sometimes the start-up cost of building the development team, focused on making prototype is high.

1.2.1 Minimum Viable Product

The use of early prototypes in software development has led companies especially startups to evolve into using methodologies such as Minimum Viable Product(MVPs)[17] aimed at enabling them gain customer's interests in their products with less effort and expense on the product whose success on market is uncertain. Such methodologies are good because they help the developers to write code according to what the customers want and then drop the previous prototype that didn't work depending on the customer feedback. The fundamental deficiency MVPs is that developers update code depending on the feedback given to them by customers and don't consider how these changes will affect the entire software hence leading into a technical debt as little or no documentation is done about the new changes[3] to the software.

1.3 Objective of this study

There is need of a proper approach that has to be followed by developers if they are to use the early prototypical methodologies to develop systems that will not lead to technological death in the near future due to immaturity in their coding practices.

The objective of this study is to create an approach that identifies gaps in methods to artifacts like architectures, use cases,database designs, user interface designs and implementation of technology stacks to guide software development teams in early prototypical architectural systems to minimise occurrence of technical debt which is always costly

2 Statement of the problem

2.1 Part A The Ideal

Software development teams always prefer when they deliver their projects in the least time possible and using relatively easy technologies.
The teams will desire to implement new features to the project in the future

using a well guided documentation to enable developers to quickly analyze the code and refactor it if necessary

2.2 Part B The Reality

According to Edith et al [5], companies have several reasons for projects to end up in Technical Debt [4]Pragmatism and prioritization – when you have to make sub-optimal decisions because of the situation on the market.The software software development decisions that result in prioritizing speed [10] or release over the well-designed code, the company prioritizing customer needs over the quality of code and software documentation which in the long run will cost the company a lot of resources to refactor such software to suit deemed changes in it.

In this case, the implementation of critical functions can be considered more important than overall quality. ~~Ignorance –~~ the inability of the developers to develop high quality applications Certain attitudes of developers can also cause Technical Debt. [6] Inexperienced and unsupervised developers are most likely to cause the technical debt in during software developer because they always choose technologies without considering their future consequences. Most times programmers and management are hesitant to improve code, for fear of introducing new problems. Technical Debt caused by such an attitude without any external necessity like market needs or customer demands is reckless and deliberate debt[7].

2.3 Consequences of the technical gap

Technical debt can reduce the team's agility, produce poor quality code, strain the testing team and eventually reduce the company's overall productivity.[2][3] A lot of time and resources are spent on the rework of these gaps that were left in the code as well as in the documentation. [5]

3 The significance of this study

This study aims at building an approach that will enable software teams to identify the technical gaps in the projects at prototyping stage and also map the debts in the existing project documentations to facilitate proper project refactoring thus supporting business mitigation of future technical debt in projects

The study will enable project leaders have a good approach for identifying and raising technical debt by making such challenges visible to enable a startup or a firm's software development team to get these changes prioritised in the development stage.

4 References

[1] Cunningham, W., 1992. The wycash portfolio management system. SIGPLAN OOPS Mess. 4 (2), 29–30

[2] Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C., 2016b. Managing technical debt in software engineering (dagstuhl seminar 16162). Dagstuhl Rep. 6 (4), 110–138.

[3] Martini, A., Bosch, J., Chaudron, M., 2015. Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study. Inf. Softw. Technol. 67, 237–253.

[4] FOWLER, M., Technical debt quadrant. www.martinfowler.com/bliki/TechnicalDebtQuadrant.html

[5] TOM, E., AURUM, A. and VIDGEN, R., 2013. An exploration of technical debt. Journal of Systems and Software, 86(6), pp. 1498-1516.

[6] ELM, J., 2009. Design Debt Economics: A Vocabulary for Describing the Causes, Costs and Cures for Software Maintainability Problems. IBM,

[7] Y. Guo and C. Seaman (2011), A portfolio approach to technical debt management, MTechnical Debt 11: Proc. of the 2nd Workshop on Managing Technical Debt.

[8] P. Kruchten, R.L. Nord and I. Ozkaya, I. (2012). Technical Debt: From Metaphor to Theory and Practice. IEEE Software, Published by the IEEE Computer Society.

[9] KTATA O., LEVESQUE G., 2010. Designing and implementing a measurement program for scrum teams: what do agile developers really need and want? 3rd C* Conference on Computer Science and Software Engineering 2010, C3S2E '10, Montreal, QC, Canada, May 19, 2010–May 20, 2010. Association for Computing Machinery, 101–107, *A5

[10] YLI-HUUMO, J., MAGLYAS, A. and SMOLANDER, K., 2014. The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company. 8892, pp. 93-107

[11] GOLDEN J.M., Transformation patterns for curing the human causes of technical debt, Cutter IT Journal 23 (10) (2010) 30–35. [12] Architectural design decisions that incur technical debt — An industrial case study Mohamed Soliman , Paris Avgeriou , Yikun Li

[13] Carlos Fernández-Sánchez Juan Garbajosa Carlos Vidal Agustín Yagüe An Analysis of Techniques and Methods for Technical Debt Management: a Reflection from the Architecture Perspective

[14] Mendes, Leonardo et al. "Documentation Technical Debt: A Qualitative Study in a Software Development Organization." Proceedings of the XXXIII Brazilian Symposium on Software Engineering (2019): n. pag.

[15] Mahil Carr Department of Information Systems City University of Hong Kong 83 Tat Chee Avenue Hong Kong, Dr. June Verner College of Information and Technology Drexel University 4131 Chestnut St Philadelphia PA 19104 "Prototyping and Software Development Approaches"

[16] "Prototyping Considered Dangerous" Michael E. Atwood, Bart Burns, Andreas Girgensohn, Alison Lee, Thea Turner,and Beatrix ZimmermannNYNEX

Science and Technology500 Westchester AvenueWhite Plains, NY 10604 USAatwood, bart, andreasg, alee, thea, bz@nynexst.com

[17] MVP Explained: A Systematic Mapping Study on theDefinitions of Minimal Viable ProductValentina Lenarduzzi, Davide TaibiFaculty of Computer ScienceFree University of Bolzano/Bozen39100 Bolzano/Bozen - Italy