Quest Journals
Journal of Software Engineering and Simulation
Volume 6 ~ Issue 5 (June 2020) pp: 23-27
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Technical Debt and Commits

Jaspreet Bedi
BBK DAV College for Women, Lawrence Road, Amritsar

Abstract—. *Technical debt is the debt in the software development incurred due to preference of short term deadlines over the strategic ones. The paper studies the relation of commits and the technical debt.*

Index Terms—*OSS, TD.*

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[8] 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. It led to very interesting and easy to understand concept of Technical Debt Quadrant.

Curtis [2012] provides the definitions of Technical Debt, principle and interest. 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.

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.

II. Organization Of Paper

This paper will be investigating the relation of commits and the technical debt. The paper is organized into four sections. The first section introduces the topic. The second section is organization of paper. The third section deals with review of related literature. It includes the major milestones in the theoretical framework of technical debt The fourth section deals with tools and software used for the study. It further includes the method of investigation in detail. The fifth section is discussion of the work and conclusions drawn. It also includes the limitations. The sources consulted for the study are listed in the last section that is bibliography.

III. Review Of Literature

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. [1] . he referred there only 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."[1] 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.

Steve McConnell[2007] 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 [4]. However, Cunningham[2009] never intended for technical debt to be used as an excuse to write poor code[5]. "…A lot of bloggers at least have explained the debt metaphor and confused it, I think, with the idea that you could write code poorly with the intention of doing a good job later and thinking that that was the primary source of debt. I'm never in favor of writing code poorly, but I am in favor of writing code to reflect your current understanding of a problem even if that understanding is partial. Despite the 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 [5] and design and quality flaws as technical debt. Some emphasized on paucity of time. In short, you may need to use suboptimal designs in the short term, because the schedule does not allow longer term designs to be used. Robert C. Martin in his blogs comes close to Wards original definition by saying "A mess is not a technical debt" [7] but he also diverges from Ward by including technical frameworks into the discussion.

Martin Fowler[2009] further divided intentional and unintentional debt into reckless and prudent debt. Fowler introduced very popular TD quadrant where he mentioned four types of TD. 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. "…The "Technical Debt" metaphor is gaining recognition in the software development community, not only to communicate the debt to the non-technical stakeholders but also to maintain the long term health of the software project…" 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 [8] 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: a fine-grained study of authorship. 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, 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 A 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 [23].

Alves et al.[2015] investigated the influence of developers on the introduction of code smells in 5 open source software systems [24]. Developers have been classified in different groups based on two characteristics, namely: a) developer participation, calculated as the time interval between his first and last commit and b) 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: 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 supported 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 [September 2016] in 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. [26] studied the strategies followed by different researches and found six most cited strategies viz., Cost-Benefit Analysis, Portfolio Approach, Options Investment Analytic Hierarchy Process (AHP), caculation 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. [Jan. 2017] analyzed developer-related factors, on 5 open source Java projects, that could influence the likelihood of a commit to induce a fix [28]. 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 ong [June 2017] and thesis of Sultan Wehaibi [April 2017] were worth mentioning. The research on TD metaphor took pace. Mrwan BenIdris [September 2018] pointed that the total number of selected empirical studies have nearly doubled from 2014 to 2016.

IV. 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.

V. 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.

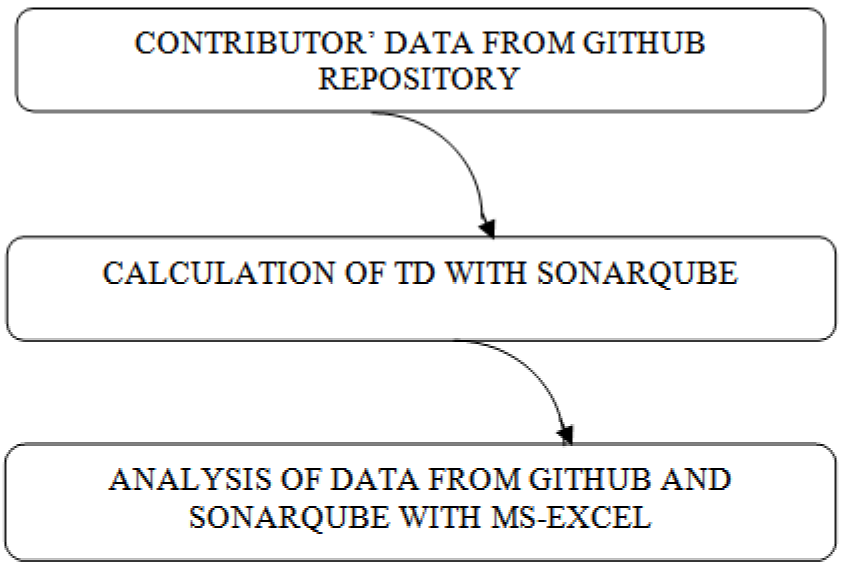
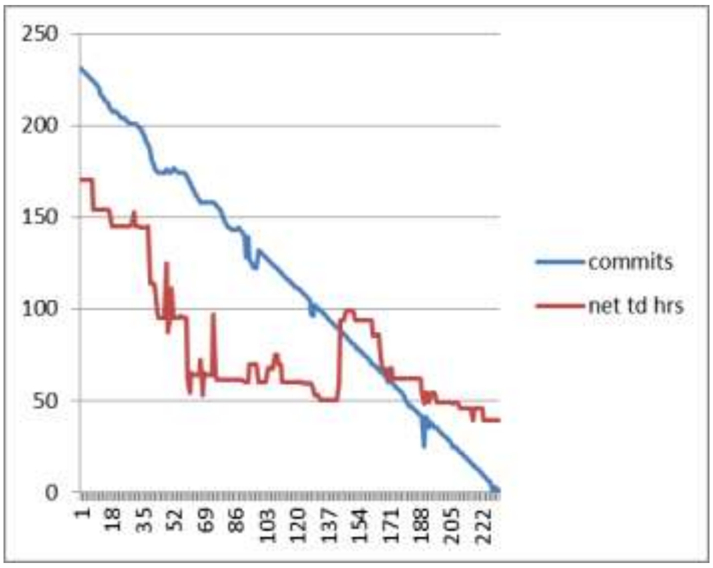
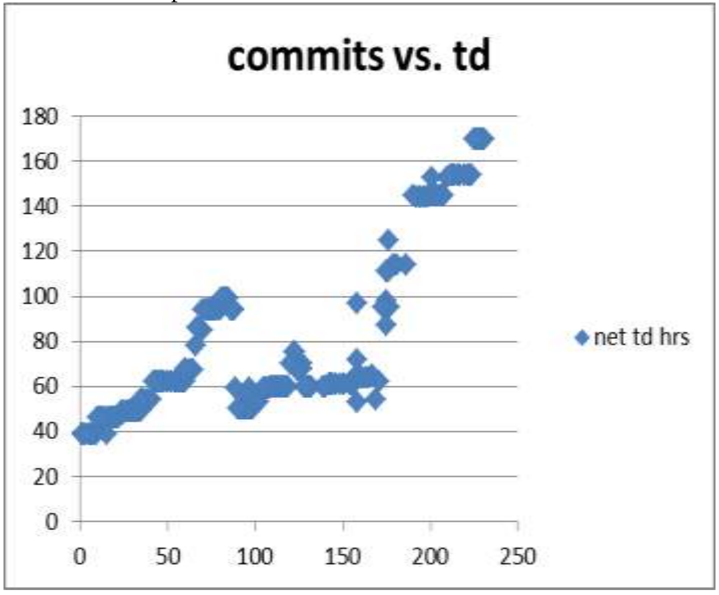

CONTRIBUTOR' DATA FROM GITHUB REPOSITORY
CALCULATION OF TD WITH SONARQUBE
ANALYSIS OF DATA FROM GITHUB AND SONARQUBE WITH MS-EXCEL

Figure 1.: process of analysis

Graph1 and graph2 were drawn to make the pictorial layout shows the relation of number of commits and that of TD.



Graph1: trend of commits and TD in hours



Graph2: TD in hours w.r.t. number of Commits

Results of correlation amongst the two variables TD and commits are tabulated in table 2. The correlation between two variables comes out to be .78 approx. which is more than .5 and is significant.

	commits	net td hrs
commits	1	
net td hrs	0.775834	1

Table1: table of correlation between TD and no. of commits

VI. 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 as the number of commits grows or shrinks accordingly there is corresponding change in TD. We cannot reduce the number of commits but some other factors influencing TD should also be considered.

VII. 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," n 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]. 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.
[22]. 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/
[23]. 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.
[24]. 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.
[25]. Thesis : Per Classon, Managing Technical Debt in Django Web Applications [2016]
[26]. 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.
[27]. Theodoros Amanatidis, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Ioannis Stamelos, Who is Producing More Technical Debt? A Personalized Assessment of TD Principal Conference Paper · May 2017
[28]. 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.
[29]. Zhixiong Gong, Feng Lyu,Technical debt management in a largescale distributed project - An Ericsson case study[June 2017]
[30]. Thesis : Sultan Wehaibi,On The Relationship Between Self-admitted debt and software quality [April 2017]
[31]. 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
[32]. 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
[33]. https://vizteck.com/blog/benefits-using-sonarqube/
[34]. https://github.com/vektra/mockery