A Study On Relation Between Duplication Metrics 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 principle 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 reasons of technical debt is using duplications. Study and analysis of this factor is ignored in the literature. The objective of this study is to develop a prediction model of duplication metrics for technical debt. To unveil effect of duplication metrics, SonarQube tool is used for finding technical debt and for calculating duplication metrics. 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 as we increase duplication metrics in any form there is corresponding considerable change in the technical debt.*

Keywords: Technical debt (TD), Duplication Metrics, Do It Yourself (DIY).

I. INTRODUCTION

Debt is a popular 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.

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.

Reasons and causes for TD

Researchers have mentioned many reasons for TD. Dr. 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 out that lack of test, bad design, lack of documentation and poor readability are reasons of arising technical debt. 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, 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.

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

II. 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. Mrwan BenIdris [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. RESEARCH QUESTIONS

RQ#1: Is there any relation of duplicate lines and technical debt?

The duplicate line metric measured in percentage units is needed for this research question to be answered. SonarQube tool provides it in the required form. It is expected that duplicated lines will increase code smells thereby increasing technical debt. So, we have:

H_0: duplicate lines increase technical debt.

This research question has a goal to investigate the relation of duplicate lines and the technical debt.

RQ#2: debt does any relation occurs between duplicate blocks and technical debt?

It is expected that duplicated blocks will increase code smells thereby increasing technical debt. Since block is made up of collection of lines.

So, we have:

H_0: duplicate blocks increase technical debt.

This research question has a goal to investigate the relation of duplicate blocks and the technical debt.

RQ#3: are duplicate files and technical debt related?

It is expected that duplicated files will increase code smells thereby increasing technical debt. As file is a collection of blocks.

So, we have:

H_0: duplicate files increase technical debt.

This research question has a goal to investigate the relation of duplicate files and the technical debt.

RQ#4: are duplication density and technical debt related to each other?

To know the relation of the two variables correlation coefficient was calculated.

III. METRICS

Duplications are described in research literature in terms of Duplicated blocks(Number of duplicated blocks of lines.), Duplicated files (Number of files involved in duplications.), Duplicated lines (Number of lines involved in duplications.) and Duplicated lines (%) (duplicated_lines / lines * 100)[13] here it is assumed that for Java projects duplication is said to occur if there are at least 10 successive and duplicated statements ignoring the number of tokens and lines, differences in indentation and in string literals etc. while for Non-Java projects there should be at least 100 successive and duplicated tokens and Those tokens should be spread at least on 30 lines of code for COBOL, 20 lines of code for ABAP or 10 lines of code for other languages. Sonar uses the following four metrics to cover code duplication. Duplicated lines can also be expressed as a percentage value.

Name	Description
Duplicated Lines	Number of physical lines touched by duplication.
Duplicated Blocks	Number of duplicated blocks participating in duplication.
Duplicated Files	Number of files containing duplicated lines or blocks.
Density of Duplicated Lines	Density =(Duplicated lines/physical lines)*100

Table1: Source: https://www.oreilly.com/library/view/sonar-code-quality/9781849517867/ch08s02.html

All these metrics are considered for analysis. The relation of these variables with technical debt is considered.

IV. 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.

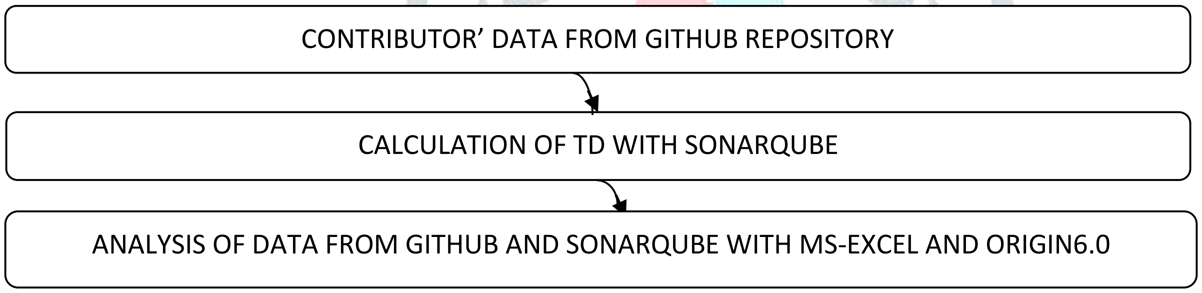
CONTRIBUTOR' DATA FROM GITHUB REPOSITORY
CALCULATION OF TD WITH SONARQUBE
ANALYSIS OF DATA FROM GITHUB AND SONARQUBE WITH MS-EXCEL AND ORIGIN6.0

Fig1: Process Of Analysis

In order to find answer to research questions, correlation of the all duplication metrics and TD is considered individually and tabulated in figure2. A strong positive relation exists between all duplication metrics and TD. It shows that increasing number of duplicated lines, duplicated blocks and duplicated files will increase the TD. It further indicates that null hypothesis of each research question needs to be accepted. It is however not possible to reduce the number of lines of codes and number of commits in the growing projects. As the new version is undoubtedly bound to increase number of commits but as a caution the developer should use "do not repeat yourself" approach.

	Duplines	TD
Duplines	1	
TD	0.718389	1

Figure: Correlation table for Duplicated Lines and technical debt

	Dupblocks	TD
Dupblocks	1	
TD	0.611045	1

Figure: Correlation table for Duplicated Blocks and technical debt

	Dupfiles	TD
Dupfiles	1	
TD	0.693218	1

Figure: Correlation table for Duplicated Files and technical debt

	Dupdensity	TD

Dupdensity	1	
TD	0.599984	1

Figure: Correlation table for Density of Duplicated Lines and technical debt

Figure2: Correlation matrix Of Metrics of duplication And TD

From Figures (Figure (3-6)) it is clear that there is an increasing trend of td w.r.t. all types of duplicate metrics however the trend is not uniform.

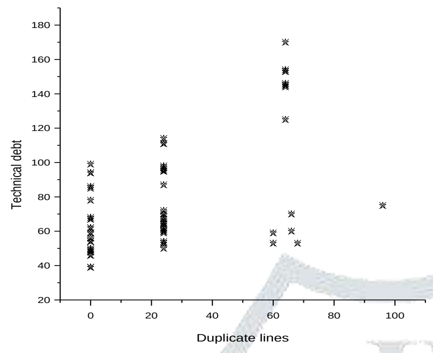

Figure3: Trend of changing TD w.r.t. duplicated lines

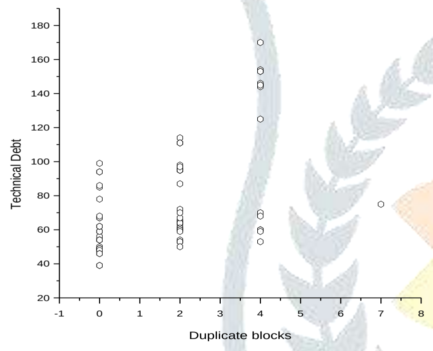

Figure4: Trend of changing TD w.r.t. duplicate blocks

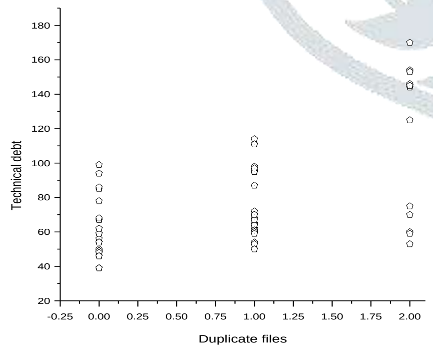

Figure5: Trend of changing TD w.r.t. duplicate files

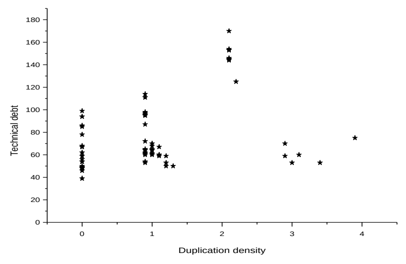

Figure6: Trend of changing TD w.r.t. duplicate density

The overall effect of duplicated metrics on TD may be combined in the following regression table(Figure 7):

Regression Statistics	
Multiple R	0.89705
R Square	0.804699
Adjusted R Square	0.801242
Standard Error	16.2518
Observations	231

	Df	SS	MS	F	Significance F
Regression	4	245946.6	61486.64	232.7971	6.5E-79
Residual	226	59691.37	264.1211		
Total	230	305637.9			

ANOVA TABLE

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	60.90858	1.619846	37.60145	3.1E-99	57.71664	64.10051	57.71664	64.10051
Duplines	4.799305	0.285042	16.8372	9.08E-42	4.237625	5.360984	4.237625	5.360984
Dupblocks	-6.73287	2.332847	-2.88612	0.004278	-11.3298	-2.13595	-11.3298	-2.13595

Dupfiles	14.45627	7.622307	1.896575	0.059159	-0.5636	29.47615	-0.5636	29.47615
Dupdensity	-104.951	6.331713	-16.5755	6.47E-41	-117.428	-92.4744	-117.428	-92.4744

Fig 7:Regression Statistics

F-value (.000) in Table indicates that the regression model is fit to explain the relationship between variables. This relationship can be expressed in regression equation as:

$$TD = 4.799305 * duplines - 6.73287 * dupblocks - 104.951 * dupdensity + 14.45627 * dupfiles$$

The regression equation shows the negative relationship between TD and duplicated blocks, TD and duplicated density on the one hand and TD and duplicated lines and TD and duplicated files on the other. However, there is strong positive relationship between TD and all types of duplicated metrics statistically significant (p-value = .000) as well. Coefficient of determination (Adjusted R square) shows that approximately 80 % variation in TD is due to duplicated lines, duplicated blocks, duplicated files and duplication density.

V. 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 duplication metrics viz., duplicate lines, duplicate blocks, duplicate files, duplicate density effect TD. It is statistically proved that all these factors effect TD in significant way. As these metrics grow there is corresponding increasing change in TD. Apart from these factors prediction model of these metrics for TD is developed. DIY (Do It Yourself) approach instead of 'Quick And Dirty' Approach of Copy And Paste approach should be used in order to keep a control over TD.

VI. 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) *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]*
15) *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].*
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) *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*
25) *Theodoros Amanatidis, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Ioannis Stamelos, 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,"The WyCash Portfolio Management System", http: //c2.com/doc/oopsla92.html.*
29) *Yuepu Guo, Carolyn Seaman, Rebeka Gomes, Antonio Cavalcanti, Graziela Tonin, Fabio Q. B. Da Silva, André L. M. Santos ,Clauirton Siebra,"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://vizteck.com/blog/benefits-using-sonarqube/*
32) *https://github.com/vektra/mockery*
33) *https://en.wikipedia.org/wiki/Origin_(data_analysis_software)*
34) *https://tommcfarlin.com/code-smells/*
35) *https://help.github.com/en/articles/github-glossary*
36) *https://3back.com/scrum-industry-terms/the-4-types-of-technical-debt/*
37) *https://3back.com/development/clean-code-ugly-code/*
38) *https://www.castsoftware.com/blog/the-causes-of-technical-debt-do-not-exist-in-a-vacuum*Jun 17, 2015 by Stephanie W.
39) *https://www.codacy.com/blog/how-to-prioritize-your-technical-debt/*
40) *https://www.bmc.com/blogs/technical-debt-explained-the-complete-guide-to-understanding-and-dealing-with-technical-debt/*
41) *http://hackernoon.com*
42) *https://www.ndepend.com/docs/technical-debt*
43) *https://www.appian.com/blog/tactically-tackling-the-issues-of-technical-debt/*
44) *https://raygun.com/blog/manage-technical-debt/*
45) *http://modernperlbooks.com/mt/2009/02/backwards-compatibility-is-technical-debt.html*
46) *https://blog.crisp.se/2013/10/11/henrikkniberg/good-and-bad-technical-debt*
47) *https://www.cutter.com/article/defining-technical-debt-492591*
48) *https://docs.sonarqube.org/latest/user-guide/metric-definitions/*
49) *http://thinkapps.com/blog/development/technical-debt-calculation/*
50) *http://www.danube.com/system/files/CollabNet_WP_Technical_Debt_041910.pdf*
51) *https://medium.com/appsflyer/three-tips-for-managing-technical-debt-while-maintaining-developer-velocity-and-sanity-f3d4a080052c*
52) *http://www.codingthearchitecture.com/2006/12/15/code_metrics.html*