RESEARCH REVIEW JOURNALS
Volume-04
Issue-05
May-2019

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com[UGC Listed Journal]

Relationship between Cognitive Complexity and Technical Debt: An Empirical Analysis

Jaspreet Bedi

BBK DAV College For Women, Lawrence Road, Amritsar, Dr. Kuljit Kaur, Guru Nanak Dev University, Amritsar (India)

ARTICLE DETAILS

Article History
Published Online: 25 May 2019

Keywords
Cognitive complexity(CC),technical debt(TD).

ABSTRACT

Technical debt is important concept from perspectives of developer, software practitioner, team manager, programmer and analyst. So it becomes a mandatory to conduct research on the concept by considering various aspects that have impact on technical debt. In the context of maintenance, cognitive complexity is very crucial. This paper analyzes the relations between technical debt and cognitive complexity. A lot of research has been done on the cyclomatic complexity but surprisingly cognitive aspect is ignored. The study is performed on vektra mockekery PHP application. The open source github data of the versions of vektra mockery is taken and analysed by using statistical tools of excel and graphical tools of origin. The empirical study sheds light on previously unexplored factor of cognitive complexity on technical debt. It presents significant results that can be used to further explore the related factors to technical debt so that control of technical debt can be done as a consequence.

1. Introduction

Highlight a section that you want to designate with a certain Technical debt is a very popular metaphor in the contemporary field of software engineering and open source systems. The term is quite significant in business world as well. The legacy systems do not suffice in dynamic software development world. Quite often there is need to add new features in the software. When you release a feature it's not the end of the job, it's just the beginning[33]. Not all programmers are perfect. There are plenty of programmers who haven't a clue about good design, and they merrily rack up technical debt without even realizing they're doing it. [32] It is bound by time deadlines. In this process of adding new features some quality factors are ignored for which the debt is accrued. The debt referred to as technical debt is sometimes essential while sometimes it is intentional. It is incurred at the rate of compound interest. Clamor has grown in such research domain these days. It is not detrimental to have technical debt always. In many situations it is prudent from a strategic point of view. The debt however if not paid for a long time will lead to a situation in which it will become hard to make a change in the system in prescribed time.

Due to high pressure to increase productivity, there is compromise and that gets reflected in the code. It is actually technical debt and it leads to low code quality which in turn leads to low morale of the team and low productivity. Technical debt sucks, and it's a particularly common problem for the teams I work with. Technical debt affects everything they do. It disrupts plans, kills productivity, and creates defects. [32] There is need to explore the contributing factors to this debt and to know their relation. It may otherwise result in intricate repercussions and in extreme cases the situation of technical bankruptcy.

Cyclomatic Complexity is one of the very popular measures for complexity. It uses mathematical model to assess methods, producing accurate measurements of the effort required to test them. Cyclomatic complexity was developed by Thomas J. McCabe in 1976 and metric indicates the maximum number of test cases needed to achieve 100% branch coverage. [33]. It is however popular but not the only measure of complexity. A pure application of this concept of complexity is therefore controversial. For this reason the cognitive complexity is still raised. Cognitive complexity is a measure of the comprehensibility of a class.[33] Cognitive Complexity. It does not have practice of using mathematical models to assess software maintainability instead it is a new metric which is formulated to more accurately measure the relative understandability of methods.[29]. Actually, it uses human judgement to assess how structures should be counted. [36]. Cognitive Complexity offers a measurement of how hard code is to understand. The cognitive complexity is determined by three rules viz.;Structures that make it possible to combine several instructions legibly in one block are ignored.,For each interruption in the flow of the code, the cognitive complexity increases by 1.For each nesting, the cognitive complexity is increased by 1.[33] Unfortunately, traditional measures such as McCabe's cyclomatic complexity don't do a great job of exposing the large-scale technical debt I see most often.[29]

The quantification of TD is an arduous task. These days some tools are available and are used in this context. Technical debt (TD) reflects technical compromises that can yield short-term benefit but may hurt health of a software system when seen at strategic level. There is a lot of research going on the large plethora of contributing factors of technical debt and the methods to reduce it. Complexity is one of the important factors. The research is needed to establish relation between technical debt and complexity measures. Earlier the emphasis was on cyclomatic complexity but now need is there to include relation of TD and cognitive complexity.

This paper will be investigating the relation of TD and cognitive complexity. The paper is organized into four sections. The first section introduces the topic. It includes the TD concept and two types of complexities. Section II is historical background. It is divided into two parts. First part includes the major milestones in the theoretical framework of technical debt

while second part covers the historical research on TD and complexity in any form. Section III comprises the objectives of the study. Section IV describes the research methodology. It also includes hypothesis. Section V includes the tools used for study. Section VI is research design and results. It includes the method of investigation in detail. Section VII includes the conclusion. The conclusion is drawn on basis of analysis. It also includes the contribution of the study and suggestion for further research. The sources consulted for the study are listed in the last section that is bibliography.

2. Historical Background

1992 is considered birth year of the metaphor TD when *Ward Cunningham* 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] . There was a large gap in the further significant research in the area till Zeller et.al[2005] studied significance of specific day of week for TD. It was concluded that programming on Friday is higly 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]. 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] further categorized 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 that specialized experience and general experience can affect the quality of a developer's work. F. Rahman and P. Devanbu[2011] conducted similar study on Ownership, experience and defects: a fine-grained study of authorship. Tom et al. [2013] conducted systematic literature review to establish a theoretical framework of technical debt and identified two elements of TD viz., code decay and design debt.

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 concluded [23]. Alves et al.[2015] investigated the impact of developers on introduction of code smells in five open source software systems [24]. Developers were classified in groups based on two characteristics viz., developer participation and developer authorship; calculated as time interval between his first and last commit and amount of modified files and lines of code respectively. The authors investigated relation of these characteristics with insertions, deletions and types of code smell viz., dead (unused) code, large classes, long methods, long parameter list (of methods) and unhandled exceptions. It was concluded that groups with fewer participation in code development tended to have a greater engagement in introducing and removing code smells. Authors supported that groups with higher participation level code more responsibly than others.

Everton da S. Maldonado [September 2016] in Thesis included Self-Admitted Technical Debt. Alves et al. [26] studied the strategies followed by different researches 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]. 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.

These are the major milestones in the conceptual framework of TD. As for as complexity is concerned not much has been done. There was an article on TD and complexity by Yammer' CTO ," *For years, the two things that most frustrated me to hear from product managers were "how hard would it be…" and "can't you just…"[30] Dag Liodden speaking at first Marks' summit pointed*" A component or system slowly devolves into unnecessary complexity through lots of incremental changes, often exacerbated when worked upon by several people who might not fully understand the original design."[31] a lot has been done on the theoretical framework of technical debt but the role of factors affecting technical debt are lacking in the literature.

3. Objectives of the study

- To study the relationship between Cognitive Complexity and Technical Debt.
- To study the impact of Cognitive Complexity on Technical Debt

4. Research Methodology

In order to relationship and impact of CC on TD, correlation and regression tools of statistics have been applied. It is clear that the as the cognitive complexity of software will increase there will be a tendency of the developer to ignore or overlook many aspects when there is a need to make a change in the software. As the new requirement cannot be introduced or fulfilled without any time deadline. But it may surely increase complexity as lines of code may highly likely increase and so is the role of duplications. In a hurry all other aspects like clean, readable and concise code of TD will be ignored thereby increasing the cognitive complexity. So it is expected that TD will be increasing. So, hypothesis can be framed as under:

H_0: No significant relation exists between CC and TD.

5. Tools Used

Following are the tools used for carrying out analysis in current study:

a. SonarQube

SonarQube is a web-based open source platform which is used for measuring and analyzing the source code quality. It is maintained by SonarSource. The tool is written in java. It can analyze and manage code of twenty plus programming languages. More than 50 plugins are available for extending the functionality of the tool. SonarQube receives files or projects as input, analyzes them and calculates a set of metrics. It stores output in a database and shows the same on a dashboard.

b. MS-Excel

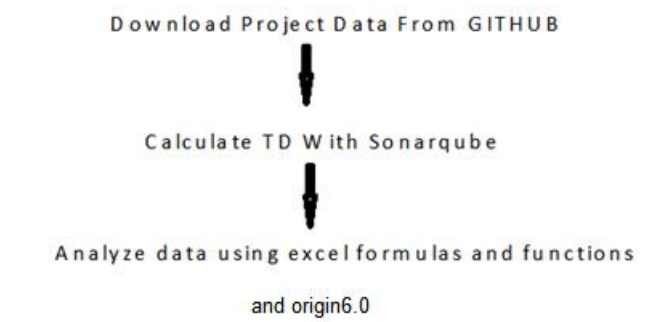
MS EXCEL 2010 was used for storage and analysis of the results. A number of mathematical and statistical functions are there in the library of Excel.. Graph drawing features were of great help for knowing the relation of the variables considered for study. Regression function of excel analysis pack.

c. Origin6.0

Origin 6.0 is a graphic drawing tool. Origin is a proprietary computer program used for data analysis. Origin Lab Corporation.[34]

6. Research Design And Results

Figure 1: steps for analysis



Vetkra Mockery one of the PHP applications in the top twenty list from github is chosen for the current study. It 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. The data of all 231 projects of Vektra Mockerywas collected from github. Sonarqube was used to find technical debt and cognitive complexity for all projects. The outputs from sonar dashboard were entered in Excel worksheet. TD calculated was standardized using mathematical functions of excel as the data was in different units of time that is days, hours and minutes.

The process of analysis is a three step process and is depicted in fig 1.

All contributors of application were considered. Figure2 shows the trend of TD and the CC in the form of line graph. It is clear that with increase in cognitive complexity, the technical debt will increase. Table1 shows the correlation coefficient of cognitive complexity and technical debt comes out to be 0.855847>0.5 it is indicating that the two terms are strongly correlated. It provides the answer to first research question.

F-value (.000) in Table 2 indicates that the regression model is fit to explain the relationship between variables. Positive relationship between TD and cognitive complexity is there. It is found to be statistically significant (p-value = .000) as well. The correlation coefficient is found is shown in table 2.

Figure2: Line Graph of TD and CC

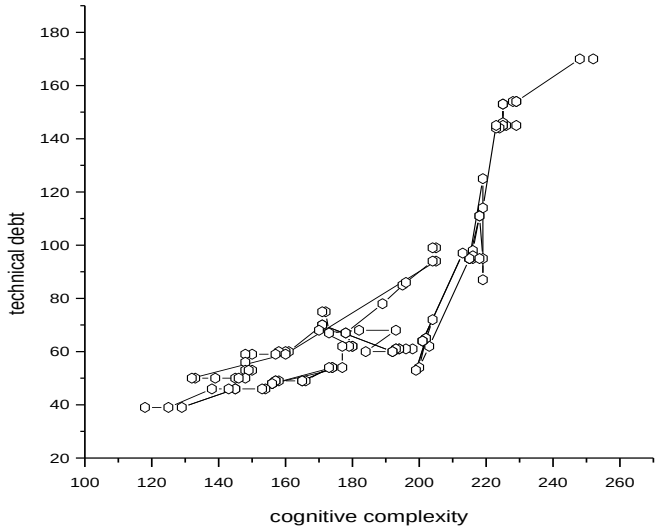


Table1:correlation table for TD and CC

	TD	*CC*
TD	1	
CC	0.855847	1

Table2: Regression Statistics

Multiple R	0.855847
R Square	0.732475
Adjusted R Square	0.731307
Standard Error	18.89592
Observations	231

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	-104.492	7.484546	-13.9611	1.85E-32	-119.24	-89.7449	-119.24	-89.7449
cogcompl	0.991562	0.039599	25.03986	1.67E-67	0.913536	1.069588	0.913536	1.069588

Table3: ANOVA table

	Df	*SS*	*MS*	*F*	*Significance F*
Regression	1	223872.1	223872.1	626.9945	1.67E-67
Residual	229	81765.81	357.056		
Total	230	305637.9			

The null hypothesis of on significant relation between CC and TD can be rejected on the basis of results obtained in table 1, 2 and 3.

7. Conclusion

Cognitive complexity is an important aspect of code which is not given due importance in research. Being novel or due to research on cylcomatic complexity aspect, the cognitive aspect of code is ignored. In context of technical aspect however the impact of cognitive complexity should be considered. To study factors affecting technical debt is a demanding area of research in software engineering. This study contributes a lot from research point of view. It further suggests that research is needed in future to discover the relation of TD with other aspects of code. From the present study it can be well concluded that TD is strongly correlated to cognitive complexity. The cognitive complexity has a positive impact on TD.

References

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. 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.
10. 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.
11. Nitin Taksande, “Thesis: Empirical study on technical debt as viewed by software practitioners” [2011]
12. Strutz, N., “Technical Debt - it's still a bad thing, right?”, in The Dopefly Tech Blog. 2011.
13. 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.
14. F. Rahman and P. Devanbu, “Ownership, experience and defects: a fine-grained study of authorship,” Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu, USA, 2011, p. 491.

15. P. Runeson, M. Host, A. Rainer, and B. Regnell, “Case Study Research in Software Engineering: Guidelines and Examples”, 1st ed. Wiley Publishing, 2012.
16. 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.
17. 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
18. 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.
19. 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.
20. 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.
21. 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.
22. Theodoros Amanatidis, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Ioannis Stamelos, “Who is Producing More Technical Debt? A Personalized Assessment of TD Principal “ Conference Paper · May 2017
23. 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.
24. Zhixiong Gong, Feng Lyu, “Technical debt management in a largescale distributed project - An Ericsson case study”[June 2017]
25. Thesis : Sultan Wehaibi, “On the relationship between self-admitted technical debt and software quality” [April 2017]
26. 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
27. https://vizteck.com/blog/benefits-using-sonarqube/
28. https://github.com/vektra/mockery
29. https://medium.com/@nickboyce/on-technical-debt-complexity-and-opportunity-cost-c767df1ffdef
30. file:///C:/Users/hp/Desktop/td%20abt/Technical%20Debt%20is%20a%20Systemic%20Problem%20_%20Agile%20Alliance.html
31. https://hackernoon.com/there-are-3-main-types-of-technical-debt-heres-how-to-manage-them-4a3328a4c50c
32. https://www.jamesshore.com/Blog/An-Approximate-Measure-of-Technical-Debt.html
33. https://www.triology.de/en/blog-entries/code-review-static-code-analysis
34. https://en.wikipedia.org/wiki/Origin_(data_analysis_software)

