# On the capacity of Technical Debt to indicate security issues in open-source software products

George David Apostolidis[1], Miltiadis Siavvas[1], Ilias Kalouptsoglou[1], Dimitrios Tsoukalas[1], Alexander Chatzigeorgiou[2], Apostolos Ampatzoglou[2], Dionysios Kehagias[1], and Dimitrios Tzovaras[1]

[1] Information Technologies Institute / Centre for Research and Technology Hellas, Thessaloniki, Greece
{geoapos, siavvasm, iliaskaloup, tsoukj, diok, Dimitrios.Tzovaras}@iti.gr
[2] University of Macedonia, Thessaloniki, Greece
{achat, a.ampatzoglou}@uom.edu.gr

**Abstract.** Technical Debt (TD) arises from insufficient design and poor coding practices, affecting important software quality attributes. As systems evolve, accumulated TD degrades code structure and increases maintenance complexity. However, poor code quality can potentially lead to the introduction of additional issues, including vulnerabilities. In this study, we examine the relationship between TD and software security across a broad sample of open-source Java projects, using a combination of static code analysis and statistical methods both at project- and at class- level of granularity. Two standalone TD and security assessment tools are employed to evaluate the selected source code artifacts regarding both their quality, measured in terms of TD, and their security level, expressed through a quantitative security score (i.e., the Security Index) and the number of the detected potential security issues. Statistical tests are then performed in order to to examine the existence of potential interrelation. The results of the analysis show that higher TD is associated with weaker security. Specifically, at project level, the high TD density and the Security Index of the studied projects were found to be negatively correlated, while at class level, high TD probability demonstrates a clear positive correlation with the number of security issues identified in each class, especially in specific vulnerability categories.

**Keywords:** Software Security · Software Quality · Technical Debt · Static Code Analysis · Correlation Analysis

## 1 Introduction

Technical Debt (TD) is an inevitable aspect of modern software development, often resulting from trade-offs between rapid delivery and code quality [1]. While TD is traditionally associated with maintainability and long-term development costs [2], it could potentially increase defect rates [3]. As systems grow in complexity and age, unresolved TD can degrade software structures potentially leading to the introduction of security risks, as well [4].

Open-source projects, which form a vital part of today's software infrastructure, are particularly susceptible to the accumulation of TD [5]. The collaborative nature of open-source development, combined with the pressure for rapid feature delivery, often leads to short-term design compromises [5]. Over time, these compromises not only undermine code maintainability, but may also weaken the security posture of software systems [6].

Security is an important aspect of the Software Development Life-Cycle (SDLC) and requires special attention [7–9]. Recently, various research endeavors have attempted to link TD to security, either theoretically, such as through the concept of Security Debt [10], or empirically by conducting statical analyses [11]. TD has been also used as an indicator for constructing Machine Learning (ML) models to predict internal security risk, showing promising results [5].

Despite the recent endeavors and the positive results, there is still a lot of work to be conducted to increase the confidence in these observations about the connection between TD and security. More specifically, current literature lacks a more fine-grained analysis of their interconnection More specifically, their interconnection has not been fully explored both at project- and at class-level of granularity [12], while a comprehensive large-scale empirical evaluation and statistical analysis of this hypothesis is missing, as well. In addition, it remains unclear how TD correlates with specific security weaknesses (e.g., null pointer, resource handling, etc.)

To this end, in the present work we empirically examine the relationship between TD and software security based on a dataset of popular and widely-used open-source projects. In particular, we chose the most starred projects from robust organizations on GitHub (e.g., Apache), managing to retrieve 414 projects, comprising 783,486 classes, which were then statically analyzed with two individual tools for evaluating in a quantitative manner their TD and security level. Then, statistical analysis was conducted, in both project and class levels of granularity, to explore correlations between TD indicators and software security, leading to some interesting observations.

The rest of the paper is structured as follows. Section 2 presents the previous work on the domain. Section 3 describes in detail the methodology of our research, while Section 4 presents our results. Section 5 discloses threats to validity, and finally, in Section 6, we summarize the conclusions of the analysis and provide suggestions for future work.

## 2 Related Work

The relationship between TD and software security has recently gained attention in the research community [4, 5, 13]. Rindell et al.[10] introduce the notion of Security TD (SecTD), arguing that unaddressed vulnerabilities act as liabilities that accrue interest in the form of increasing risk and maintenance costs. Izurieta et al.[13, 14] extend this by linking SecTD to software artifacts (e.g., architecture, code) and proposing remediation strategies compatible with agile and DevSecOps practices. These studies are theoretical, aiming to transfer TD

knowledge to software security in a compelling manner. Although their analysis and mathematical modeling are strong, they lack empirical evaluation and evidence for a practical TD-security relationship. Therefore, another line of work focuses on empirically examining whether TD metrics can serve as indicators of security risk in software.

Towards this direction, several approaches attempted to utilize software metrics (which are known TD indicators) as indicators of the existence of vulnerabilities in software artifacts [7]. Early efforts to link software quality metrics with security vulnerabilities focused on Complexity, Cohesion, and Coupling (CCC) metrics. Initially, Shin and Williams [11] as well as Chowdhury et al. [15] demonstrated that CCC could serve as early indicators of the existence of vulnerabilities in software systems. Subsequent studies [16, 17] leveraged statically extracted software metrics (e.g., Fan-in, Fan-out, Nesting Complexity, etc.) to build ML models capable of classifying PHP files to vulnerable or non-vulnerable.

Building on this trend, later works examined concrete TD metrics, like code smells, code duplication, and SQALE Index, with respect to their connection to software security. Notably, Siavvas et al.[3] analyzed 50 OSS projects and found a strong positive correlation between TD, measured at project-level in terms of SQALE Density [18], and security, measured as Static Analysis Vulnerability Density (SAVD). Extending this line, Siavvas et al.[5], in a follow-up study, showed that TD metrics can be used as inputs for building predictive models able to predict internal security risks. Gigante et al. [19] performed a user study to test whether quality-oriented refactoring guided by SonarQube [20] reduces flaws reported by Fortify [21]; although some overlap emerged, they concluded that security remediation demands changes beyond general code-quality fixes.

Although several research efforts have explored the relationship between TD and software security, either theoretically or empirically, further investigation is needed to draw safer conclusions. This study explores this relationship in depth using a large dataset of Java projects containing TD and security data at both project- and class-level granularity. Instead of building ML models to predict security issues, we examine correlations among TD Density, software quality metrics, and a quantitative security score (Security Index). We also study how the likelihood of high-TD classes relates to the number of security issues and affects specific vulnerability types.

## 3 Research Methodology

In this section, we describe the methodology that we followed for examining the relationship between TD and Software Security in open-source projects. Our methodology is based on Static Code Analysis (SCA) and statistical analysis. The entire process is illustrated in Figure 1.

As depicted in Figure 1, we gather open-source repositories, and, after applying a filtering process, we analyze them through different SCA tools. Specifically, we have implemented a RESTful API that invokes two standalone tools, namely the Security Assessment Model (SAM) [22] and Technical Debt (TD) Classifier

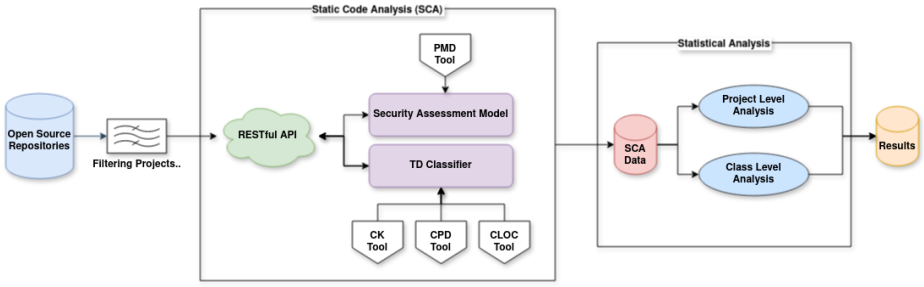


**Fig. 1.** Overview of the research methodology of the study

[23]. These tools rely on other SCA tools to perform their analysis. In particular, SAM uses the PMD analyzer [24], while TD Classifier employs CK [25], PMD's Copy/Paste Detector (CPD) [26], and CLOC [27] tools and aggregates their results. In this way, a dataset of software projects along with information about their security issues, security level, and TD is collected. Subsequently, we perform a statistical analysis, both at project- and at class-level, and thus, we evaluate the relationship between software quality and software security. In the following sub-sections, more details are provided for each of the SAM, TD Classifier, data collection and statistical analysis steps.

### 3.1 Security Assessment Model

The Security Assessment Model (SAM) [22, 28] was developed as an hierarchical model that leverages low-level indicators to assess the internal security level of software systems. The process starts with extracting low-level indicators (i.e., security-related alerts) from the source code. The model then maps these findings to security properties (e.g. Resource Handling) and property scores are assigned using thresholds that have been empirically determined. These property-level scores are then combined to assess higher-level security attributes, the CIA triad (i.e., Confidentiality, Integrity, and Availability). The Security Index (SI), a unified security score that ranges from 0 to 1 and represents the overall security of the examined software, is then created by the weighted average of these values. The internal structure of the proposed model is shown in Figure 2.

As can be seen by Figure 2 the model has a hierarchical structure. Four hierarchical layers make up the model's general architecture: the measures, properties, characteristics, and overall security layer. The model starts with the low-level measures computed via SCA and through several steps of aggregation, they compute the security score of the broader system, i.e., its *Security Index*.

The model specifies a set of 8 unique security properties that correspond to specific vulnerability categories, such as Null Pointer, Resource Handling, Exception Handling, etc., each of which has a code-level measure that directly determines it. The code-level measures that are used to measure those properties are actually the densities of SCA alerts that belong to these properties, i.e.,

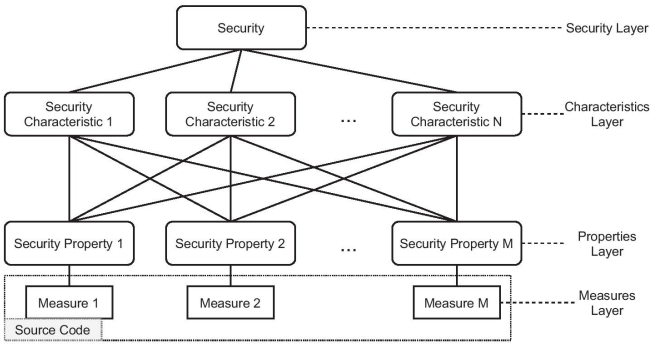


**Fig. 2.** Overview of SAM

vulnerability categories. More specifically, the model employs the Static Analysis Vulnerability Density (SAVD), which quantifies security issues in relation to software size. We have to note that the computation of SAVD includes division of the number of security-related SCA alerts with the total lines of code (LOC) of the analyzed software project to standardize our security metrics and render them independent of project size, which is a common approach that is adopted in the literature. Considering $\text{SAVD}_i$ as the static analysis vulnerability density for the $i$-th vulnerability category, $N_i$ the number of SCA alerts in category $i$, and LOC the total lines of code in the software product, SAVD is computed as:

$$\text{SAVD}_i = \frac{1000 \times N_i}{\text{LOC}}$$

Hence, for the present analysis we focus on three different measures that are produced by SAM: (i) the total Security Index of the analyzed project, (ii) the total number of security issues and densities that a project or a class contains, and (iii) the densities (SAVDs) of the various properties (i.e., specific vulnerability types). The first two measures facilitate project- and class-level analysis, whereas the third measure enables the conduction of more fine-grained analysis taking into consideration the type of the identified security issues.

### 3.2 Technical Debt Classifier

The Technical Debt Classifier (TD Classifier) was developed by Tsoukalas et al. [23] to automatically identify software classes with high levels of TD. Its foundation was established in an earlier empirical study [29], where the same authors systematically evaluated statistical and ML algorithms for their effectiveness in classifying software classes as having either High or Not-High TD.

To construct the ground truth, Tsoukalas et al. [29] utilized a dataset created by Amanatidis et al. [30], which investigated the consistency of TD assessments across three leading TD management tools (i.e., SonarQube, CAST, and Squore) on 25 Java open-source projects. This study aimed to evaluate the degree of

agreement (or diversity) among these tools and identify profiles of files sharing similar levels of TD (e.g., that of high TD levels in all employed tools) through archetypal analysis. In this context, a class is labeled as high-TD only if all three tools consistently identified it as having high TD, i.e., an archetype referred to as the Max-Ruler profile. This labeling strategy was adopted to ensure a high-confidence and practically useful definition of high TD, avoiding the inclusion of files identified as high-TD by only one (Rebel profile) or two (Partner profile) tools, which could introduce uncertainty or result in an overwhelming number of flagged files. Therefore, a dataset of 18,857 classes was compiled, with 1,283 classes identified as high-TD, establishing in that way the "ground truth".

Furthermore, to enrich the dataset, Tsoukalas et al. [29] incorporated code metrics from several SCA tools. Structural metrics (e.g., complexity and coupling) were extracted using the CK tool [25], code duplication was measured via the CPD tool [26], and size-related metrics (e.g., lines of code, with and without comments) were gathered using the CLOC tool [27]. Subsequently, several preprocessing steps were applied, including outlier detection, missing values handling, oversampling, and a statistical analysis to discriminate metrics with predictive power. Multiple ML models were evaluated through cross-validation to identify a robust classifier for detecting high-TD classes.

In the context of this study, the TD Classifier plays an important role, allowing for the identification of software classes with high levels of TD. By invoking this tool, we managed to retrieve for each analyzed project the High Technical Debt Density (HTD Density), which represents the percentage of classes with high TD in the analyzed project. Furthermore, we retrieved the High TD Probability (HTD Probability), which represents the probability for a class of the analyzed project to have high TD. HTD Probability is essentially derived directly from the output of the classifier and represents the estimated likelihood that a given class falls into the high-TD category. Rather than making a binary decision (i.e., high-TD vs. not high-TD), the classifier outputs a continuous score in the range [0,1], which expresses its confidence in the high-TD classification.

### 3.3 Data Collection

By running SAM and TD Classifier we collected quality and security measurements from various open-source repositories written in Java and we applied a filtering step by ranking the projects based on their number of stars. Moreover, to ensure that the selected software projects are actively developed, we included the additional requirement that the projects should have been updated in the last year. We also considered the filtering criterion that the selected projects are maintained for an extended period, potentially spanning years. To automate the analysis of the selected projects with the TD Classifier and SAM tools, we implemented an additional tool that triggers the analyses and gathers those results that are necessary for the present study.

Through this tooling, the study generated one dataset for project-level and one for class-level analysis. The former contains the results that were retrieved from TD Classifier and SAM at project-level of granularity. Overall, we retrieved

the 500 most popular Java projects based on the number of stars. From these projects, 414 were able to be analyzed by the two tools without errors or warnings. We decided to exclude those projects that their analysis contained errors or warnings, in order to strengthen the reliability of our evaluation results.

Moreover, the class-level dataset contains the results of these tools for the individual classes of the analyzed projects. In particular, it contains 783,486 Java classes, providing detailed insights into metrics, such as High TD Probability (HTD Probability) and Total Security Issues in each class. Fragments of the project-level and class-level datasets are depicted in Table 1 and Table 2 respectively. For instance, we can see the values of Total Lines of Code (LOC), HTD Density, SI, Misused Functionality Issues, Null Pointer Issues, Coupling Between Objects (CBO), Weighted Methods per Class (WMC), and HTD Probability.

**Table 1.** An example of key metrics for indicative Apache projects

| **Project** | **Total LOC** | **HTD Dens** | **Sec. Index** | **Misused Func.** | **Null Ptr** | **CBO** | **WMC** |
|---|---|---|---|---|---|---|---|
| dubbo | 372,350 | 0.0651 | 0.3013 | 0.0199 | 0.0013 | 12 | 40 |
| kafka | 942,167 | 0.1372 | 0.4012 | 0.0218 | 0.0010 | 33 | 68 |
| incubator-seata | 264,887 | 0.0569 | 0.5903 | 0.0080 | 0.0004 | 232 | 16,674 |
| flink | 2,268,660 | 0.1015 | 0.3527 | 0.0204 | 0.0018 | 23 | 45 |
| skywalking | 153,115 | 0.0400 | 0.4172 | 0.0442 | 0.0006 | 14 | 15 |

**Table 2.** Key metrics for indicative class files from different projects

| **File** | **LOC** | **HTD Proba** | **Total Issues** | **M. Func. Issues** | **NullPtr Issues** | **CBO** | **WMC** |
|---|---|---|---|---|---|---|---|
| MockEngine.java | 213 | 0.01 | 20 | 6 | 0 | 18 | 32 |
| ListenerMap.java | 319 | 0.51 | 44 | 4 | 0 | 8 | 28 |
| MockRequestCycle.java | 178 | 0.01 | 27 | 4 | 1 | 10 | 28 |
| TestUrlValidator.java | 120 | 0.02 | 0 | 0 | 0 | 3 | 11 |
| TreeNodeView.java | 399 | 0.24 | 83 | 9 | 5 | 22 | 58 |

### 3.4 Statistical Analysis

Subsequently, the study conducts a statistical analysis to evaluate the relationship between TD and software security at project- and class- level of granularity.

**Project-Level Statistical Analysis:** More specifically, at project level, correlation analysis was employed in order to determine the interconnection between

HTD Density and SI. To determine the statistical test that we need to employ, normality tests were conducted to examine the distributional characteristics of the examined variables. For this purpose, we employed the Shapiro-Wilk test [31]. Specifically, HTD Density yielded a test statistic of 0.7784 (p-value $< 0.05$), and SI a test statistic of 0.9521 (p-value $< 0.05$). Considering as the null hypothesis that the distributions are normal, we can reject the null hypothesis for both variables, since both p-values fall below the significance threshold of 0.05.

Hence, the Spearman rank correlation [32] was employed for the subsequent correlation analysis. As a non-parametric method, it does not rely on assumptions of normality and is well-suited for detecting monotonic relationships, in contrast to Pearson correlation that requires both variable to be normally distributed. In particular, Spearman analysis [32] was utilized to determine the correlation between TD and software security. To check the statistical significance of the computed correlation, we defined the following null hypothesis:

> $H_0$: No statistically significant correlation is observed between TD and software security in project-level of granularity.

Then we tested the null hypothesis in the 95% confidence interval. More specifically, we repeated this analysis to investigate the correlation among the following indicators: (i) HTD Density and SI, (ii) HTD Density and security metrics, (iii) quality metrics and SI, and (iv) quality metrics and security metrics.

At this point, we have to clarify that, from now on, the term security metrics will refer to the densities of specific security issues. In other words, the density of a security issue is defined as the number of statically identified issues in a particular security category (e.g., null pointer) divided by the total lines of code in the analyzed project. These densities serve as indicators of the presence of specific types of vulnerabilities in the project [3, 33].

**Class-Level Statistical Analysis:** Similarly, at the class-level analysis, we employed correlation analysis to determine the relationship among TD-related and software security-related metrics in class-level of granularity. First, we conducted the Shapiro-Wilk test [31] to check whether the compared variables follow a normal distribution. The statistical test returned a test statistic of 0.545 (p-value $<$ 0.05) for HTD Probability, which is the probability for a class to have high TD, and a test statistic of 0.071 (p-value $< 0.05$) for the Total Security Issues, which is the total number of statically identified security issues in the analyzed class. All computed p-values are by far lower than the 0.05 significance threshold, and therefore, we can reject the null hypothesis that the variables follow a normal distribution. Thus, we proceeded by using the Spearman rank correlation [32].

Specifically, we investigated potential correlation among the following indicators at the class level: (i) HTD Probability and Total Security Issues, (ii) HTD Probability and security metrics, (iii) quality metrics and Total Security Issues, and (iv) quality metrics and security metrics. To check the statistical significance of the computed correlations, the following hypothesis was tested in the 95% confidence interval based on the p-values of the Spearman correlation:

$H_0$: No statistically significant correlation is observed between TD and software security in class-level of granularity.

**Strength of the Correlations:** In both cases of project-level and class-level analysis, we used the guidelines provided by Cohen et al. [34] to judge the strength of the observed correlations. Cohen et al. rules suggest that a correlation among 0.1 and 0.3 is considered weak, between 0.3 and 0.5 is moderate, and above 0.5 is strong. Even a weak correlation is typically considered sufficient in the literature as evidence that a factor may have the potential to indicate security risks or the presence of vulnerabilities [11, 15, 35–37].

## 4 Results

In this section, we present the results of our study about the potential interconnection between TD and software security. While the primary focus of this study is to explore the relationship between TD and software security, we present also a correlation analysis among various software and security metrics as a complementary perspective.

### 4.1 Results of the Project Level Analysis

Figure 3 presents the results of the Spearman correlation between several quality metrics (including HTD Density) and SI at the project-level analysis. We can see that projects with HTD Density exhibit a weak negative correlation ($\rho = -0.277$), which is considered significant (p-value $< 0.05$). Therefore, we can state that the more classes with high TD the lower the security of the software projects, at least for the studied dataset.

Furthermore, Figure 3 shows that SI demonstrates a weak negative correlation with all the examined quality metrics, with the most notable examples being the following ones: Coupling Between Objects (CBO), Fan-out, Duplicated lines, Fan-in, WMC, Response for class (RFC), and the size of the project expressed through the total of its lines. Elevated values in these metrics, typically indicative of higher complexity and tighter coupling, are associated with poorer security performance. This finding is in line with prior research endeavors, which also observed that there is a weak correlation between quality-related metrics and security [5, 11, 15, 35].

Furthermore, Figure 4 provides an overview of how HTD Density correlates with various security-related metrics. We can see that, based on Cohen et al. [34], there is a moderate positive Spearman correlation between HTD Density and Null Pointer Density. There is also a weak positive correlation between HTD Density and the Resource Handling Density. Conversely, HTD Density demonstrates a negative correlation with the Misused Functionality Density, no correlation with the Logging Density, and very weak correlation with the Densities of Synchronization, Assignment, and Exception Handling. Therefore, we can argue that there is a relationship between HTD Density and the existence

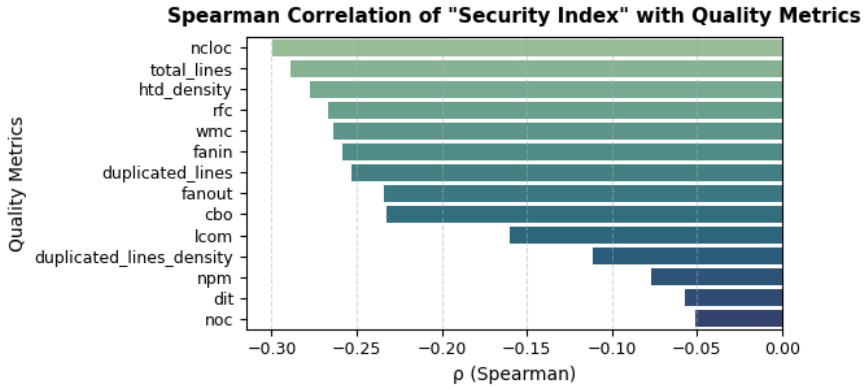


**Fig. 3.** Correlation of Security Index with Quality Metrics

of security issues of specific security categories. In particular, these findings show that high TD is more indicative of the Null Pointer and Resource Handling vulnerabilities and less so for the others.

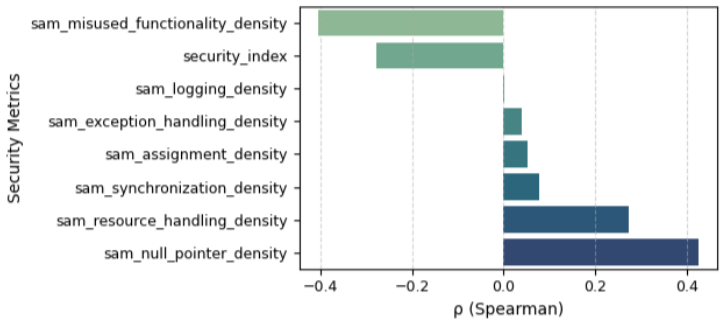


**Fig. 4.** Correlation of HTD Density with Security Metrics

In addition, we examined the correlation among various quality-related metrics with security-related metrics. Table 3 presents the computed correlations using Spearman analysis at project-level. Through this analysis, we made some interesting observations about the potential interconnection of specific quality metrics with specific security metrics and thus to specific vulnerability types.

As can be seen on Table 3, several quality metrics are correlated with security ones, some of which positively and some negatively. In particular, we can observe moderate positive correlation between quality metrics, such as WMC, Fan-in, Non-Commented Lines of Code (NCLOC), total lines, Lack of Cohesion of Methods (LCOM), and Depth of Inheritance Tree (DIT) with Null Pointer Density. Considering both Figure 4 and Table 3, it seems that Null Pointer issues

**Table 3.** Spearman correlation between quality and security metrics at the project level. Significant correlations (p-value $< 0.05$) are marked with *. Text color denotes strength: black = very weak, blue = weak, green = moderate.

| Metrics | res_hand | assign | except | misused | sync | null | log |
|---|---|---|---|---|---|---|---|
| wmc | 0.25* | 0.05 | -0.02* | -0.39* | 0.06 | 0.41* | -0.03 |
| dit | -0.01 | -0.06 | -0.06 | -0.32 | -0.09 | 0.29* | -0.12 |
| rfc | 0.12* | 0.04 | 0.05 | -0.26 | 0.17 | 0.19 | 0.04 |
| ncloc | 0.16* | -0.00 | 0.09 | -0.28* | 0.11* | 0.35* | -0.06 |
| total_lines | 0.15* | -0.01 | 0.09 | -0.26* | 0.11* | 0.34* | -0.06 |
| Fan-in | 0.13* | 0.01 | -0.03 | -0.33* | 0.11* | 0.40* | 0.02 |
| fan-out | 0.00 | 0.04 | 0.04 | -0.15* | 0.13* | 0.12* | -0.02 |
| noc | -0.00 | -0.02 | 0.03 | -0.15* | -0.08 | 0.11* | -0.02 |
| cbo | 0.00 | 0.04 | 0.04 | -0.15* | 0.13* | 0.12* | 0.02 |
| lcom | 0.13* | -0.04 | -0.09 | -0.46* | -0.01 | 0.34* | -0.11* |

are the most related ones with the high values of TD and high values of quality metrics in general. In contrast, there is a negative correlation of several quality metrics and Misused Functionality Density, similarly with the observation about the correlation of HTD Density and Misused Functionality Density (Figure 4).

### 4.2 Results of the Class Level Analysis

At this section, we aim at identifying potential correlations among software quality and software security indicators at class-level of granularity. For this purpose, we focus on the HTD Probability retrieved from the TD Classifier and the Total Security Issues identified by SAM at class level, along with various quality and security-related metrics retrieved through SCA.

Figure 5 illustrates the Spearman correlation between Total Security Issues and various software quality metrics. The plot reveals that, apart from Depth of Inheritance Tree (DIT), all the examined quality metrics are correlated positively with the Total Security Issues in a class. Nevertheless, the strongest positive correlations are observed with WMC, Fan-in, and Max Nested Blocks, each exceeding a coefficient of 0.45. This suggests that classes with greater complexity, deeper nesting, greater method complexity, and increased coupling, are more prone to security-related issues.

Furthermore, Figure 6 presents the Spearman correlation between High Technical Debt Probability (HTD Probability) and a set of security-related metrics. We can see that all the examined security metrics are positively correlated with HTD Probability. Nonetheless, HTD Probability demonstrates stronger correlation with Assignment and Resource Handling issues, all exceeding $\rho = 0.30$ (i.e., moderate correlation). Moderate correlation exists also in relation to the total security issues in a class. Therefore, we can argue that classes which are more probable to have high TD are also more prone to vulnerabilities, and especially to vulnerabilities of Assignment and Resource Handling types.

In addition, Table 4 presents the Spearman correlation analysis on class-level among the examined class-level quality and security metrics. In Table 4, `res_hand`, `assign`, `except`, `misused`, `sync`, `null`, and `log` columns stand

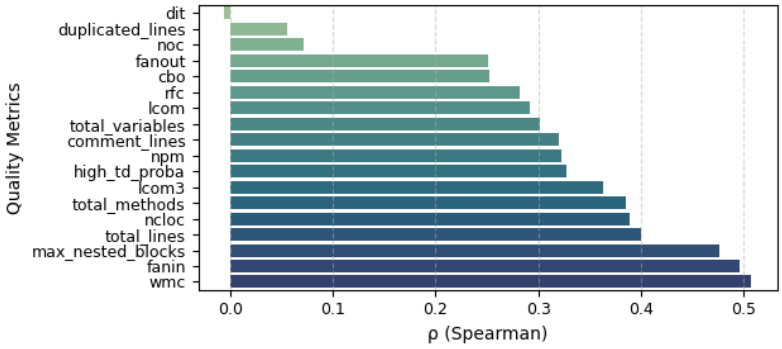


**Fig. 5.** Correlation of Total Security Issues with Quality Metrics

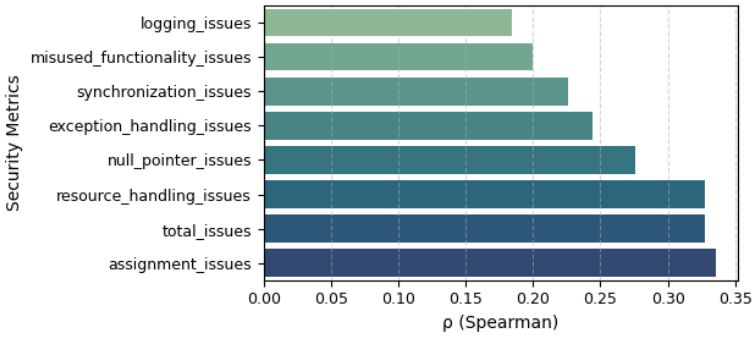


**Fig. 6.** Correlation of HTD Probability with Security Metrics

for the densities of Resource Handling, Assignment, Exception Handling, Misused Functionality, Synchronization, Null Pointer, and Logging issues, respectively. Concisely, one can observe the significant correlation of software metrics, such as WMC, CBO, RFC, LCOM, Max Nested Blocks, Fan-in, Fan-out, Non-commented Lines of Code (NCLOC), total methods, total variables, and Total Lines of Code with the densities of the different security-related metrics, with some metrics more and some less.

These observations align with prior research showing weak yet statistically significant correlations between software quality and security metrics. Hence, our findings further support the common observation that software metrics can be viewed as weak, yet statistically significant indicators of security issues.

Most notably, there is a moderate positive correlation of WMC with Resource Handling and Assignment densities, RFC with Resource Handling, Max Nested Blocks with Resource Handling, Assignment, and Exception Handling, total variables with Resource Handling, Fan-in with Assignment, NCLOC with Resource Handling, as well as Total Lines of Code with Resource Handling.

Therefore, it is clear that there is a relationship among most of software metrics and the densities mainly of Resource Handling and Assignment issues.

### 4.3 Threshold-based Analysis

Our analysis also revealed some deeper observations regarding the correlation between TD and software security under specific conditions (i.e., below/above specific thresholds). In particular, we calculated the median value of each examined metric after removing outliers, and then, we divided the dataset into two categories; one for values below the median and one for values above it.

Based on the threshold-based analysis, we created two groups of projects, namely the "low HTD Density" for those with HTD Density $< 0.06$, and "high HTD Density" for those with HTD Density $> 0.06$. We noticed in "low HTD Density" the correlation between HTD Density and SI is moderately negative $\rho = -0.336$ (p-value $< 0.05$). In contrast, for projects with HTD Density $\geq 0.06$, the correlation weakened substantially $\rho = -0.064$ (p-value $< 0.05$), possibly due to a saturation effect, where, at higher levels of TD Density, other structural or contextual factors begin to play a more significant role.

Furthermore, in our class-level analysis, we observed that the relationship between total (security) issues and HTD Probability varies depending on class-level thresholds. Specifically, in classes exceeding the median value of 5 total security issues, the relationship becomes substantially stronger, with a positive Spearman correlation of $\rho = 0.753$ (p-value $< 0.05$) in comparison with classes containing less than 5 issues $\rho = -0.178$ (p-value $< 0.05$).

## 5 Threats to validity

This section provides a remark on threats to the validity of our study. There is a construct validity related to the use of SCA tools that may focus only on issues detectable in code, missing dynamic or logic-based vulnerabilities. Threats to internal validity concern the exclusion of 17% of projects due to SCA errors or warnings, since it could introduce selection bias. However, it was necessary to ensure the reliability of our results, and therefore, we excluded projects that could not be analyzed. Moreover, to enhance the reliability of our benchmark and minimize the risk of using unrepresentative and low-quality projects, we focused on popular and actively maintained ones.

Threats to external validity relate to the fact that our analysis is limited to Java open-source projects retrieved from GitHub and therefore the results may not generalize to other languages. However, Java is used as a proof of concept and is a representative language of open-source object oriented programs. Finally, relying only on SAM and TD Classifier tools, which are external tools, not only limits the generalizability of our findings, but also constraints us to their limitations and weaknesses. However, the selection of tools that leverage cutting-edge techniques in the field of TD and security assessment and have been published in high-rank venues minimizes that risk.

**Table 4.** Spearman correlation between quality metrics and security issues at the class-level. Significant correlations (p-value $< 0.05$) marked with *, text color indicates strength: very weak, weak, moderate.

| Metric | res_hand | assign | except | misused | sync | null | log |
|---|---|---|---|---|---|---|---|
| cbo | 0.19* | 0.15* | 0.20* | -0.04* | 0.15* | 0.16* | 0.14* |
| wmc | 0.33* | 0.35* | 0.24* | 0.05* | 0.24* | 0.29* | 0.19* |
| dit | -0.01* | 0.00 | -0.01* | -0.04* | -0.04* | 0.01* | -0.06* |
| rfc | 0.31* | 0.16* | 0.26* | -0.11* | 0.22* | 0.21* | 0.22* |
| lcom | 0.21* | 0.22* | 0.16* | 0.13* | 0.20* | 0.20* | 0.14* |
| npm | 0.15* | 0.21* | 0.09* | 0.05* | 0.13* | 0.18* | 0.05* |
| total_methods | 0.22* | 0.24* | 0.14* | 0.02* | 0.18* | 0.23* | 0.09* |
| max_nested_blocks | 0.38* | 0.36* | 0.33* | 0.09* | 0.26* | 0.28* | 0.27* |
| total_variables | 0.31* | 0.17* | 0.22* | -0.04* | 0.21* | 0.22* | 0.20* |
| fan-in | 0.21* | 0.40* | 0.16* | 0.23* | 0.19* | 0.21* | 0.11* |
| fan-out | 0.19* | 0.15* | 0.20* | -0.04* | 0.14* | 0.16* | 0.14* |
| noc | 0.03* | 0.04* | 0.02* | -0.04* | 0.04* | 0.04* | 0.02* |
| duplicated_lines | 0.09* | 0.01* | 0.05* | -0.09* | 0.03* | 0.08* | 0.04* |
| ncloc | 0.31* | 0.20* | 0.24* | -0.05* | 0.23* | 0.26* | 0.19* |
| total_lines | 0.31* | 0.19* | 0.23* | -0.05* | 0.23* | 0.27* | 0.18* |

## 6 Conclusion and Future Work

In this study the primary purpose was to investigate the potential relationship between TD and software security. We used SCA-based methods to collect information about the software quality and security of open-source projects and then employed a statistical analysis. In brief, the findings showed a statistically significant negative correlation between HTD Density and Security Index in project-level analysis, indicating that projects with more classes with high TD exhibit lower security level. Moreover, we observed a significant positive correlation between HTD Probability and total number of security issues in class-level analysis, especially in case of classes with more than 5 total security issues.

These findings provide useful implications for practitioners. Project managers and developers could have an indication of how the quality compromises that they may make to meet strict production deadlines or to deliver more functionalities could affect the security level of their software product. In addition, they could have an indication of how prone their software is to important security issues, such as null pointer and resource handling issues, based on the level of TD that has been accumulated to their system over the years.

Suggestions for further research include the investigation of potential causal links among TD variables and software security and how certain quality-related characteristics impact security. We are currently conducting a broader and more thorough empirical study, and causal analysis will be a core element of the experimentation. Additional dynamic security testing tools will be also considered in future studies, in order to account for dynamic and logic-based vulnerabilities.

***Acknowledgements*** Work reported in this paper has received funding from the European Union's Horizon Europe Research and Innovation Program through the DOSS project, Grant Number 101120270.

## Appendix

Supplementary material related to this article can be found online:
https://sites.google.com/view/td-sec-correlation/

## References


1. Cunningham, W.: The wycash portfolio management system. In: OOPSLA Experience Reports. (1992)
2. Avgeriou, P., Bogdanov, C.A., Kruchten, M., Lago, P., Nord, R.: An overview and comparison of technical debt measurement tools. IEEE Software **38**(3) (2020)
3. Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D., Anicic, N., Gelenbe, E.: An empirical evaluation of the relationship between technical debt and software security. In: Empirical Analysis on Software Quality. Springer (2019) To appear.
4. Rindell, K., Holvitie, J.: Security risk assessment and management as technical debt. In: International Conference on Cyber Security and Protection of Digital Services (Cyber Security), IEEE (2019)
5. Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Tzovaras, D.: Technical debt as an indicator of software security risk: A machine learning approach for software development enterprises. Enterprise Information Systems **16**(5) (2022)
6. Gkortzis, A., Mitropoulos, D., Spinellis, D.: Vulinoss: a dataset of security vulnerabilities in open-source systems. In: Proceedings of the 15th International conference on mining software repositories. (2018) 18–21
7. Kalouptsoglou, I., Siavvas, M., Ampatzoglou, A., Kehagias, D., Chatzigeorgiou, A.: Software vulnerability prediction: A systematic mapping study. Information and Software Technology **164** (2023)
8. Kelli, V., Radoglou-Grammatikis, P., Lagkas, T., Markakis, E.K., Sarigiannidis, P.: Risk analysis of dnp3 attacks. In: 2022 IEEE International Conference on Cyber Security and Resilience (CSR). (2022) 351–356
9. Siganos, M., Radoglou-Grammatikis, P., Kotsiuba, I., Markakis, E., Moscholios, I., Goudos, S., Sarigiannidis, P.: Explainable ai-based intrusion detection in the internet of things. In: Int. Conf. on Availability, Reliability and Security. (2023)
10. Rindell, K., Holvitie, J., et al.: Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. In: the 14th International Conference on Availability, Reliability and Security (ARES), ACM (2019)
11. Shin, Y., Williams, L.: An empirical model to predict security vulnerabilities using code complexity metrics. In: Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM (2008)
12. Verendel, V.: Quantified security is a weak hypothesis: A critical survey. In: Proceedings of the New Security Paradigms Workshop (NSPW), ACM (2009)
13. Izurieta, C., Rice, D., Kimball, K., Valentien, T.: A position study to investigate technical debt associated with security weaknesses. In: Proceedings of the International Conference on Technical Debt, Gothenburg, Sweden, IEEE (2018)
14. Izurieta, C., Prouty, M.: Leveraging SecDevOps to Tackle the Technical Debt Associated with Cybersecurity Attack Tactics. In: Proceedings of the Second International Conference on Technical Debt (TechDebt), IEEE Press (2019)
15. Chowdhury, I., Zulkernine, M.: Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities? In: Proceedings of the ACM Symposium on Applied Computing (SAC), ACM (2010)

16. Walden, J., Stuckman, J., Scandariato, R.: Predicting vulnerable components: Software metrics vs text mining. In: IEEE 25th international symposium on software reliability engineering, IEEE (2014)
17. Kalouptsoglou, I., Siavvas, M., Tsoukalas, D., Kehagias, D.: Cross-project vulnerability prediction based on software metrics and deep learning. In: International Conference on Computational Science and Its Applications, Springer (2020)
18. Letouzey, J.L., Ilkiewicz, M.: Managing technical debt with the sqale method. IEEE software **29**(6) (2012) 44–51
19. Gigante, D., Pecorelli, F., Barletta, V.S., Janes, A., Lenarduzzi, V., Taibi, D., Baldassarre, M.T.: Resolving security issues via quality-oriented refactoring: A user study. In: International Conference on Technical Debt, IEEE (2023)
20. SonarSource. https://www.sonarsource.com/products/sonarqube/ (2025)
21. OpenText: Opentext static application security testing (fortify). https://www.opentext.com/products/static-application-security-testing (2025)
22. Siavvas, M., Kehagias, D., Tzovaras, D., Gelenbe, E.: A hierarchical model for quantifying software security based on static analysis alerts and software metrics. Software Quality Journal **29**(2) (2021)
23. Tsoukalas, D., Chatzigeorgiou, A., Ampatzoglou, A., Mittas, N., Kehagias, D.: Td classifier: Automatic identification of java classes with high technical debt. In: Proceedings of the International Conference on Technical Debt, IEEE (2022)
24. PMD: Pmd source code analyzer. https://pmd.github.io/ (2025)
25. Aniche, M.: Java code metrics calculator (ck). https://github.com/mauricioaniche/ck/ (2015)
26. PMD Open Source Project: Finding duplicated code with cpd (2025)
27. Danial, A.: Count Lines of Code cloc. https://github.com/AlDanial/cloc#quick
28. Siavvas, M.G., Chatzidimitriou, K.C., Symeonidis, A.L.: Qatch-an adaptive framework for software product quality assessment. Expert Systems with Applications **86** (2017) 350–366
29. Tsoukalas, D., Mittas, N., Chatzigeorgiou, A., Kehagias, D., Ampatzoglou, A., Amanatidis, T., Angelis, L.: Machine learning for technical debt identification. IEEE Transactions on Software Engineering **48**(12) (2021) 4892–4906
30. Amanatidis, T., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: Evaluating the agreement among technical debt measurement tools: Building an empirical benchmark. Empirical Software Engineering **25**(5) (2020)
31. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality. Biom. (1965)
32. In: Spearman Rank Correlation Coefficient. Springer New York (2008)
33. Walden, J., Doyle, M.: Savi: Static-analysis vulnerability indicator. IEEE Security & Privacy **10**(3) (2012)
34. Cohen, J.: Statistical power analysis for the behavioral sciences. Academic press (2013)
35. Shin, Y., Williams, L.: Is complexity really the enemy of software security? In: Proceedings of the 4th ACM Workshop on Quality of Protection (QoP), Alexandria, Virginia, USA, Association for Computing Machinery (2008)
36. Camilo, F., Meneely, A., Nagappan, M.: Do bugs foreshadow vulnerabilities? a study of the chromium project. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE (2015)
37. Moshtari, S., Sami, A.: Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction. In: Proceedings of the 31st annual ACM symposium on applied computing. (2016)