*Quest Journals*
*Journal of Software Engineering and Simulation*
*Volume 7 ~ Issue 7 (July 2021) pp: 42-47*
*ISSN(Online) :2321-3795 ISSN (Print):2321-3809*
www.questjournals.org

**Research Paper**

# A Study on Relation between LOC Metric and Technical Debt

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

***ABSTRACT****: In rapidly growing software industry and fast pace of changes, time and deadlines are playing pivotal role in the survival and success of organizations. The quick and dirty approach is very popular amongst the software practitioners. It includes copy and paste criteria at many situations like at lines, blocks or files level of code or may be used for the comments. It leads to technical debt which has to be paid later. It includes principles amount and the interest as well. This may be due to many reasons viz., lack of experience, issues, code smells etc. one of the major aspect in this regard is LOC, the lines of code. Study and analysis of this factor is ignored in the literature. The objective of this study is to explore the relationship between LOC and TD. To unveil effect of LOC metrics, SonarQube tool is used for finding technical debt and for calculating technical debt. Origin 6.0 tool is used to graphically analyse the data. MS Excel is also used to calculate the regression and correlation coefficient. The dataset taken for the purpose is open source version control system from github. It is proved through the study that increases in number of lines of code increases the technical debt.*
***Keywords****: Technical debt (TD),LOC, Do It Yourself (DIY).*

## I. Introduction

### 1.1 LOC

Lines of Code (LOC) refer to any line in a program that is not a comment, blank line, or header. This includes lines with variable declarations, as well as both executable and non-executable statements, regardless of how many statements appear on a single line.Since LOC measures only the volume of code, it's useful for comparing or estimating the size of projects only when they're written in the same programming language and follow the same coding standards.

### 1.2 Features of Lines of Code (LOC)

1 **Change Tracking:** LOC can be used to monitor changes in a codebase over time, helping to track growth or reduction and offering insights into project progress.
2 **Limited Reflection of Complexity:** While LOC gives a basic sense of code size, it doesn't reflect complexity. Two programs with the same LOC can differ significantly in how complex they are.
3 **Simple to Measure:** LOC is quick and easy to calculate, making it a convenient metric for developers.
4 **Easily Understood:** Representing code size in terms of lines is straightforward and accessible, even for non-technical stakeholders.

### 1.3 Advantages of Lines of Code (LOC)

1 **Effort Estimation:** LOC can be used as a rough starting point for estimating development effort and setting project timelines. While it's not precise, it can help in early planning stages.
2 **Comparative Analysis:** LOC enables high-level comparisons of productivity across different projects or development teams by indicating the amount of code produced over time.
3 **Benchmarking Tool:** When evaluating different versions of the same program, LOC can serve as a benchmark to assess how changes impact the overall size of the codebase.

Lines of Code (LOC) is a metric that measures the size of a codebase by counting all non-comment, non-blank lines, including variable declarations and all types of statements. It's useful for tracking code growth over time and is simple to calculate and interpret. However, LOC does not account for code complexity and is most effective when comparing projects written in the same language and following consistent coding standards.

### 1.4 Technical Debt

Debt is a popular term of the financial world. It is considered with respect to some principal amount. During its payment, an interest term is also added. Similarly in technical world of software engineering the debt is considered in the software development process when some changes are required. Often there is need to make the changes during software development and consequently the system should adapt to these changes. During

this process intentionally or unintentionally 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] in his blog posted famous TD quadrant concept. 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.

Curtis [2012] provides the definitions of Technical Debt in terms of 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. Principle 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.

**1.5 Reasons / Causes for TD**

Researchers have mentioned many reasons for TD. Dan Rawsthorne [2012] 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 lack of test, bad design, lack of documentation and poor readability as the reasons for technical debt. Similarly Stephanie [2015] in his blog 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 like time pressures, Overly complex technical design, Poor alignment to standards, Lack of skill, Suboptimal code, Delayed refactoring and Insufficient testing. However general agreement on the types does not exist. Similar to many definitions of TD metaphor there are many types of technical debt.

**1.6 Types of TD**

From perspectives of software development and business etc., literature includes many types of TD and to deal with the issues related to technical debt classification of technical debt is needed. 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. Somename 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 McConnel [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.

## II. Organization Of Paper

` The first section of the study introduces the topic by considering various references from related studies. The second section describes the organization of the paper. The third section is discussion of background from literature. The fourth section frames research questions. The fifth section of the paper is results and discussion. It also includes the statistical summaries. The sixth section concludes while seventh section of the paper lists the resources consulted for the study.

## III. Background

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] 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." 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 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. Martin Fowler [2009] introduced concept of TD quadrant for categorizing TD. He divided 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. They 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.

Alves et al [2015] investigated the influence of developers on the introduction of code smells in 5 open source software systems. Developers have been classified in different groups based on two characteristics, namely developer participation (calculated as the time interval between his first and last commit) 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 viz., 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 group with higher participation level code more responsibly during maintenance whereas the other groups tend to focus on error correction actions.

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. MrwanBenIdris[2018] pointed that the total number of selected empirical studies have nearly doubled from 2014 to 2016. The popularity of the term technical debt is on rise since few years. Software developers and managers increasingly use the concept to communicate key tradeoffs related to release and quality issues. This concept is clear from the fact that the concept of technical debt is related to software quality and research is on rise in new tools and techniques required for estimating technical debt.

## IV. Research Questions

The visitors or users put a if there is something apealing/interesting in the verson or commit is there. The technical debt of a worth project will not be too high. It is expected that the project that will be liked by many will be one involving lesser TD. So hypothesismay be framed as:

$H_0$: td will increase with growing number of LOC in a project.

$H_A$: td will not increase with growing number of LOC in a project.

Following obvious Research questions can hence be framed:

RQ#1: Does td correlate with LOC?

This research question has a goal to provide an initial investigation to determine if some correlation exists between two parameters viz., LOC and td. To achieve this goal, correlation of the two is considered

RQ#2: to know what is trend of changes and how td varies with increasing numer of LOC?

Goalis to check provide whether LOC can be assumed as an index to reduced td or not. To achieve this goal,regression as a measure is considered and the scatterplot and column graph showing the trend of change is drawn.

## V. Results And Discussions

Vetkra Mockery, one of the PHP applications in the top twenty list of github is considered for the study. 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. The data of 231 versions of Vektra Mockery 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 the application were considered. The process of analysis is a three step process and is depicted in figure 1.

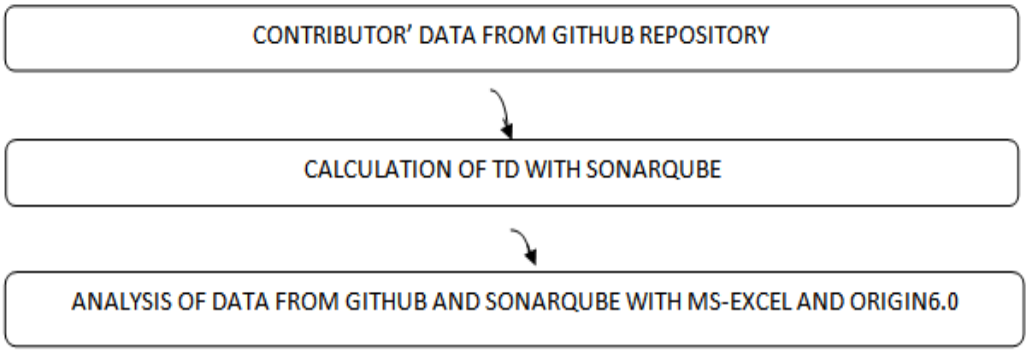


Fig1: Process Of Analysis

In order to find answer to research question RQ#1, correlation of the LOC and TD is considered individually that is does td vary with increasing numer of lines of code? So, goal is to provide to determine if some correlation exists between two parameters viz., lines of code and td. It is tabulated in table 1.

| | Loc | net td |
|---|---|---|
| Loc | 1 | |
| net td | 0.696594 | 1 |

Table 1. LOC and TD

Correlation of the two is considered and it comes out as approx .7 which is more than .5. so a positive correlation exists between LOC and TD.

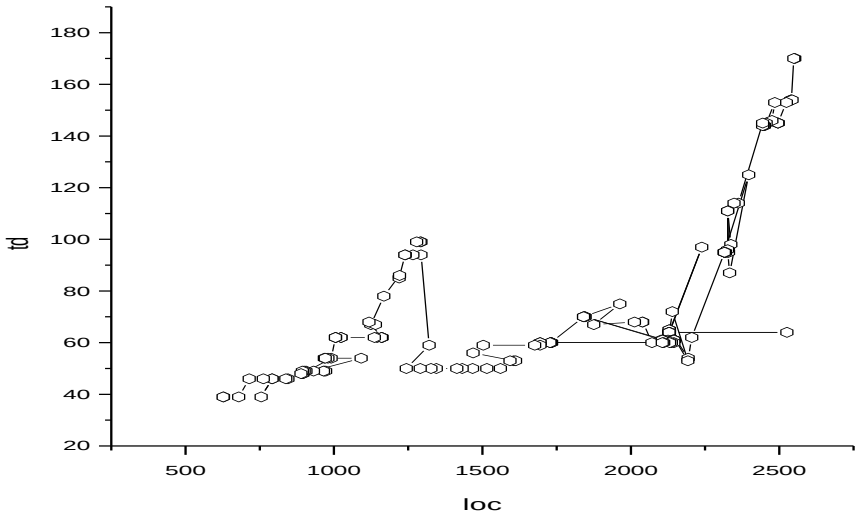


figure 2. Relation of LOC and TD

It may be concluded that technical debt will increase with increase in LOC. Hence alternate hypothesis $H_A$ is correct.

## VI. Conclusion

There are a number of factors affecting technical debt. It is an important area of research to unveil these factors. It can be well concluded that LOC strongly affects TD as correlation between TD and LOC is 0.7. it is clear from the graphical representation as well. Further research is needed to explore effects of other factors on technical debt.

## Bibliography


[1]. Atwood, J., Coding Horror: Paying Down Your Technical Debt, in programming and human factors. [2009].
[2]. Cunnigham, W., "Debt Metaphor Isn't a Metaphor, in Powers of Two", R. Mayers, Editor. [2009]
[3]. 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
[4]. Clean Code, Ugly Code, Technical Debt and CleanUp Stories By Dan Rawsthorne
[5]. 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.
[6]. Fowler, M. Technical debt quadrant, [2009]; http:// martinfowler.com/bliki/TechnicalDebtQuadrant.html.
[7]. Georgios Digkas ; Mircea Lungu ; Paris Avgeriou ; Alexander Chatzigeorgiou ; Apostolos Ampatzoglou,"How do developers fix issues and pay back technical debt in the Apache ecosystem?"[ 05 April 2018]
[8]. 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.
[9]. 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].
[10]. 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].
[11]. 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.
[12]. 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].
[13]. Markus Lindgren , [ Thesis] Bridging the software quality gap [Oct. 2012]
[14]. MrwanBenIdris, 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]
[15]. Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, IpekOzkaya, RaghvinderSangwan, 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].
[16]. O'Neill, D. "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/
[17]. 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.
[18]. P. Runeson, M. Host, A. Rainer, and B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, 1st ed. Wiley Publishing, [2012].
[19]. Per Classon, [ Thesis]Managing Technical Debt in Django Web Applications [2016]
[20]. 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
[21]. Steeve McConnell. Technical Debt. http://blogs.construx.com/blogs/stevemcc /archive/2007/11/01/technical-debt-2.aspx.
[22]. Strutz, N., Technical Debt - it's still a bad thing, right?, in The Dopefly Tech Blog. [2011].
[23]. Sultan Wehaibi,[ Thesis] :On The Relationship Between Self-admitted debt and software quality [April 2017]
[24]. TereseBesker , 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
[25]. Theodoros Amanatidis, Alexander Chatzigeorgiou, ApostolosAmpatzoglou, IoannisStamelos, Who is Producing More Technical Debt? A Personalized Assessment of TD Principal Conference Paper · [May 2017]
[26]. Tom, Edith, Aybüke Aurum, and Richard T. Vidgen. "A Consolidated Understanding of Technical debt." ECIS. [2012].
[27]. 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.
[28]. W. Cunningham,"TheWyCash Portfolio Management System", http: //c2.com/doc/oopsla92.html.
[29]. Yuepu Guo, Carolyn Seaman, Rebeka Gomes, Antonio Cavalcanti, GrazielaTonin, Fabio Q. B. Da Silva, André L. M. Santos ,ClauirtonSiebra,"Tracking technical debt — An exploratory case study", IEEE Software [2011]
[30]. Zhixiong Gong, Feng Lyu,Technical debt management in a largescale distributed project - An Ericsson case study[June 2017]
[31]. https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/
[32]. https://vizteck.com/blog/benefits-using-sonarqube/
[33]. https://github.com/vektra/mockery
[34]. https://en.wikipedia.org/wiki/Origin_(data_analysis_software)
[35]. https://tommcfarlin.com/code-smells/
[36]. https://help.github.com/en/articles/github-glossary
[37]. https://3back.com/scrum-industry-terms/the-4-types-of-technical-debt/
[38]. https://3back.com/development/clean-code-ugly-code/
[39]. https://www.castsoftware.com/blog/the-causes-of-technical-debt-do-not-exist-in-a-vacuumJun 17, 2015 by Stephanie W.
[40]. https://www.codacy.com/blog/how-to-prioritize-your-technical-debt/
[41]. https://www.bmc.com/blogs/technical-debt-explained-the-complete-guide-to-understanding-and-dealing-with-technical-debt/
[42]. http://hackernoon.com

[43]. https://www.ndepend.com/docs/technical-debt
[44]. https://www.appian.com/blog/tactically-tackling-the-issues-of-technical-debt/
[45]. https://raygun.com/blog/manage-technical-debt/
[46]. http://modernperlbooks.com/mt/2009/02/backwards-compatibility-is-technical-debt.html
[47]. https://blog.crisp.se/2013/10/11/henrikkniberg/good-and-bad-technical-debt
[48]. https://www.cutter.com/article/defining-technical-debt-492591
[49]. https://docs.sonarqube.org/latest/user-guide/metric-definitions/
[50]. http://thinkapps.com/blog/development/technical-debt-calculation/
[51]. http://www.danube.com/system/files/CollabNet_WP_Technical_Debt_041910.pdf
[52]. https://medium.com/appsflyer/three-tips-for-managing-technical-debt-while-maintaining-developer-velocity-and-sanity-f3d4a080052c
[53]. http://www.codingthearchitecture.com/2006/12/15/code_metrics.html