

# An Empirical Evaluation of the Relationship between Technical Debt and Software Security

Miltiadis Siavvas<sup>\*†</sup>, Dimitrios Tsoukalas<sup>†§</sup>, Marija Jankovic<sup>†</sup>, Dionysios Kehagias<sup>†</sup>,  
Alexander Chatzigeorgiou<sup>§</sup>, Dimitrios Tzouvaras<sup>†</sup>, Nenad Anicic<sup>‡</sup>, Erol Gelenbe<sup>\*</sup>

<sup>\*</sup> Imperial College London, SW7 2AZ, London, United Kingdom

<sup>†</sup> Centre for Research and Technology Hellas, Thessaloniki, Greece

<sup>§</sup> Department of Applied Informatics, University of Macedonia, Thessaloniki 54643, Greece

<sup>‡</sup> Faculty of Organizational Sciences, University of Belgrade, Belgrade, Serbia

m.siavvas16@imperial.ac.uk, tsoukj@iti.gr, jankovicm@iti.gr, diok@iti.gr,  
achat@uom.gr, dimitrios.tzouvaras@iti.gr, anicic.nenad@fon.bg.ac.rs, e.gelenbe@imperial.ac.uk

**Abstract**—Technical Debt (TD) is commonly used in practice as a measure of software quality. Due to the potential overlap between software quality and software security, an interesting topic is to investigate whether TD can be used as a software security indicator as well. However, although some software-related factors (e.g. software metrics) have been studied for their ability to indicate security risk in software products, no research attempts exist specifically focusing on TD. To this end, in the present study, we empirically evaluate the ability of TD to indicate security risks in software products. For this purpose, a relatively large code repository comprising 50 open-source software applications was constructed and analyzed using popular open-source static analysis tools, in order to calculate their TD and security level (i.e. vulnerability density). Subsequently, statistical analysis was employed, to assess the relationship between TD and software security. The results of the empirical study revealed a statistically significant positive and strong correlation between the TD and the vulnerability densities of the studied software products. This provides preliminary evidence for the ability of TD to be used as an indicator of software security. To the best of our knowledge, this is the first study that empirically evaluates the relationship between TD and software security.

**Keywords**—*software security; technical debt; vulnerability prediction; empirical study; static analysis*

## I. INTRODUCTION

Secure software development demands the identification and mitigation of security risks early enough in the overall software development lifecycle (SDLC) [1]. For this purpose, appropriate mechanisms are needed to help software developers detect and remove potential security issues before the final release of software products [2], [3]. Vulnerability prediction is a common mechanism, which focuses on predicting the existence of various security issues (i.e. vulnerabilities) in software products. This information may be leveraged by software developers and engineers for prioritizing their testing and fortification efforts. Even though it is a relatively new area of research, numerous research attempts have been conducted, focusing chiefly on the ability of several software-related factors (e.g. software metrics) to predict the existence of vulnerabilities in software [4]–[9].

Technical Debt (TD) [10] is an appealing factor which deserves individual merit regarding its ability to indicate security risks in software products. TD is inspired by the financial debt and used to quantify long-term quality problems of software products, which are caused by quality compromises that provide short term benefits. Strict production deadlines often force developers to focus mainly on the code they produce, and therefore neglect its quality, leading to the introduction of design and code quality issues. Design quality issues are often termed as code smells, while code quality issues refer to poorly written code that violates the best coding practices or coding rules. These quality issues require future rework for their correction. Systematically avoiding fixing the quality issues leads to the aggregation of the effort required for their mitigation. TD is a measure of the effort required for fixing these issues in the future, and is used as a measure of quality. Hence, the higher the TD of a software product, the more the unresolved quality issues that it contains, and the poorer its overall quality.

Consequently, it is reasonable to expect that some of the quality compromises that are made by the developers may potentially have security implications since most of the security vulnerabilities that a software product contains are caused by coding and design errors [1]. However, the relationship between different software-related factors and software security has been extensively studied in the literature (i.e. software metrics [4]–[6], text features [7], [11], or even popularity [9]). To the best of our knowledge, no studies are documenting the attempts to investigate the relationship between software security and TD.

To this end, in the present paper we attempt to empirically evaluate the relationship between TD and software security. The problem that the present work attempts to solve can be summarized in the following research question:

**RQ:** *Is Technical Debt closely related to software security?*

A positive answer to this question will suggest that TD may highlight potential security risks in software products and that it may be used as an indicator of software security. In other words, this will indicate that TD can potentially be used as the basis for the construction of vulnerability prediction models.

In order to provide answers to the aforementioned research question an empirical study was conducted. In particular, we initially constructed a relatively large code repository comprising 50 real-world open-source Java applications retrieved from the GitHub<sup>1</sup> online repository. The software applications of the resulting benchmark repository were analyzed using popular static code analyzers, to calculate their TD as well as their security level. Subsequently, statistical analysis was employed, to determine whether a statistically significant correlation exists between their TD and security level. Hypothesis testing was also employed, in order to reach safer conclusions regarding the significance of the observed results. To the best of our knowledge, this is the first study in the field of vulnerability prediction that investigates the relationship between TD and software security.

The remainder of the paper is structured as follows: Section II discusses the related work in the field of vulnerability prediction. Section III describes the experiment setup, while Section IV presents the results of the experiment. Finally, Section V concludes the paper and discusses ideas for future work.

## II. RELATED WORK

Numerous research endeavors can be found in the field of software security focusing on the ability of several factors to indicate security risks (e.g. vulnerabilities) in software products. The vast majority of them emphasized specifically on the ability of common software metrics to predict the existence of security issues in software components. In particular, Shin et al. [4], [5] and Chowdhury et al. [12] were the first to observe the ability of common coupling, cohesion, and complexity (CCC) metrics to indicate security risks in software products. Based on these metrics they also proposed vulnerability prediction models, which demonstrated promising results [13], [14]. These observations were supported by the results of a large number of follow-up studies (e.g. [6], [15]), which also considered additional metrics.

Apart from software metrics, significant emphasis was also given on static information retrieved directly from the source code, including keywords and static analysis alerts. Neuhaus et al. [7] were the first to investigate the ability of specific keywords (i.e. import statements and function calls) to indicate the existence of security issues in software components. Based on their observations, more elaborate studies focusing on text-mining of source code have been conducted (e.g. [11]). Regarding static analysis alerts, the most representative attempt was made by Gegwick et al. [8], who showcased that a close relationship exists between the densities of security-relevant static analysis alerts and actual vulnerabilities. This observation was also supported by recent studies (e.g. [16]).

Recently, the research community shifted its focus towards factors that are not directly related to the source code of software applications. For instance, Roumani et al. [17] examined the relationship between the firm's financial records (e.g., size, financial performance, sales, research and development expenditures etc.) and security vulnerabilities that may exist in their software products, revealing a strong association between these two factors. In addition, in a recently published empirical study, Siavvas et al. [9] examined the relationship between the popularity of open-source software applications and their security level (i.e. vulnerability density), leading to the conclusion that popularity may not be a reliable indicator of software security.

Although several highly divergent factors have been studied for their ability to indicate security risks in software products, no research attempt exists explicitly focusing on TD. Therefore, the purpose of the present study is to empirically evaluate the relationship of TD with software security, and the ability of TD to indicate security risks in software products.

## III. EXPERIMENT SETUP AND METHODOLOGY

### A. Benchmark Repository

For the present study, a relatively large benchmark repository of software applications was constructed. In particular, we mined the online GitHub repository and retrieved a large number of real-world open-source software applications written in Java programming language based on their popularity. From the retrieved software applications, we kept only those that were able to compile without any errors or warnings successfully. This resulted in a benchmark repository of 50 open-source Java applications, comprising approximately 1.7 million lines of code. It should be noted that compilation was necessary since the static code analyzers that were used in the present study for calculating the TD and the security level of the software products require the binary files of the applications to execute.

### B. Indicator of Software Security

The Static Analysis Vulnerability Density (SAVD) [18] was selected as an indicator of software security. The Vulnerability Density metric [19] is defined as the total number of vulnerabilities that a software product contains per thousand lines of code. The SAVD is the Vulnerability Density metric that is calculated based on vulnerabilities reported by static analysis tools. SAVD has been widely used in the literature as an indicator of software security [6], [9], [20], as well as for the conduction of vulnerability prediction [8].

In order to quantify the SAVD of the selected software applications, the FindBugs [21] static code analyzer was utilized. FindBugs is a popular open-source static analysis tool for Java applications, widely used in the literature for security auditing purposes (e.g. [9]). Similarly to [9], the tool was properly configured in order to detect only security-relevant issues (i.e. bugs), which belong to the bug categories of *Performance*, *Malicious Code*, and *Multithreaded Correctness*

<sup>1</sup><https://github.com>

provided by FindBugs. We also included another 128 security-related issues that belong to the *Security* bug category provided by the FindSecurityBugs<sup>2</sup> plugin, which is a popular FindBugs plugin.

### C. Indicator of Technical Debt

Several models and methods for the calculation of TD have been proposed over the years [22], with the SQALE method to be the most popular among them [23]. For this purpose, in the present work we decided to use the SQALE method for quantifying the TD of the selected software products, and particularly the SQALE Index plugin that is provided by SonarQube<sup>3</sup>, which is a popular static analysis platform for detecting software issues (e.g. bugs, code smells, etc.) and for monitoring TD.

The SQALE Index, since it is a measure of TD, quantifies the effort that is required by the developers in order to fix all the code smells that reside in the analyzed software product. This effort is expressed in minutes by SonarQube. However, it is evident that the actual value of the SQALE Index depends on the size of the software products. Larger software products are expected to have more quality issues, and, in turn, more effort will be required for their correction, which is reflected as higher TD. Hence, in order to avoid being biased by the size of the software products, and to make the products directly comparable concerning their TD, similarly to [24], we decided to normalize the SQALE Index by the size (i.e. lines of code) of the software products. We term the normalized SQALE Index as SQALE Density and we use it as the indicator of TD.

### D. Statistical Analysis

SonarQube and FindBugs were employed in order to calculate the SQALE Density and the SAVD of each one of the software products that reside in the selected benchmark repository. Based on these values, two individual rankings of the selected products were produced, one based on their TD (i.e. SQALE Density) and another based on their security (i.e. SAVD). The two rankings were compared using the *Spearman's rank correlation coefficient* ( $\rho$ ) [25], which is a non-parametric and non-sensitive to outliers test. In order to assess the strength of the calculated correlation the thresholds proposed by Cohen et al. [26] were used. According to Cohen et al. [26], a correlation value higher than 0.5 is considered strong, between 0.3 and 0.5 is considered moderate, and below 0.3 is considered low. The statistical significance of the results was tested at 95% level of confidence. Similar approaches have been used in the related literature (e.g. [4], [9], [27]).

## IV. RESULTS AND DISCUSSION

In Table I the selected software applications, along with their SQALE Densities, SAVDs, and their corresponding rankings are presented. As already mentioned, in order to provide answer to the formulated *RQ*, statistical testing

was applied. More specifically, we initially calculated the *Spearman's rank correlation coefficient* ( $\rho$ ) between the two rankings, in order to examine whether there is any relationship between TD and software security. In order to reach safer conclusions regarding the significance of the observed relationship, the following null hypothesis was formulated and tested at the 95% ( $\alpha = 0.05$ ):

$H_0$ : No statistically significant correlation exists between the two rankings.

$H_1$ : A statistically significant correlation exists between the two rankings.

TABLE I: The SQALE Densities, the SAVDs and the corresponding rankings of the analyzed open-source software applications.

Project Name	SQALE Density	SAVD	SQALE Ranking	SAVD Ranking
aho-corasick	0.100751	0	12	1
BIMserver	0.320257	1.586809	48	33
docker-client	0.09464	0.339591	11	5
emoji-java	0.122463	0.532765	15	10
EMV-NFC-Paycard-Enrollment	0.080121	1.010729	10	19
essentials	0.218784	2.356457	33	49
fast-serialization	0.434894	1.645736	50	37
ffmpeg-cli-wrapper	0.152304	1.312203	22	22
geohash-java	0.157101	0.8663	24	16
GeolIP2-java	0.071523	0.4	6	9
gifencoder	0.049389	0.363636	4	7
HotswapAgent	0.270131	1.533333	44	29
hutool	0.182167	1.542039	29	31
imgscalr	0.160807	1.128243	26	20
incubator-dubbo	0.343403	1.766615	49	44
infinitest	0.136301	0.856664	17	15
j2v8	0.243073	2.066902	40	47
Java-Chronicle	0.277928	1.581333	46	32
java-client-api	0.176241	1.352874	28	24
java-speech-api	0.143185	1.528013	19	27
JavaVerbalExpressions	0.032989	0	2	1
jcabi-aspects	0.045878	0.798722	3	12
jesque	0.182275	1.728866	30	43
jimfs	0.079692	0	9	1
JInstagram	0.148015	1.66372	21	39
jmustache	0.220505	2.060439	35	46
jongo	0.155766	0.851498	23	14
JSqlParser	0.077733	0.249231	8	4
lanterna	0.165803	1.59815	27	34
mapstruct	0.116729	0.730215	13	11
mbassador	0.160088	1.382579	25	26
mp3agic	0.220133	1.633259	34	36
mybatis-3	0.273134	1.813956	45	45
netty-restful-server	0.248489	1.697235	41	42
obd-java-api	0.225306	1.368791	38	25
okhttp	0.221066	1.337507	36	23
open-replicator	0.142396	1.528384	18	28
opsu	0.22425	2.278014	37	48
paho.mqtt.java	0.278734	1.536873	47	30
pcap4j	0.147341	1.244906	20	21
spring-roo	0.215516	1.667379	32	40
ta4j-origins	0.077285	0.396877	7	8
takes	0.025682	0.831699	1	13
threetenbp	0.122252	0.351932	14	6
thumbnailator	0.056834	0.89481	5	18
tracer	0.123737	0.872846	16	17
traccar	0.267268	1.658833	43	38
unirest-java	0.209792	1.695142	31	41
webbit	0.251977	2.503507	42	50
XChart	0.251977	2.503507	42	50
xmemcached	0.230889	1.609042	39	35

The calculated Spearman's rank correlation coefficient between the two rankings was found to be  $\rho = 0.808$ , which is a positive and strong correlation according to the suggestions of Cohen et al. [26]. In addition, the *p-value* of the test was found to be  $1.26 \times 10^{-12}$ , which is significantly smaller than the threshold of 0.05. This led us to the rejection of the null hypothesis and to the acceptance of the alternative hypothesis, which suggests that the observed relationship between the two rankings is statistically significant.

From the results of the above analysis, we can conclude that a statistically significant positive correlation exists between the

<sup>2</sup><https://find-sec-bugs.github.io/>

<sup>3</sup><https://www.sonarqube.org/>

two rankings (at least for the studied benchmark repository). In other words, the SQALE Densities of the selected software products were found to be closely related to their SAVDs. This provides empirical evidence that a strong relationship exists between TD and software security, while it may also indicate that TD may be used as an indicator of software security. This opens a new area of research on whether TD can be used as the basis for the construction of vulnerability prediction models, able to highlight the existence of potential vulnerabilities in software products.

## V. CONCLUSION

In the present paper, we empirically evaluated the relationship between Technical Debt (TD) and software security. For the empirical study, a relatively large benchmark repository comprising 50 real-world open-source Java applications retrieved from GitHub was constructed. Subsequently, the software applications of the resulting benchmark repository were analyzed using the SonarQube and FindBugs static code analysis tools, to determine their SQALE Density (see Section III-C) and Static Analysis Vulnerability Density (SAVD) [18], which were used as indicators of the TD and security levels respectively. Two individual rankings were produced, one based on the SQALE Densities of the studied applications and another one based on their SAVDs, and were compared using the Spearman's rank correlation coefficient [25]. Hypothesis testing was also applied in order to reach safer conclusions. A strong positive correlation was observed between the TD and the security level of software products, which provides evidence for the ability of TD to indicate security risks in software products.

Since a strong relationship has been found, an increase in the TD of a software product could indicate a similar increase in software vulnerabilities and vice versa. Furthermore, the strong interrelationship between TD and software security indicators, as demonstrated in this work, justifies the need for further elaboration that will finally lead to the design and development of a decision-support tool that will assist software developers to perform code optimizations taking into account critical TD and security aspects. In this study, analysis is conducted at the application level; however fine-grained recommendations at a lower level of granularity would improve the application in a real use case scenarios. For example, a possibility to perform analysis (i.e., assess the relationship between TD and security indicators) at the class, function or code block level would improve the usability of the envisioned tool. Another interesting challenge would be to consider various TD and security indicators.

The results of the present study provide preliminary evidence for the ability of TD to be used as an indicator of software security. However, in order to reach safer conclusions and exclude false positive correlation, a more elaborate empirical study is required. In particular, in our future research endeavors we are planning to replicate the present study by using a significantly larger code repository. Subsequently, if a relationship between TD and software security is still

observed, we will attempt to use TD as the basis for the construction of models able to highlight security risks in software products (e.g. vulnerability prediction models).

## ACKNOWLEDGMENT

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

## REFERENCES

- [1] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Prof., 2006.
- [2] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Static analysis-based approaches for secure software development," in *Security in Computer and Information Sciences*. Cham: Springer International Publishing, 2018, pp. 142–157.
- [3] M. Siavvas and E. Gelenbe, "Optimum checkpoints for programs with loops," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019.
- [4] Y. Shin and L. Williams, "Is Complexity Really the Enemy of Software Security?," *Proc. the 4th ACM Workshop on Quality of Protection, Alexandria, Virginia, USA, Oct., 2008*.
- [5] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in *ESEM'08: Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008.
- [6] M. Siavvas, D. Kehagias, and D. Tzovaras, "A preliminary study on the relationship among software metrics and specific vulnerability types," *International Conference on Computational Science and Computational Intelligence (CSCI) 2017*, pp. 916–921, 2017.
- [7] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," *Proceedings of the 14th ACM conference on Computer and communications security CCS 07*, p. 529, 2007.
- [8] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing Software Security Fortification through Code-Level Metrics," *Proceedings of the 4th ACM workshop on Quality of Protection*, pp. 31–38, 2008.
- [9] M. Siavvas, M. Jankovic, D. Kehagias, and D. Tzovaras, "Is Popularity an Indicator of Software Security?" in *2018 IEEE 9th International Conference on Intelligent Systems (IS)*, 2018.
- [10] W. Cunningham, "The wycash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.
- [11] R. Scandariato, J. Walden, A. Hovsepian, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.
- [12] I. Chowdhury and M. Zulkernine, "Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?" in *Proc. of the 2010 ACM Symposium on Applied Comp.*, 2010.
- [13] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.
- [14] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, pp. 294–313, 2011.
- [15] S. Moshtari and A. Sami, "Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction," *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, pp. 1415–1421, 2016. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2851613.2851777>
- [16] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," *Proceedings - International Symposium on Software Reliability Engineering*, 2014.
- [17] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, "Examining the relationship between firm's financial records and security vulnerabilities," *International Journal of Information Management*, 2016.
- [18] J. Walden, M. Doyle, G. A. Welch, and M. Whelan, "Security of open source web applications," *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009.
- [19] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers and Security*, vol. 26, no. 3, 2007.

- [20] J. Walden and M. Doyle, "SAVI: Static-Analysis vulnerability indicator," *IEEE Security and Privacy*, vol. 10, no. 3, pp. 32–39, 2012.
- [21] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM SIGPLAN Notices*, 2004.
- [22] D. Tsoukalas, M. Siavvas, M. Jankovic, D. Kehagias, A. Chatzigeorgiou, and D. Tzouvaras, "Methods and Tools for TD Estimation and Forecasting: A State-of-the-art Survey," in *2018 IEEE 9th International Conference on Intelligent Systems (IS)*, 2018.
- [23] J.-L. Letouzey and M. Ilkiewicz, "Managing technical debt with the sqale method," *IEEE software*, vol. 29, no. 6, pp. 44–51, 2012.
- [24] G. Digkas, M. Lungu, A. Chatzigeorgiou, and P. Avgeriou, "The evolution of technical debt in the apache ecosystem," in *Software Architecture*, A. Lopes and R. de Lemos, Eds. Cham: Springer International Publishing, 2017, pp. 51–66.
- [25] C. Spearman, "The proof and measurement of association between two things. By C. Spearman, 1904." *The American journal of psychology*, vol. 100, no. 3-4, pp. 441–471, 1987.
- [26] J. Cohen, *Statistical power analysis for the behavioral sciences*. Hillsdale, N.J: L. Erlbaum Associates, 1988.
- [27] M. Siavvas, K. Chatzidimitriou, and A. Symeonidis, "QATCH - An adaptive framework for software product quality assessment," *Expert Systems with Applications*, 2017.