Relationship Of Code Smells And Commits With Technical Debt

Jaspreet Bedi, BBK DAV College For Women, Lawrence Road, Amritsar.

Abstract
Technical debt in the software development incurs due to preference of short term decisions ignoring strategic consequences. Changes in requirements of software in the fast growing and dynamic technology and business domains are obvious which may lead to increase in usage of quick and dirty approach resulting in code that is no longer clean. It is the point of introduction of debt called technical debt. The decision may be apt in situation due to absence of any alternative. If not paid for a long period of time it results in increasing technical debt. Just like financial debt interest grows with delay in payment and ultimately a situation of technical bankruptcy may arise. It is needed to study the factors contributing technical debt. The paper studies effect of code smells and commit frequency on technical debt. Correlation method of statistics is used for the purpose. The data is collected from PHP application Mockery from Github.

Index Terms—OSS, Technical debt, code smells, commit frequency.

I. INTRODUCTION

Debt is a very commonly used concept in the financial world. It is always considered with respect to some principal amount. During its payment, an interest term is also added. Likewise in technical world of software engineering this debt is considered in the software development process when some changes are required. In every organization there is need to make the changes and consequently the system should adapt to these changes. During this process knowingly or unknowingly few compromises are done in a hurry to accomplish the work within deadlines of time. The development team may not face any problem at this point of time and neither there is some wrong output but in the strategic framework it matters a lot. At extreme case when the rent/debt is not paid for a large period of time it may lead to technical bankruptcy. The team will be demotivated and the productivity will reduce thereof. At this stage it will not be feasible to continue further in the project.

Martin Fowler[2009] posted famous TD quadrant concept in his blog. He starts with the question whether messy code or bad system design is TD or not. Four types of approaches for implementing code are described. He considers debt as prudent and reckless. The prudent debt to reach a release may not be worth paying down if the interest payments are sufficiently small whereas a sloppy and low quality code is a reckless debt, which results in crippling interest payments or a long period of paying down the principal.

The future costs attributable to known structural flaws in production code that need to be fixed. It includes both principle and interest. Principal is the cost of remediating must-fix problems in production code. It is calculated as product of number of hours required to remediate must-fix problems and fully burdened hourly cost of those involved in designing, implementing, and testing these fixes. Interest means the continuing costs that can result from the excessive effort to modify unnecessarily complex code, greater resource usage by inefficient code etc. Technical Debt is said to be induced by certain factors and there are many types of TD suggested in literature.

1) Reasons and causes for TD

Researchers have mentioned many reasons for TD. Dr. Dan Rawsthorne [39] in his blog says “Technical Debt is everything that makes your code hard to work with. It is an invisible killer of software which must be aggressively managed.” He pointed out that lack of test, bad design, lack of documentation and poor

readability are reasons of arising technical debt. Stephanie W. in his blog [40] categorized the causes of technical debt as intentional and unintentional. In the intentional causes he includes time constraints, Source code complexity, Business decisions which lack technical knowledge while in the unintentional causes he included lack of coding standards and guides, junior coders with little experience and poor skills and lack of planning for future developments. (Tushar sharma,2018) claimed that Technical debt may have one or more causes, such as time pressures, Overly complex technical design, Poor alignment to standards, Lack of skill, Suboptimal code, Delayed refactoring and Insufficient testing. General agreement on the types is not there in the TD metaphor as there are many definitions of the types of technical debt that exists. (Hampus Nilsson,2013).

2) Types of TD

Literature includes many types of TD from perspectives of software development and business etc. and in order to deal with the issues related to technical debt there is need to classify technical debt. Popular types include Deliberate tech debt(in the cases in which the quick way is the right way but at times the developer team knowingly does something the wrong way as quick delivery of product is must.), Accidental/outdated design tech debt(when it is needed to balance future-proofing designs with simplicity and quick delivery or developer team is naive. Some name it as naive tech debt or outdated design tech debt.) and Bit rot tech debt (when during passage of time many incremental changes are incorporated in the software system by using copy-paste and cargo-cult programming only without fully understanding the original design.) (Steve McConnel2007) in his blog categorized technical debt into unintentional debt, which is foolish to incur, and intentional debt, which might be incurred for reasons such as time to market, preservation of startup capital and delaying development expense.

Open source software (OSS) has given a new direction to technical debt. In the version control systems on daily basis many times new and new features are added and commits are done. It becomes clear that the probability of technical debt grows manifolds. In such systems, it is quite evident that an increasing number of commits may lead to more TD. This is also evident that the code smells are a major reason to introduce TD. This paper will be investigating the relation of two factors viz. commits and code smells with technical debt. The paper is organized into different sections. The first section introduces the topic. Section II is the review of related literature. It includes the major milestones in the theoretical framework of technical debt. The section III describes the variables considered in study. Section IV includes the research questions. Section V deals with tools and software used for the study. Section VI includes the method of investigation in detail. It includes the work, statistical tools and the graphical trends etc. Section VII includes the threats to validity and conclusions are drawn in section VIII. The sources consulted for the study are listed in the last section that is bibliography.

II. LITERATURE REVIEW

1992 may be considered as the birth year of the metaphor TD when Ward Cunningham in an experience report coined it first. “Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite[Cunningham,1993]. He included the possible loss that may occur later on as “… The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object oriented or otherwise.” There was a large gap in the further significant research in the area till [Zeller et.al 2005] pointed out the role of specific day in a week for TD. Study claimed that programming on Friday is more likely to generate faults than on any other day.

Some engineers and software developers consider TD as a short cut for output and they use a dirty approach. [Cunningham, 2009] never intended for technical debt to be used as an excuse to write poor code. Despite various definitions of technical debt, the blogging community has continued to preserve Cunningham’s original representation of technical debt as a trade-off between quality, time and cost. Other bloggers include quick and dirty approaches [Steve McConnell, 2007] and design and quality flaws Marinescu, R.[2012] as technical debt. Martin Fowler [2009] introduced very popular TD quadrant where

he mentioned four types of TD. Fowler divided basic two types of TD viz., intentional and unintentional debt into reckless and prudent debt. There were some papers on TD management thereafter. Brown, Nanette, et al[2010] and Nitin Taksande [2011] submitted thesis on the empirical study of TD. He emphasized on significance of TD from strategic viewpoint. Rothman, Seaman and Guo [2011] categorized types of technical debt as testing debt, Defect debt, Documentation debt and Design debt.

J. Eyolfson et al. [2011] analyzed relation of time of Day and Developer Experience and Committing Bugs. Rahman and Devanbu [2011] studied the impact of ownership and experience of the developers on the quality of code. They also conceptualized two distinct types of experience that can affect the quality of a developer's work viz., specialized experience and general experience. F. Rahman and P. Devanbu [2011] also conducted study on Ownership, experience and defects at a fine-grained level. P. Runeson, M. Host, A. Rainer and B. Regnell [2012] consolidated the historical milestones of TD in a very significant systematic literature review. Tom et al. [2013] conducted a systematic literature review to establish a theoretical framework of technical debt. The authors identified two elements of TD viz., code decay and design debt and the boundaries of TD. [Dan O'Neill, 2013] in his article described three groups of conditions which when they occur can result in the accumulation of technical debt. Management debt triggers include tight and highly prioritised completion schedules, squeezed budgets and poor communication between management and engineering.

Li, Zengyang et al [2015] conducted mapping study on technical debt and its management and came out with a detailed classification of TD. Ten types were outlined viz., Requirements TD, Architectural TD, Design TD, Code TD, Test TD, Build TD, Documentation TD, Infrastructure TD, Versioning TD, Defect TD. Eight TDM activities were also outlined viz, TD prevention, identification, measurement, representation/documentation, prioritization, monitoring, and repayment O'Neill D [2013].

Alves et al. [2015] investigated the influence of developers on the introduction of code smells in 5 open source software systems. Li, Zengyang [2015] have classified developers in different groups based on two characteristics, namely developer participation (calculated as the time interval between his first and last commit) and developer authorship (representing the amount of modified files and lines of code). The authors investigated how those two characteristics are related to the insertion and/or removal of five types of code smells as dead (unused) code, large classes, long methods, long parameter list (of methods) and unhandled exceptions. Results suggested that groups with fewer participation in code development tended to have a greater engagement in the introduction and removal of code smells. Authors suggested that groups with higher participation level code more responsibly during maintenance whereas the other groups tend to focus on error correction actions.

Everton da S. Maldonado [2016] in his thesis included one new type of TD of his times viz., Self-Admitted Technical Debt. Per Classon [2016] submitted thesis on Managing Technical Debt in Django Web Applications. He emphasized that appropriate strategies are necessary to support decisions about when and to what extend a TD item should be paid off. Alves et al. [2015] studied the strategies followed by different researches and found six strategies viz., Cost-Benefit Analysis, Portfolio Approach, Options Investment Analytic Hierarchy Process (AHP), calculation of TD Principle and Marking of Dependencies and Code Issues. Alves, Nicolli SR et al [2016] conducted very systematic mapping study of the period.

Theodoros Amanatidis et al [2017] considered software quality from the perspective of TD. Tufano et al, [2017] analyzed developer-related factors on 5 open source Java projects that could influence the likelihood of a commit to induce a fix. They found evidence that clean commits have higher coherence than fix-inducing commits. Commits with changes that are focused on a specific topic or subsystem are considered more coherent than those with more scattered changes. Furthermore their results suggested that developers with higher experience perform more fix-inducing commits that developers with lower experience. Studies over the years have proposed different approaches to measure technical debt, which has been found to impact (internal) quality. Zhixiong [2017] and thesis of Sultan Wehaibi [2017] were worth mentioning. Mrwan BenIdris [2018] pointed that the total number of selected empirical studies has nearly doubled from 2014 to 2016. It is clear from state of art that research occurred on the theoretical framework of technical debt. There is a void if quantification of technical debt is seen in research.

III. VARIABLES USED

A. *Commit frequency*

Commit frequency means number of commits.

B. *Code Smells*

Code smell, also known as a bad smell, in computer programming code, refers to any symptom in the source code of a program that possibly indicates a deeper problem.

IV. RESEARCH QUESTIONS

RQ#1: Does TD correlate with commits?

The goal is to provide an initial investigation to determine if some correlation exists between two parameters viz., commits and td.

RQ#2: How do code smells affect TD?

The goal is to determine if some correlation exists between two parameters viz., code smells and td.
Hypothesis can be easily framed as
H_0: Code smells and commits increase technical debt.

V. TOOLS AND APPLICATION USED

Following tools were used in performing the analysis of current paper:

A. *Sonarqube*

SonarQube is a web-based open source platform which is used for measuring and analyzing the source code quality. SonarQube is maintained by SonarSource. The tool is written in java. It can analyze and manage code of twenty plus programming languages like c++, PL/SQL, Cobol etc. More than 50 plugins are available to extend the functionality of SonarQube. SonarQube receives files as an input and analyzes them and calculates a set of metrics. It then stores them in a database and shows them on a dashboard.

B. *MS-Excel*

MS EXCEL 2010 was used to enter and analyse the results of study. There is a wide variety of statistical and mathematical functions of Excel which were required for the study. Graph drawing feature was also very helpful in showing the relation of the variables. Regression function of excel was used to calculate value of r^2 that is regression coefficient.

C. *Vetkra Mockery*

It is a one of the PHP applications in the top twenty list of github. Mockery provides the ability to easily generate mocks for golang interfaces. It removes the boilerplate coding required to use mocks. It is considered case for study while the unit of consideration is each revised project that is available as new version.

D. *Origin 6.0*

Origin 6.0 is a graphic drawing tool. Origin is a proprietary computer program used for data analysis. Origin Lab Corporation.[35]

VI. RESULTS AND DISCUSSION

The data of 231 versions of projects of Vektra Mockery PHP (application considered for study) was collected from github. For this all the projects were downloaded on machine. Sonarqube was used to find technical debt for all projects. The outputs from sonar dashboard were entered in Excel worksheet. TD calculated was normalized using mathematical functions of excel as the data was in different units of time that is days, hours and minutes. All 38 contributors of application were considered. The process of analysis is a three step process and is depicted in figure 1.

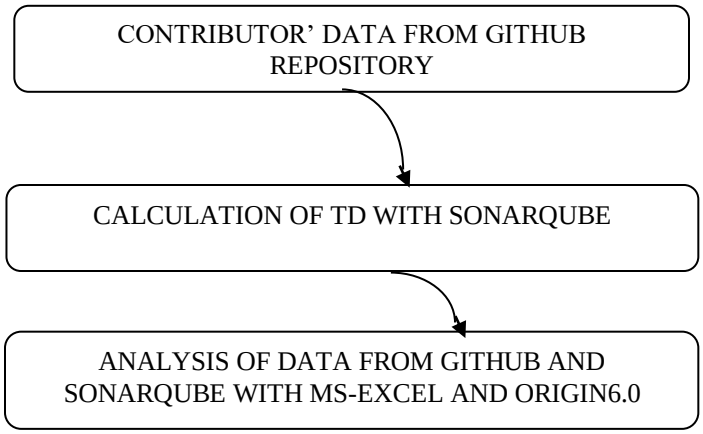


Figure 2: Process of analysis

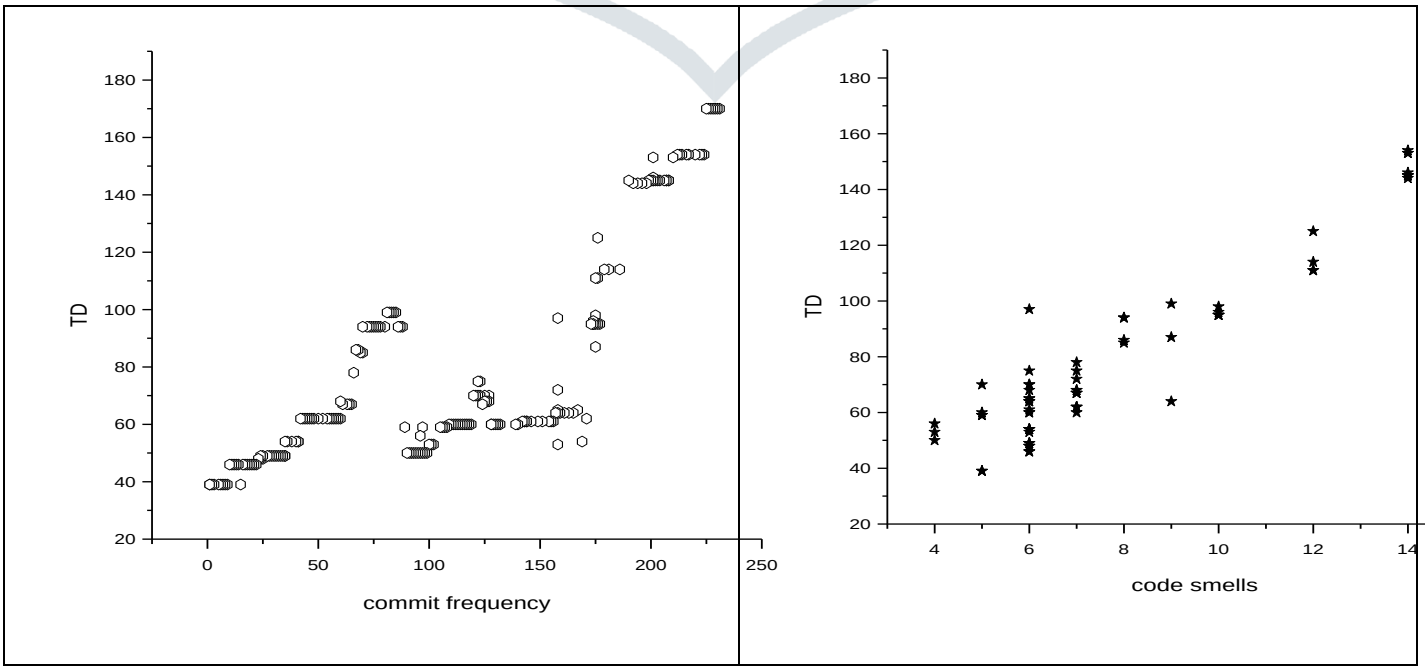
In order to find answer research questions, correlation of the commits and code smells with TD separately is considered. It comes out as approx .78 with commits which is more than .5. it indicates that a positive correlation exists between commits and TD which is statistically significant. While a correlation of the code smells and TD comes out as approx .97 which is very near to 1.

Figure 3: correlation matrix of commits and code smells with TD

	commits	*TD*
commits	1	
TD	0.775834	1

	code smells	*TD*
code smells	1	
TD	0.965048	1

Figure 4: Graphs showing increasing trend of TD in hours w.r.t. commit frequency and code smells resp.



The pictorial layouts in figure 4 shows the relation of number of commits and that of TD and relation of number of code smells and that of TD in two parts. The graphs show an overall increasing trend of TD with increase in code smells. At certain intervals it is constants like 6,7,9 etc. but it cannot be considered so significant as it is so only at a very few points. Similarly an overall increasing trend in TD due to code smells is also clear. At certain intervals it is constants like 10-50, 50-70,110-130. It is however decreasing at certain points like 95-100 but it cannot be considered so significant as it is so only at a very few points. It is obvious that null hypothesis is proved.

VII. THREATS TO VALIDITY

[R. Marinescu, 2012] points that in every experimental study, there are threats to validity. There are threats to external validity that are related with generalization of results. In order to generalize results, all releases or commits of MOCKERY are used in the study. Then also it cannot be ensured that the findings are valid for all other domains, applications or all open source systems. Many case studies are needed for establishing whether the aforementioned outcomes concerning technical debt are applicable in all applications or domains.

Threats to internal validity may also exist. They are confounding factors that may have influenced the results of this study. This threat is mainly related to the assumption that all cases of version control systems have same behavior in terms of users and technical debt inducing factors. To limit this threat, random manual inspections of projects or commits was performed. To a large extent, the results of these inspections support the assumption, but in order to draw any solid conclusions on the actual causes of the changes between releases, a detailed study is needed with the many more such projects.

VIII. CONCLUSION

There are a number of factors affecting technical debt. This is an important area of research to unveil these factors. It can be well concluded that the commits and code smells effect TD. It is statistically proved that both these factors effect TD significantly. As code smells and commit frequncy grow, there is corresponding increasing change in TD. Further research is needed to unveil more contributing factors towards technical debt. The code smells when kept under control can help in maintaining TD within limits. Commit frequency will increase in a project if there is need to make changes in the form of additions and deletions. It is can be very vividly stated that commits are bound to increase in the competitive world of business and technology b fact but the finding of the paper indicates that on committing the check should always be there on the code smells. If they do exist beyond certain limit the solutions dealing with code smells like refactoring etc. should be used.

IX. BIBLIOGRAPHY

1. W. Cunningham. The WyCash Portfolio Management System. http: //c2.com/doc/oopsla92.html.
2. Cunningham, Ward. "The WyCash portfolio management system." ACM SIGPLAN OOPS Messenger 4.2 (1993): 29-30.
3. J. Sliwerski, T. Zimmermann, and A. Zeller, “Don’t Program on Fridays! How to Locate Fix-Inducing Changes,” in Proceedings of the 7th Workshop on Software Reengineering, Bad Honnef, Germany, 2005.
4. Steeve McConnell. Technical Debt. http://blogs.construx.com/blogs/ stevemcc /archive/2007/11/01/technical-debt-2.aspx.
5. Cunnigham, W., Ward Cunningham's Debt Metaphor Isn't a Metaphor, in Powers of Two, R. Mayers, Editor. 2009
6. Atwood, J., Coding Horror: Paying Down Your Technical Debt, in programming and human factors. 2009.
7. Robert C. Martin. A Mess is not a Technical Debt. http://blog.objectmentor.com/articles/2009/09/22/ a-mess-is-not-a-technical-debt
8. Fowler, M. Technical debt quadrant, 2009; http:// martinfowler.com/bliki/TechnicalDebtQuadrant.html.

9. "Managing technical debt in software-reliant systems." Proceedings of the FSE/SDP workshop on Future of software engineering research. ACM, 2010.
10. Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. Managing technical debt in software-reliant systems. FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research Pages 47-52, 2010.
11. J. Eyolfson, L. Tan, and P. Lam, "Do Time of Day and Developer Experience Affect Commit Bugginess?," in Proceedings of the 8th Working Conference on Mining Software Repositories, New York, NY, USA, 2011, pp. 153–162.
12. Nitin Taksande, Thesis: Empirical study on technical debt as viewed by software practitioners [2011]
13. Strutz, N., Technical Debt - it's still a bad thing, right?, in The Dopefly Tech Blog. 2011.
14. L. Tan, and P. Lam, "Do Time of Day and Developer Experience Affect Commit Bugginess?," in Proceedings of the 8th Working Conference on Mining Software Repositories, New York, NY, USA, 2011, pp. 153–162.
15. F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," In Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu, USA, 2011, p. 491.
16. P. Runeson, M. Host, A. Rainer, and B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, 1st ed. Wiley Publishing, 2012.
17. Thesis : Markus Lindgren ,Bridging the software quality gap [Oct. 2012]
18. Tom, Edith, Aybüke Aurum, and Richard T. Vidgen. "A Consolidated Understanding of Technical debt." ECIS. 2012.
19. P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," IEEE Softw., vol. 29, no. 6, pp. 18–21, Nov. 2012.
20. Curtis, B.; Sappidi, J.; Szynkarski, A."Estimating the Principal of an Application's Technical Debt" Nov.-Dec. 2012 v.29 p.34-42, ISSN 0740-7459
21. Marinescu, R. (2012). Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, *56*(5), 9-1.
22. **Marinescu, Radu. (2012). Assessing technical debt by identifying design flaws in software systems. Ibm Journal of Research and Development. 56. 9:1-9:13. 10.1147/JRD.2012.2204512.**
23. Tom et.al[2013] Edith, AybüKe Aurum, and Richard Vidgen. "An exploration of technical debt." Journal of Systems and Software 86.6 (2013): 1498-1516.
24. O'Neill, D. (2013). Technical Debt In the Code. Defense Aquisition, Technology and Logistics, XLII No.2,, 35 - 38. SonarQube™. (2013). Retrieved 20/07/2013, from http://www.sonarqube.org/
25. Li, Zengyang, Paris Avgeriou, and Peng Liang. "A systematic mapping study on technical debt and its management." Journal of Systems and Software 101 (2015): 193-220.
26. L. Alves, R. Choren, and E. Alves, "An Exploratory Study on the Influence of Developers in Code Smell Introduction," in Proceedings of the 10th International Conference on Software Engineering Advances (ICSEA 2015), Barcelona, Spain, 2015.
27. Thesis : Per Classon, Managing Technical Debt in Django Web Applications [2016]
28. N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," Inf. Softw. Technol., vol. 70, pp. 100–121, Feb. 2016.
29. Theodoros Amanatidis, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Ioannis Stamelos, Who is Producing More Technical Debt? A Personalized Assessment of TD Principal Conference Paper · May 2017
30. M. Tufano, G. Bavota, D. Poshyvanyk, M. Di Penta, R. Oliveto, and A. De Lucia, "An empirical study on developer-related factors characterizing fix- inducing commits," J. Softw. Evol. Process, vol. 29, no. 1, Jan. 2017.
31. Zhixiong Gong, Feng Lyu,Technical debt management in a largescale distributed project - An Ericsson case study[June 2017]
32. Thesis : Sultan Wehaibi,On The Relationship Between Self-admitted debt and software quality [April 2017]
33. Mrwan BenIdris, Hany Ammar, Dale Dzielski , Investigate, identify and estimate the technical debt: A Systematic mapping study, International Journal of Software Engineering & Applications (IJSEA), Vol.9, No.5, September 2018
34. Terese Besker , Antonio Martini , Jan Bosch ,Technical Debt Cripples Software Developer Productivity - A longitudinal study on developers' daily software development work[TechDebt '18, May 27–28, 2018, Gothenburg, Sweden
35. https://vizteck.com/blog/benefits-using-sonarqube/
36. https://github.com/vektra/mockery
37. https://en.wikipedia.org/wiki/Origin_(data_analysis_software)
38. https://tommcfarlin.com/code-smells/
39. https://help.github.com/en/articles/github-glossary
40. https://3back.com/scrum-industry-terms/the-4-types-of-technical-debt/
41. https://www.castsoftware.com/blog/the-causes-of-technical-debt-do-not-exist-in-a-vacuum