SRAVANI BEJAWADA besr17@student.bth.se 970525-9563

Impact of code reviews on Software Code Quality

Summary paper [1]:

The thought of technical debt pulls in noteworthy consideration, particularly with regards to reconciling architecture and agile improvement. In any case, most work on technical debt is still casual and on the off chance that it gives a formalization it is regularly specially appointed. This paper gives a detailed information about a formal approach to technical debt decision Making. Utilizing this formalization, we demonstrate that ideal choice making isn't viably calculable in genuine circumstances what's more, give a few very much characterized approximations that permit to deal with the issue all things considered in handy circumstances. Joining these approximations in a solitary technique leads to a light-weight approach that can be successfully connected in software programming improvement, including agile methodologies.[1]

This paper intends to contribute:

- A detailed formalization to empower the evaluation of technical debt.
- A formal and exact reason for basic leadership on technical debt, in light of our technical debt formalization.
- We demonstrate that an exact basic leadership approach fundamentally prompts down to earth issues. We address this by giving precise, all around characterized approximations.
- We determine a for all intents and purposes appropriate basic leadership method in view of our formalization, which consolidates characterized approximations and can be utilized by and by in software improvement.

Summary paper [2]:

In this paper, we concentrate on structure debt spoken to by code smells. We think about three smells that we distinguish in open source frameworks of various domains. Our main aim is to give an exhortation on which structure debt must be paid first, as per the three scents we have broken down. Also, we talk about if the discovery of these smells could be custom-made to the application area of a framework[2]

This paper mainly aims to answer these questions:

- What is the effect of expelling a specific sort of the smell on various programming quality metrics? Furthermore, thus which smells are more basic from the structure debt purpose of view, and which ought to be evacuated first? As such, is there a smell which speaks to a marker of structure obligation more than different scents?
- Is it conceivable that a smell recognized in a framework could not be viewed as a smell in an arrangement of another domain? Are there a few sorts of conditions of the smell obligations to the domain?

In this paper mainly focus on these three code smells: God Class, Data Class and Duplicate Code, for each code smell relevant metrics are selected after refactoring.

Summary paper [3]:

To minimize the problems of the software quality the quality of the product is measured by using square metrics methods. Different methods are applied for improving the quality of the quality of the software like refactoring, requirement measurement plan and quality modelling. During refactoring the software quality attributes like Correctness, Completeness, Consistency and Non-Ambiguity are determined. Without change in the behavior how refactoring helps to improve the quality of the software is shown in this paper in detail**[3]**.

This paper mainly focus on how the requirements are plays a key in the improvement of the quality. The factors which effecting the inferior quality are explained in detail the factors like Logical design errors, Coding errors, Documentation error etc.,

Summary paper [4]:

The main aim is to investigate database pattern quality, related qualities and their associations with other software artifacts. In this paper they present an inventory of 13 database pattern smells and evoke engineers' point of view through an overview. We separate inserted Sql explanations and distinguish database composition smells by utilizing the DbDeo tool which we created. We break down 2925 generation quality frameworks (357 mechanical and 2568 all around built open-source ventures) and exactly think about quality attributes of their database diagrams. Altogether, we examine 629 million lines of code containing in excess of 393 thousand sql proclamations. We find that the record misuse smell happens generally as often as possible in database code, that the utilization of an ORM system doesn't invulnerable the application from database smells, and that some database smells. The main goal is to understand the code quality, so the research questions are formulated to show in a better way. Data base smells are categorized to understand easily like schema smells, query smells and data smells.[4]

Summary paper [5]:

In programming maintenance, the capability of additional exertion required in future as though paying enthusiasm for the bringing about debt. The vulnerability of intrigue installment further confounds the issue of what debt ought to be brought about or reimbursed and when. To encourage software directors to settle on educated choices, a portfolio approach is proposed in this paper. The methodology uses the portfolio the board hypothesis in the backspace to decide the ideal gathering of specialized debt things that ought to be caused or on the other handheld. We expect this methodology could give another point of view for the specialized debt the board.[5]

Code quality:

Paper [1]:as the length of the code is quite large the change in the functionality and the quality attributes like performance and security which shows change in behavior.

Paper [2]: new usefulness and additionally quality alterations like execution or on the other hand security upgrades can be development steps. In any case, adjustments that don't prompt

(at any rate conceivably) remotely detectable conduct change, for example, a viability enhancement are not viewed as advancement steps.

Paper [3]: explains how refactoring can be done without change in behavior and the how it improves the code quality which in turn increases the understandability of the code and the maintenance of the software would be increased.

Paper [4]: Contemplating database code quality can enable us to comprehend the utilization attributes of database code. The examination offers commitments to both research and practice. For analysts, it gives a technique to research code nature of implanted SQL explanations by mining repositories.

Paper [5]: low quality codes might be composed accidentally by a developer because of an absence of experience. Be that as it may have specialized obligation can likewise be utilized deliberately. By postponing certain upkeep assignments or doing them rapidly and less precisely, programming supervisors can exchange off programming quality with efficiency, which may have extra advantages, for example, that picked up from before time to showcase than their competitors.

Main findings:

Paper [1]: primary concentration with the methodology we give is to obviously recognize the approximations we make over an exact formalization. We expect that further particular methodologies will be inferred later on a similar formal premise utilizing unique or sorts of approximations. A further advantage of our careful formalization is that it obviously distinguishes imitations and presumptions of our methodology.

Paper [2]: the main findings are the detection of the code smells the detected code smells are god class data class and the duplicate code smells. The god class and the data class are detected by using iplasma. and the duplicate code smell by google codepro analytics.

Paper [3]: In this investigation, scientists concentrated on the "Faulty definition of requirements" issues, where the recognizable proof of software requirements is the fundamental and critical perspective to decide the quality and the collapse of a product.

Paper [4]: examine on the mining database smells underway quality frameworks including both the machine tool as the open-source programming. We dissect SQL proclamations to quantify construction nature of social databases with an emphasis on execution what's more, viability quality properties. We investigate event examples of database construction scents and make sense of the level of co-event among blueprint smells. We likewise think about the factors that influence the thickness of database smells.

paper [5]: specialized debt is a potential misfortune to programming ventures. In this manner overseeing specialized obligation centers around diminishing the negative effect of the obligation. Notwithstanding, as made reference to before, specialized obligation likewise has its advantage side, which enables it to be utilized as a technique. Besides, specialized debt isn't risky as long as the advantage exceeds its expense. This view has motivated us to take a gander at the bringing about of specialized obligation as a venture.

The similar findings from all the papers is that how code smells are affecting the quality attributes. How metrics are applied for improving the quality for the code smells.

Learnings:

Paper [1]:

This helped me to infer a way to deal with specialized obligation basic leadership that is adequately light-weight to incorporate it even into dexterous improvement approaches with insignificant overhead. Obviously, the particular approximations we talked about are just a few models of conceivable approximations.

Paper [2]:

The complexity of the code smells is learned The Extract Method refactoring procedure negatively affects framework dependency. We have as it was utilized this procedure because the greater part of the copy code has been recognized in similar classes or in classes that as of now had a sort of dependency.

Paper [3]:

Learned how to calculate the requirement quality for the quality attributes. The difference between the software quality attributes and the use cases scenarios before refactoring and after refactoring the code is clearly explained.

Paper [4]:

Application type (Desktop, Mobile, or Web) has no critical effect on the database smell thickness, utilization of an ORM structure doesn't keep away from the database blueprint scents, and the smell clone table in mechanical ventures and smell esteems in trait definition in open-source ventures display the most astounding co-event with other database smells.

Paper [5]:

The center segment of the proposed methodology is a "technical debt list." The rundown contains technical debt "things", every one of which speaks to a deficient assignment that may cause issues in the future. Models of technical debt things incorporate source code that should be altered to adjust to the compositional structure (counting refactoring), test cases that should be worked out, documentation that should be refreshed, and so on. Everything incorporates the area of the debt, the time at which it is recognized, the dependable individual, the motivation behind why it is viewed as, technical debt a gauge of the foremost, assessments of the normal intrigue sum (EIA) and intrigue standard deviation (ISD), and appraisals of the connections of this thing with other technical debt things. As with budgetary debt, the main alludes to the exertion required to finish the undertaking.

REFERENCES:

[1] K. Schmid, “A Formal Approach to Technical Debt Decision Making,” in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, New York, NY, USA, 2013, pp. 153–162.

[2] F. A. Fontana, V. Ferme, and S. Spinelli, “Investigating the Impact of Code Smells Debt on Quality Code Evaluation,” in *Proceedings of the Third International Workshop on Managing Technical Debt*, Piscataway, NJ, USA, 2012, pp. 15–22.

[3] “(PDF) Software quality measurement and improvement using refactoring and square metric methods,” *ResearchGate*. [Online]. Available:

https://www.researchgate.net/publication/280818173_Software_quality_measurement_and_improvement_using_refactoring_and_square_metric_methods. [Accessed: 25-Nov-2018].
[4] T. Sharma, M. Fragkoulis, S. Rizou, M. Bruntink, and D. Spinellis, “Smelly Relations: Measuring and Understanding Database Schema Quality,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, New York, NY, USA, 2018, pp. 55–64.
[5] Y. Guo and C. Seaman, “A Portfolio Approach to Technical Debt Management,” in *Proceedings of the 2Nd Workshop on Managing Technical Debt*, New York, NY, USA, 2011, pp. 31–34.