Exploring the Relation between Technical Debt Principal and Interest: An Empirical Approach

Areti Ampatzoglou[1], Nikolaos Mittas[2], Angeliki-Agathi Tsintzira[3], Apostolos Ampatzoglou[3], Elvira-Maria Arvanitou[3], Alexander Chatzigeorgiou[3], Paris Avgeriou[1], Lefteris Angelis[4]

[1] Department of Computer Science, Institute for Mathematics, Computer Science and AI, University of Groningen, Netherlands

[2] Department of Chemistry, International Hellenic University, Kavala, Greece

[3] Department of Applied Informatics, University of Macedonia, Greece

[4] School of Informatics, Aristotle University of Thessaloniki, Greece

areti.ampatzoglou@rug.nl; nmittas@chem.ihu.gr; angeliki.agathi.tsintzira@gmail.com; a.ampatzoglou@uom.edu.gr; earvanitoy@gmail.com; achat@uom.edu.gr; paris@cs.rug.nl; lef@csd.auth.gr;

Context: The cornerstones of technical debt (TD) are two concepts borrowed from economics: principal and interest. Although in economics the two terms are related, in TD there is no study on this direction so as to validate the strength of the metaphor.

Objective: We study the relation between Principal and Interest, and subsequently dig further into the 'ingredients' of each concept (since they are multi-faceted). In particular, we investigate if artifacts with similar levels of TD Principal exhibit a similar amount of TD Interest, and vice-versa.

Method: To achieve this goal, we performed an empirical study, analyzing the dataset using the Mantel test. Through the Mantel test, we examined the relation between TD Principal and Interest, and identified aspects that are able to denote proximity of artifacts, with respect to TD. Next, through Linear Mixed Effects (LME) modelling we studied the generalizability of the results.

Results: The results of the study suggest that TD Principal and Interest are related, in the sense that classes with similar levels of TD Principal tend to have similar levels of Interest. Additionally, we have reached the conclusion that aggregated measures of TD Principal or Interest are more capable of identifying proximate artifacts, compared to isolated metrics. Finally, we have provided empirical evidence on the fact that improving certain quality properties (e.g., size and coupling) should be prioritized while ranking refactoring opportunities in the sense that high values of these properties are in most of the cases related to artifacts with higher levels of TD Principal.

Conclusions: The findings shed light on the relations between the two concepts, and can be useful for both researchers and practitioners: the former can get a deeper understanding of the concepts, whereas the latter can use our findings to guide their TD management processes such as prioritization and repayment.

1. INTRODUCTION

Technical Debt (TD), originally introduced in 1992 [11], is a metaphor that represents the impact of shortcuts taken during development, usually to meet business goals, such as limited time or budget [19] on maintainability. The cornerstones of the TD metaphor are two terms borrowed from the concept of debt in finance: principal and interest. *TD Principal* is the effort required to eliminate inefficiencies in the current design or implementation of a software system [3]; typical examples of such inefficiencies are code and design smells. On the contrary, *TD Interest* is the additional development effort required to modify the software (adding new features or fixing bugs), due to the presence of such inefficiencies [3]. The assessment of principal and interest depends on the type of TD (e.g., code, design, testing TD). The scope of this work is limited to *TD on the source code*, which is the most studied type of TD in the literature [2], the most supported by tools [2], and one of the most important in industry [5]. For simplicity, in the rest of the paper when we refer to TD, we imply code TD. For assessing TD Principal and TD Interest:

- On the one hand, *principal of TD* is relatively straightforward to quantify: one needs to specify the relevant types of code inefficiencies (e.g. understandability issues, violations of coding practices) and subsequently identify them in code, usually through automated analysis tools. In most approaches, principal is subsequently quantified by summing up the estimated effort to fix each individual inefficiency. As an example, SonarQube [3], [21] calculates TD Principal as the sum of the time required to fix code smells, which map to different **aspects of *TD Principal*** (e.g., Coding Standards, Understandability)—for more details see Section 3.1.
- On the other hand, quantifying *TD Interest* is more difficult, since an accurate calculation would require comparing the current version of a system with a zero-TD version, with respect to their difference in maintenance effort [1]. Of course, such a debt-free version does not exist and would be unrealistic to create in a real-world

setting. In most cases, TD interest is quantified indirectly using proxies for the two core **aspects of *TD Interest*:** (a) maintainability (e.g. complexity, coupling, cohesion, size) reflecting the difficulty to make changes; and (b) actual maintenance effort, based on historical data (e.g. LOC modified per revision)– see Section 3.2.

In economics, TD *Interest* and TD *Principal* are related through the interest rate: interest is calculated as a percentage of a loan (principal), paid to the lender periodically for the privilege of using that money. However, in the technical debt literature, the term interest rate has not and cannot be defined: according to Schmid [35], it is not possible to relate principal directly to an interest rate for a given interest period. That is because the actual interest rate depends on the specific maintenance activities performed and these cannot be determined a priori. To the best of our knowledge, there is currently no approach to relate *TD Principal* and *Interest*. Clarifying the relation between principal and interest would further validate the strength of the TD metaphor in software development and maintenance. More importantly, this relation can be practically used in estimating TD indices, but also managing individual TD items. For example, if a certain aspect of TD Principal, such as a specific type of code smell, incurs more interest compared to others, then it should be ranked higher through: ***preventing*** the associated code smells, ***prioritizing*** the refactorings of those smells, and eventually ***repaying*** them with refactoring applications.

According to Eisenberg [12], Lockheed Martin is monitoring TD, by using an excel sheet, in which class names are colored based on their perceived levels of technical debt. Such an approach serves two purposes: (a) ranking of classes with respect to their levels of TD, and (b) grouping classes into groups of similar TD. In this paper, we build upon the rationale of such an approach for TD monitoring, by investigating if ***artifacts with similar levels of TD Principal have similar levels of TD Interest***. More specifically, we investigate if there is a relation: (*a*) between *TD Principal* and *TD Interest*; and (*b*) between the aspects of *TD Principal* and *TD Interest*. To achieve the aforementioned goals and by taking into consideration the multifaceted nature of the examined concepts, we employ the Mantel test [24]. The main benefit of using the Mantel test, compared to a traditional correlation analysis, is that it offers the opportunity to study the relationship of concepts that can be decomposed into aspects (in our case TD principal and interest), in a hierarchical manner. Since this study focuses on exploring the relations between TD concepts and their aspects, we believe that the Mantel test is more appropriate than traditional correlation, which would be able to accurately answer only goal (a). More details for the Mantel test are provided in Section 4.4.1. In addition, we have performed *Linear Mixed Equation* (LME) modelling to investigate the extent to which the obtained results are not affected by the random effect of project selection, but they are due to the examined factors and parameters; this supports the generalizability of the results. The main findings of the study validate the relation between TD Principal and TD Interest (illustrating that classes with similar levels of TD Principal tend to have similar levels of TD Interest), and that certain quality properties (e.g., coupling and size) should be prioritized while ranking refactoring opportunities.

The rest of the paper is organized as follows: Section 2 provides an overview of related work, in Section 3, we present the background information that is required for understanding underlying concepts. In Section 4, the case study design is overviewed. The results are presented in Section 5, and discussed in Section 6. Threats to validity are presented in Section 7, whereas Section 8 concludes the paper.

2 RELATED WORK

The goal of this section is to present: (a) works aiming to connect TD Principal to TD Interest—see Section 2.1; and (b) studies that focus on the quantification of TD Interest—see Section 2.2. We note that we do not discuss works on calculating TD Principal, since it is considered a straightforward task: According to Alves et al. [2], the principal is related to the effort / cost to eliminate the debt from a given system or artifact. Current software analysis tools offer estimates of *TD Principal* based on counts of detectable violations (e.g., SonarQube, CAST, Squore, etc.).

2.1 Relation between TD Principal and Interest

In the literature, we have identified five studies that aim at exploring the relation between TD Principal and Interest. Table 1 outlines the studies, by presenting the TD Principal / Interest assessors, and the main conclusion of each study. Next, we present these studies in detail, and compare them against our study.

Table 1. TD Principal – TD Interest Relation

Ref.	TD Principal Assessor	TD Interest Assessor	Main Outcome
[10]	Modularity anomalies as index of TD	Maintainability (expressed as stability) as a main characteristic related to	Improvement of modularity is related to important benefits in terms of stability, and lead to

Ref.	TD Principal Assessor	TD Interest Assessor	Main Outcome
		TD interest	the reduction of TD interest
[16]	Architecture roots (flawed structures)	Coupling and Cohesion (indicating higher maintenance effort)	TD items incur high maintenance penalties
[18]	SQALE method	Structural proxies (quality metrics)	Some quality metrics are positively related and others are negatively related to TD principal
[23]	Coupling between components	Defect-related activity	Highly-coupled components are more prone to defects, and costlier to maintain
[41]	Modularity Violations	Defect Proneness Change Proneness	Classes with more modularity violations and code smells, are more defect- and change-prone

Zazworka et al. [41], compare the similarities and differences among four approaches for TD identification, namely modularity violations, grime buildup, code smells and automatic static analysis. Given the fact that there are plenty of tools that can automatically detect a number of source code anomalies, the study considers four main techniques for TD detection, selected primarily by the criterion of authors' previous experience. The study aims at investigating if these approaches result in pointing out the same set of problematic issues. Moreover, the authors explore the extent to which the four techniques point to instances with high *TD Interest*. Since interest (i.e., the probable future cost of not fixing the debt) is regarded as difficult to detect and measure, the authors select to use two interest indicators, i.e., defect-proneness and change-proneness. The selection of the proxies has been made among a number of interest indicators, based on their correlation with problematic code manifestation and future maintenance cost. The authors conduct a case study, where they implement the four techniques on Apache Hadoop software. Their results show that: (a) different techniques identify different TD issues, (b) classes identified with high TD, with the use of modularity violations and code smells, seem to be more defect-prone, while modularity violations are strongly related to change-prone classes. Zazworka et al. [41] use four different approaches to TD identification and focus on defect-proneness and change-proneness as TD interest indicators, as their goal is the comparison among the different methodologies. In our work, as mentioned earlier, we opt to focus on one TD principal estimation methodology, while we propose a more detailed approach of TD interest estimation, based on well-established maintainability metrics.

Moreover, Conejero et al. [10] perform an empirical study aiming to evaluate if modularity anomalies at requirements level affect maintainability attributes and therefore increase system's interest. The study is based on a previously established framework that uses modularity metrics to identify and quantify modularity anomalies. A set of metrics is also used to assess the system's stability, as a particular maintainability attribute. The authors regard the presence of crosscutting concerns in a system as a negative effect on modularity, which in turn is considered as a significant index of TD [2]. Modularity violations are expected to raise interest, given the fact that they may affect different quality attributes and hence cause negative impact on the system's quality. The paper attempts to prove the relation between modularity anomalies at the early stages of software development, i.e., the requirements level, with software maintainability—with regard to stability. Since maintainability is deemed a main contributor to *TD Interest*, if the aforementioned relation is proven, then modularity violations can be linked to interest escalation. The study is performed on three different industrial applications. The results suggest the existence of a relationship between the Degree of Crosscutting properties, as a measure of modularity, and the stability of a certain feature. Therefore, the authors conclude that improvement of modularity could deliver important benefits in terms of stability, enhancing the maintainability of the system and lead to the reduction of *TD Interest*. Compared to our study, Conejero et al. [10] focus on modularity anomalies, as indicator of its quality, while they do not explicitly refer to principal. Also, they refer to stability as a maintainability attribute, whereas our work considers: inheritance, coupling, cohesion, complexity and size, as maintainability-related properties.

Additionally, Kosti et al. [18], explore the correlation between: (a) structural proxies (i.e., quality metrics); and (b) monetized values using the SQALE method, by conducting a case study on a set of 20 OSS projects. The results of the study suggest that a model of seven structural metrics, quantifying different aspects of quality (i.e., coupling, cohesion, complexity, size, and inheritance) can accurately estimate TD principal as appraised by SonarQube. More specifically, two coupling metrics (MPC and MOA) and one cohesion metric (LCOM) are positively related to technical debt principal, while CIS, NOP and DIT are negatively related to the accumulation of TD. While both the study of Kosti et al. [18] and ours use the same maintainability metrics, Kosti et al. do not explicitly refer to a connection between these metrics and TD interest.

McCormack and Sturtevant [23] address architectural debt by attempting to assess the potential value derived from a refactoring effort. To this end, they try to evaluate the relationship between system design decisions, which lead to TD, and increased costs of maintenance, which represent the interest to be paid. More specifically, the authors analyze how components with higher levels of coupling can be associated with higher maintenance costs. Firstly, based on their prior work, the authors apply a technique, namely Design Structure Matrix to analyze the system's architecture, capture the dependencies and calculate the coupling between components. This allows them to divide the components into groups, according to their levels of coupling, and then to characterize the system as a Core-Periphery one—if it possesses a large, dominant, cyclic group of components, namely the Core—or as a Hierarchical one—if it has small cyclic groups or no cyclic groups at all. In the empirical study, the authors analyze two large projects of similar size, one of them classified as a Core-Periphery system and the second as a Hierarchical one. The descriptive analysis supports the hypothesis that, for both systems, high-coupling components are more probable to experience defects and hence to be related with higher defect-related activity. The authors also perform a predictive analysis, by creating two statistical models, one to predict the probability of a component to experience a defect and another one to predict the volume of defect-related activity. Number of lines of code in a file and its cyclomatic complexity are used as control variables and are proven to be positive and statistically significant for both systems and both models. However, models with predictor variables added show a strong relation between components with high coupling and: (a) the likelihood of experiencing a defect, as well as (b) the defect-related activity. The authors also introduce a financial analysis, where they calculate a cost per line of code per year, by component category, to maintain each system. The analysis indicates that tightly coupled components cost more to maintain than loosely coupled ones. Finally, the study suggests that repaying architectural TD by reducing coupling and lowering maintenance costs may be valuable; however, managers should also consider the cost of these refactorings. In comparison to our study, the work of McCormack and Sturtevant [23] uses architectural design flaws that lead to tightly-coupled components as a TD proxy, and assumes that maintenance costs represent TD interest; our work focus on source code issues to identify TD principal and adopt a set of maintainability metrics as TD interest proxies, so as to provide a more thorough analysis of the relationship between the two terms.

Furthermore, Kazman et al. [16] focus on architectural TD in terms of flawed architecture structures, termed as architecture roots [39]: these are considered technical debt items that incur high maintenance penalties as identified through coupling and cohesion. They validate their methodology by conducting a case study with an industrial partner. However, Kazman et al. [16], do not differentiate between *TD Interest* and *TD principal*, and therefore it is not evident if they consider maintenance penalties as principal or interest, since the term debt is used collectively in that study.

Compared to related work, our study has a clear focus on identifying the relationship between TD Principal and TD Interest. To this end, it adopts well-validated valuation approaches, and provides an in-depth statistical analysis of the aforementioned relationship, based on the Mantel test and LME.

2.2 Quantification of TD Interest

In this section we present papers that deal with TD Interest calculation. In Table 2, we summarize studies that quantify TD interest, including those already presented in Section 2.1. In particular, in Table 2, we note how TD interest is assessed, if the study acknowledges the relation of TD Interest to maintainability, if it uses historical data (from software repositories) for TD interest calculation and whether they propose explicit metrics for assessing interest. We remind that maintainability and the use of historical data to calculate maintenance effort are the two core aspects of TD interest.

Table 2. TD Interest Assessment

Ref.	TD Interest Assessor	Maintainability	History	Metrics
[7]	The extra effort needed to maintain a system due to the accumulation of TD in terms of wasted software development time	x		
[10]	Maintainability (expressed in terms of stability) as a main characteristic related to TD interest	x	x	
[16]	Coupling and Cohesion indicate higher maintenance effort		x	
[18]	Structural proxies (i.e., quality metrics)		x	x
[23]	Defect related activity	x	x	x

Ref.	TD Interest Assessor	Maintainability	History	Metrics
[28]	Effort spent on maintenance activities based on historical data Effort needed to rebuild a system Level of software quality	x	x	x
[41]	Defect Proneness and Change-proneness	x	x	

Concerning the studies already discussed in Section 2.1, Conejero et al. [10] acknowledge that "*maintainability is one of the main characteristics contributing to Technical Debt interest*", and decide to capture it through the quality property of stability which is one of the most important maintainability attributes. Specifically, Conejero et al. [10] measure stability by calculating the number of use cases changed in each release of the three systems they study. They define as a change in a use case: (i) a modification of the feature that the use case addresses, or (ii) a modification, addition, or deletion in the system that affects the particular feature. Similarly, Zazworka et al. [41] recognize that defect-proneness and change-proneness—the two proxies they use for TD Interest—are connected to future maintenance costs. The authors use historical data on the number of times a class is involved in fixing bugs, that were injected, resolved, or alive in a version, to measure defect-proneness. They also use data on the number of changes affecting the class divided by the total number of changes in the repository, to calculate change-proneness.

Moreover, McCormack and Sturtevant [23], as mentioned earlier, aim at analyzing the relationship between design decisions and maintenance costs, generally deemed to represent TD Interest. The authors approach maintenance costs through maintenance effort, i.e., effort spent on defect related activities: a metric computed through bug-tracking and version control systems represents the development activity that aims at defect correction. On the other hand, Kazman et al. [16] focus on the relationship between architecture roots, i.e., flawed structures, and the penalties they incur in terms of higher maintenance costs. They retrieve data concerning the number of defects fixed, the number of changes associated with architecture roots that were fixed, and the number of lines of code committed to fix the defects and to make the changes during the prior year. They then estimate the cost of refactorings to calculate the penalty in terms of maintenance cost. However, as mentioned before, Kazman et al. [16] do not refer to this as TD interest. Finally, Kosti et al. [18] have used structural quality metrics, that are used to assess maintainability (coupling, cohesion, complexity, inheritance, and size metrics), as proxies of TD interest.

In addition to the five studies of Section 2.1, we have identified two more studies that attempt to estimate interest and relate it to maintainability. The first one is by Nugroho et al. [28], who use SIG/TUV's software quality assessment method for TD measurement. Particularly, they perform source code analysis that involves metrics, such as lines of code (LOC), code duplication, McCabe's cyclomatic complexity, parameter counts, and dependency counts, to map the system's quality properties to a five-star rating system, and calculate the Repair Effort, i.e., the effort needed to reach the ideal quality level, which represents the amount of the system's TD. With regard to the system's *TD Interest*, which is defined as "*the extra maintenance cost spent for not achieving the ideal quality level*", the authors estimate it as a function of: (a) the effort spent on maintenance activities within a year, calculated based on historical data, as a percentage of number of LOC estimated to change yearly for maintenance reasons; (b) an estimate of the effort needed to rebuild a system using a particular technology, determined by the total size of the system (measured in LOC or Function Points) and a language productivity factor; (c) a factor used to account for the level of quality, assuming that the higher the quality level, the less the maintenance effort.

Besker et al. [7], attempted to quantify *TD Interest* in terms of wasted software development time through an empirical study, based on the admission that interest is defined as the extra effort needed to maintain a system due to the accumulation of TD. They perform a web-based survey answered by 258 software stakeholders and they conduct interviews consisted of unstructured, semi-structured and fully structured questions with development teams within seven software companies. The results of the study show that, according to the respondents, 36% of all software development time is on average wasted because of paying the interest of TD. The study also reveals that the system age influences the wasted time, however there is no linear correlation between the two. Moreover, Architectural Design and Requirement TD seem to cause the most negative effect on software development, whereas there is no significant differentiation on how a stakeholder's role affects the perception of wasted time. Finally, the respondents estimate that a significant percentage of wasted time is spent on understanding and measuring TD issues.

The state of the art in quantifying TD Interest is based on maintainability and change proneness (either due to defects or new features). Furthermore, the metrics used as proxies for TD Interest are either structural ones, or rely on historical data. We adopt the same strategy and use both types of metrics.

3 BACKGROUND INFORMATION

In this section, we present background information. Section 3.1 describes the estimation of *TD Principal* using SonarQube (based on SQALE), while Section 3.2 presents our approach for assessing *TD Interest*.

3.1 TD Principal Calculation

For the purpose of this study, we have decided to estimate principal at the source code level, based on the computations provided by a widely used platform, namely SonarQube [3], [21]. SonarQube can assess the quality of software relying on quality measures and issues, such as coding rule violations. The platform algorithm was originally based upon an adopted version of the SQALE method proposed by Letouzey and Ilkiewicz [20], in which a remediation index is obtained for the requirements of an applicable Quality Model. For example, for a requirement stating that all files should have at least 70% code coverage, the corresponding remediation action is to write additional tests; a remediation function maps effort to each action. Finally, for each artifact, the remediation index relating to all the characteristics of the model is obtained by adding all remediation indices linked to all quality requirements. The resulting SQALE Index is considered to represent *TD Principal* of the source code.

Table 3. Default metrics-tags provided by SonarQube

Aspect TD Principal	Code Smell Tags	Description	Aspect TD Principal	Code Smell Tags	Description
Understandability	brain-over-load	There is too much to keep in your head at one time	Security / Runtime	Cwe	Relates to a rule in the Common Weakness Enumeration.
	confusing	Will take maintainers longer to understand than is really justified by what the code actually does		Bug	Something is wrong and it will probably affect production
Poorly Written Code	clumsy	Extra steps are used to accomplish something that could be done more clearly and concisely		owasp-.*	Relates to rules in the OWASP Top-10 security standards
	bad-practice	The code likely works as designed, but the way it was designed is widely recognized as being a bad idea		unpredicta-ble	The code may work fine under current conditions, but may fail erratically if conditions change
	design	There is something questionable about the design of the code		Suspicious	It's not guaranteed that this is a bug, but it looks suspiciously like one
	lock-in	Use of environment-specific features		Security	Relates to applications' security
	unused	Unused code		Pitfall	Nothing is wrong yet, but something could go wrong in the future
Coding Standards	Cert	Relates to rules in a CERT standard	Coding Standards	Misra	Relates to rules in MISRA standards
	Convention	Coding convention violation		sans-top25-.*	Relates to the SANS Top 25 Coding Errors, which are security-related

SonarQube calculates TD Principal by identifying code smells (as the corresponding Quality Model requirements) and calculating their remediation. For identifying the existence of ***code smells***, SonarQube version 7.9 (for Java) relies on 562 rules (e.g., "*Method overrides should not change contracts*", "*Package declaration should match source file directory*", "*Parameters should be passed in the correct order*", "*Unused labels should be removed*"). SonarQube rules are associated with (by default) nineteen tags (see Table 3[1]); however, the user is given the chance to create custom tags at will. Since the number of tags is quite high, it is expected to lead to a sparse table in the data collection phase, we decided to group the tags into 4 categories. We call these categories ***Aspects of TD Principal***: (a) *Understandability*, (b) *Poorly Written Code*, (c) *Security/Runtime*, *and (d) Coding Standards* (see Table 3). We note that some tags could potentially belong to different tag categories, as well as, additional tag categories

[1] From the nineteen default tags of SonarQube, we deleted user-experience, since it does not relate to software maintainability.

could have been created. For instance, groups could have also been created by inspecting the effects of a specific code smell, instead of the cause of the smell (e.g., *clumsy code* is a poorly written code, which has a negative effect on understandability). However, in our study we have selected not to consider cause-effect relations, in the sense that such a taxonomy would not uniquely categorize all tags. Thus, we have built the classification schema only based on the root of the smell and not on the affected categories. To eliminate (as much as possible) the objectivity of this categorization, each one of the senior researchers of the study proposed a classification of his own, and after some negotiation rounds (similar to the Delphi technique) they have reached a consensus on the Aspects of TD Principal, as well as the mapping of Code Smell Tags to them. Nevertheless, since this decision is still objective a relevant threat to construct validity has been identified. Moreover, we acknowledge that SonarQube is not a perfect solution for measuring TD principal, as a perfect solution does not exist; we expand on the limitations of using SonarQube in the Threats to Validity Section.

3.2 TD Interest Calculation

In this study, we calculate interest using the FITTED framework, as it has been proposed [4], [6], [8] and empirically validated in our previous work [6], [37]. The validation was performed in an industrial setting and contrasted the scores for TD Principal and TD Interest with the perception of software engineers. The results suggested a rank correlation for 0.83 for TD Principal and 0.73 for TD Interest. We only recap the basic notions of the FITTED framework here and refer to the aforementioned works for further details.

Assuming that a system has an *actual* implementation, and a *hypothetical* optimal implementation (in terms of maintainability—i.e., ease to maintain), maintaining the optimal system would require less effort than maintaining the actual system (see Figure 1). Despite the fact that a system can by no means be characterized as globally optimal, based solely on the optimization of some structural characteristics, there is a plethora of studies aiming at software optimization, guided by the application of software refactorings (e.g., [14][29][30])[2]. As shown in Figure 1, adding a new feature A to the optimal system would need a certain effort, noted as Effort$_{(optimum)}$, whereas adding the same feature to the actual system necessitates a larger effort, noted as Effort$_{(actual)}$. ***The difference between these two efforts represents the TD Interest that is accumulated during this maintenance activity.***

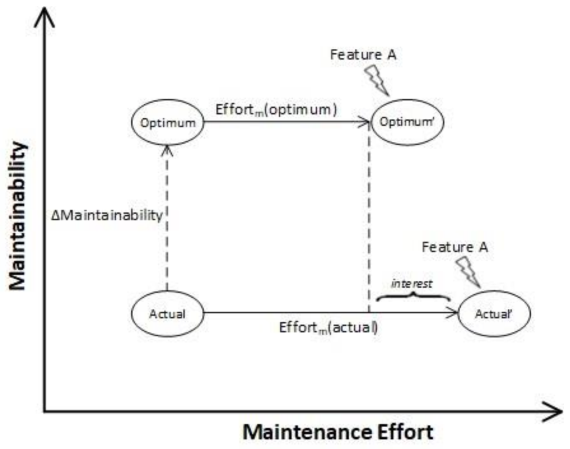

Fig. 1: Increased Maintenance Effort for TD items

According to FITTED [8], maintenance effort is inversely related to the maintainability of the system—see Eq. (1). Although the relation between effort and maintainability is not necessarily (or by definition) linear, several studies model maintenance effort (through regression modeling) as a polynomial of maintainability indicators (e.g., [38], [42]), achieving satisfactory prediction accuracy. In particular, van Koten and Grey [38] propose a linear model[3], whereas Zhou et al. [42] propose a multivariate adaptive regression spline model[4]. Despite the differences in the

[2] The relevant research area is termed search-based software engineering.

[3] $effort = c_0 + \sum w_i maintainability\_indicator_i$

[4] $effort = c_0 + \sum w_i \prod (maintainability\_indicator_i - t_i)$

modelling of the solutions, both studies suggest that there can be a linear relation between maintenance effort and maintainability.

$$Effort = a\frac{1}{maintainability} \tag{1}$$

Given Eq. (1), the maintenance effort for the optimal system (which is unknown), can be estimated as the product of the maintenance effort for the actual system and the ratio of the maintainability of the actual over the maintainability of the optimal system (we call this ratio *Maintainability Level*) [8]—see Eq. (2).

$$\frac{Effort_{optimum}}{Effort_{actual}} = \frac{\frac{a}{maintainability_{optimum}}}{\frac{a}{maintainability_{actual}}} = \frac{maintainability_{actual}}{maintainability_{optimum}} = MaintainabilityLevel \tag{2}$$

Finally, based on its definition in Figure 1, TD Interest can be calculated using the difference between the actual and the optimal effort, as follows [8]—see Eq. (3):

$$\begin{aligned} TD\ interest = \Delta Effort &= Effort(actual) - Effort(optimum) \\ &= Effort(actual) - Effort(actual) \times (MaintainabilityLevel) \\ &= Effort(actual) \times (1 - MaintainabilityLevel) \end{aligned} \tag{3}$$

In practice, the above calculation is multiplied by the constant value of Unit Cost of Maintenance (e.g. \$ or € per hour), but that is irrelevant for this study (although it is calculated), since the Mantel test employed in this study considers distance matrices. Therefore, as **aspects of TD Interest**, we consider (a) the Maintainability Level; and (b) the actual Maintenance Effort of the system under study. The former is related to structural properties of the corresponding system while the latter can be estimated using historical information, as elaborated below.

Maintainability Level. Although no single function can capture all aspects of quality, for the sake of simplicity, we assume that the optimal system is the one that optimizes a certain fitness function assessing the quality of software (e.g., in terms of complexity, cohesion, coupling, etc.). Thus, to calculate the maintainability level, we first identify a set of similar artifacts (e.g., classes, packages, systems—see [6]), we then calculate the optimal value of the metric score within the set of similar (in terms of lines of code, number of methods, cognitive complexity, etc.) artifacts. The maintainability optimal artifact is an artificial one that is assigned the "*best*" metric scores, among the similar artifacts: i.e., the metric score of lowest complexity, highest cohesion, lowest coupling, etc. For example, given five similar artifacts with complexity scores: 2, 5, 3, 8, 11; the artificial optimal artifact would be assigned a complexity score of 2. Then we calculate the average ratio of the metric score of the artifact under study, compared to the optimal value. Maintainability, although not associated to a universally accepted definition, is widely accepted as the ease of making changes into a system [15]. The set of metrics that we have selected to use in our study for quantifying maintainability (see Table 4) belong to well-known metric suites [9], [22]. The metrics selection was based on a secondary study by Riaz et al. [32], which reported on a systematic literature review (SLR) aimed at summarizing software metrics that can be used as maintainability predictors.

In particular, Riaz et al. [32] have performed a quality assessment of maintainability models, through a quantitative checklist, in order to identify studies of high-quality score, i.e., studies that provide reliable evidence. More specifically, the checklist was comprised of 19 questions and each model was assessed for each criterion by a three-point scale: yes, no, or partially, with associated scores of 1, 0, and 0.5 respectively. The range of the total score of each study was between 0 and 19. All studies that have scored 7 or below were excluded from the list of selected studies, whereas among the studies with the highest scores were those of van Koten and Gray [38], and Zhou and Leung [42]. Both studies (i.e., [38], [42]) have used the same definition of maintainability and they have been based on two metric suites proposed by Li and Henry [22] and Chidamber et al. [9], i.e., two well-known object-oriented set of metrics. The employed suites contain metrics that can be calculated at the source-code level, and can be used to assess well-known quality properties, such as inheritance, coupling, cohesion, complexity and size. We note that according to Riaz et al. [32], another study (performed by Misra [27]) scored equally to the previously mentioned ones. However, Misra was using metrics coming from multiple suites (2 out of the 4 are already considered), and we preferred to select those that were common in the studies.

Table 4. Maintainability Properties and Metrics

Property	Metric	Description
Inheritance (Inh)	DIT	Depth of Inheritance Tree: Inheritance level number, 0 for the root class.
	NOCC	Number of Children Classes: Number of direct sub-classes that the class has.
Coupling (Cpl)	MPC	Message Passing Coupling: Number of send statements defined in the class.
	RFC	Response for a Class: Number of local methods plus the number of methods called by class methods.
	DAC	Data Abstraction Coupling: Number of abstract types defined in the class.
Cohesion (Coh)	LCOM	Lack of Cohesion of Methods: Number of disjoint sets of methods in the class.
Complexity (Com)	CC	Cyclomatic Complexity: Average cyclomatic complexity of methods in the class.
	WMPC	Weighted Method per Class: Weighted sum of methods. Each method of the class is assigned to a weight equal to 1.
Size (Size)	SIZE1	Lines of Code: Number of semicolons in the class.
	SIZE2	Number of Properties: Number of attributes and methods in the class

Maintenance Effort: Since the evolution of the software cannot be predicted, it is not possible to foresee what kind of modifications will be made in a system during future releases. Hence, we base our assessment of future maintenance effort on historical data, by considering past effort spent on maintenance activities. More specifically, as maintenance effort we assume the average lines of code added/deleted/modified between all pairs of successive versions of a system. This strategy has been used in a variety of studies e.g., [10], [16], [18], [23], [28], and [41].

4 CASE STUDY DESIGN

Case study is an observational method that is used for studying phenomena in a real-life context [34]. This case study has been designed and is presented according to the guidelines of Runeson et al. [34].

4.1 Research Objectives and Research Questions

The goal of the paper, as mentioned in Section 1, is to investigate the relation between *TD Principal* and *Interest*, as well as the relation between the *Aspects of TD Principal* (see Section 3.1) and the *Aspects of TD Interest* (see Section 3.2). An overview of the aspects of *TD Principal* and *TD Interest*, is presented in Figure 2. We note that the calculation of each aspect at a higher level, is an aggregation of the lower level:

- We remind that ***TD Principal*** has four aspects that correspond to the categories of code smell tags. As explained in Section 3.1 TD Principal is calculated as the sum of TD Principal due to Understandability, Security/Runtime Issues, TD Principal due to Poorly Written Code, and TD Principal due to the Violation of Coding Standards.
- ***TD Interest*** on the other hand has two aspects (at the second level): Maintainability Level (ratio of maintainability of actual vs. optimal case) and Maintenance effort. The aggregation formula from the 2nd to the 1st level is provided in Section 3.2 (see Eq. 3). Maintainability is further decomposed into five structural properties, while Maintenance effort constitutes historical change. We consider these five structural properties and the historical changes as aspects of TD interest at the third level. The function for aggregating the 3rd level aspects to Maintainability Level is the average of the distance from optimal, as explained in Section 3.2.

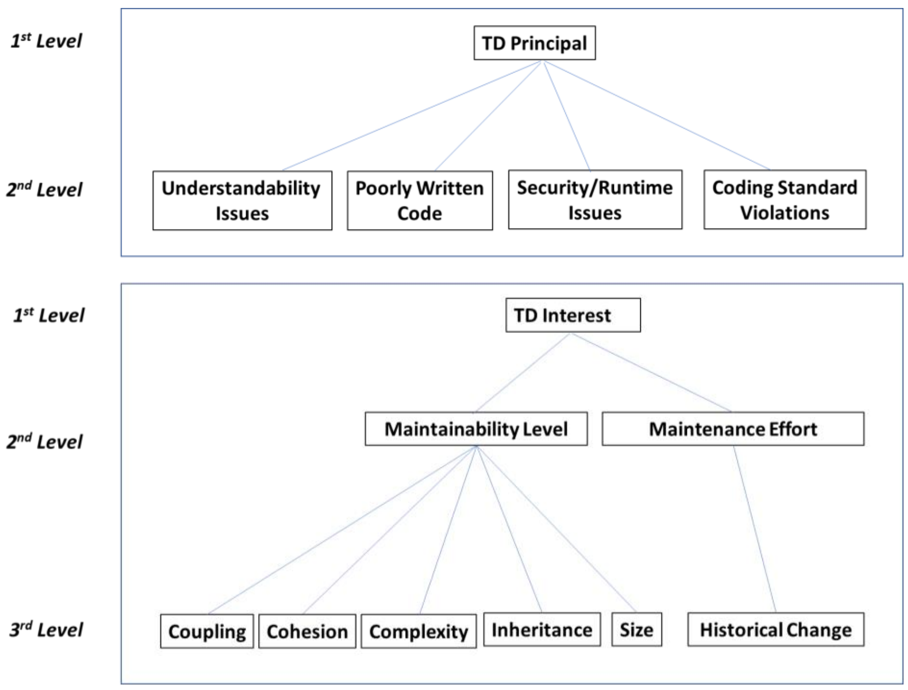

Fig. 2: Aspects of TD Principal and TD Interest

Based on this overview, and the goal of this study (i.e., to explore the relation between TD Principal and TD Interest, as well as their aspects), we have formulated three research questions.

RQ$_1$: *Is TD Principal related with TD Interest?*
This research question aims to explore if source code artifacts with a similar level of TD Principal are presenting similar levels of TD Interest. The existence of such a relation, would suggest that interest-related information could be subsumed by principal-related information, and therefore TD management, based only on *TD Principal* would make sense. With respect to Figure 2 this research question explores the relation of the 1st level of *TD Principal* with the 1st level of the *TD Interest* hierarchy.

RQ$_2$: *Which Aspects of TD Interest are more related to TD Principal?*
Given the fact that *TD Interest* calculation considers two 2nd level aspects (maintainability level and maintenance effort), we first explore if any one of these two *aspects* of *TD Interest* is more related to *TD Principal* (RQ$_{2.1}$). The answer to this RQ can shed light into the importance of the constituents of TD Interest, and guide researchers in their future attempts to quantify interest. Next, we focus on the maintainability level aspect of *TD Interest* and investigate which of the 3rd level aspects (i.e., coupling, cohesion, complexity, size, or inheritance) are more related to *TD Principal* (RQ$_{2.2}$). We note that we are not performing a similar analysis for the historical change data (i.e., we do not further split this concept), since no actionable outcome can be reached by such an investigation: you cannot change the history of a project. In contrast, structural properties that appear to be of more interest can be improved or prioritized; for instance, in case cohesion ends up being an important property, classes that suffer from low cohesion can be refactored (e.g., split method, split class, etc.). With respect to Figure 2 this research question focuses on the relation between the 1st level of *TD Principal* and the 2nd and 3rd Level of *TD Interest*.

RQ$_3$: *Which Aspects of TD Principal are more related to TD Interest?*
The third research question deals with comparing different *Aspects* of *TD Principal* (of the 2nd level), with respect to the interest that they are expected to incur. In this research question, we investigate if and which of the *Aspects* of *TD Principal* are related to the highest interest. This question becomes extremely relevant for preventing, prioritizing, and repaying specific aspects of *TD Principal* within the same technical debt item. For example, if it turns out that understandability issues are producing more interest compared to poorly written code issues, then the issues of the aspect should be prioritized. With respect to Figure 2 this research question focuses on the 2nd level of *TD Principal* and the 1st level of *TD Interest*.

4.2 Case Selection and Unit Analysis

The study of this paper is characterized as multiple, embedded case study [34], in which the cases are the OSS projects and the units of analysis are their classes. The reason for using OSS systems is the vast amount of available data in OSS repositories, in terms of versions and classes. To obtain data from high-quality projects (see Table 5) that evolve over a period of time, we require that the software systems:

- Are popular OSS project of the Apache community. This ensures that the investigated projects are recognized as important by the OSS community, i.e., there is substantial system functionality and substantial development activity in terms of bug-fixing and adding requirements.
- Are written in Java. We include this criterion because of the employed metric calculation tools.
- Contain more than 70 classes. This ensures that we will not include "toy examples" in our dataset. After data collection, a manual inspection of the projects has been performed to guarantee that the classes are not trivial.
- Have more than 5000 commits. We have included this for similar reasons to the first criterion. Although the selected number of versions is ad/hoc, it is set to a relatively high value, in order to guarantee high activity and evolution of the project. Also, this number of revisions provides an adequate set of repeated measures as input to the statistical analysis.

Table 5. OSS Projects Selected for the Case Study

Name	Short Description	#classes
Apache XML Graphics (XMLGraph)	Apache XML Graphics Commons is a library that consists of several reusable components used by Apache Batik and Apache FOP	109
Commons Math (ComMath)	It is a library for mathematics and statistics components	901
Commons Collection (ComColection)	The Apache Commons Collections package contains types that extend and augment the Java Collections Framework.	307
Commons Net (ComNet)	Apache Commons Net library implements the client side of many basic Internet protocols.	148
Commons IO (ComIO)	Commons IO is a library to assist with developing IO functionality.	113
Commons Jelly (ComJelly)	Jelly is a tool for turning XML into executable code.	73
Http Components – Core (HTTPCore)	Http Core is a set of HTTP transport components, used to build client and server services.	368
Http Components – Client (HTTPClient)	Http Components is responsible for creating and maintaining a toolset of Java components.	294
Apache Struts (Struts)	Struts is an MVC framework for creating modern Java web applications.	622
Xerces 2 Java (Xerces2Java)	Xerces2 is a library for parsing, validating and manipulating XML files	665

4.3 Data Collection and Pre-Processing

For the quantification of TD Principal, we used the SonarQube API to obtain the TD Principal metric for each one of the classes of the projects under analysis. Next, for each row in the dataset we recorded the number of instances of issues in each Tag Category, which are concentrated in the class (5 variables). Regarding TD Interest, we used the TDM toolkit[5] of the SDK4ED platform[6], developed in the context of the SDK4ED project[7]. On the completion of data collection, each class (unit of analysis) was characterized by 12 variables—10 maintainability metrics, maintenance history, and TD Interest. The pre-processing was completed by the deletion of rows that contained missing values. Missing values can be found in cases a maintainability metric cannot be calculated: e.g., CC cannot be calculated for abstract methods, or LCOM cannot be calculated for interfaces.

[5] https://github.com/AngelikiTsintzira/Technical-Debt-Management-Toolbox

[6] http://sdk4ed.se.uom.gr/

[7] https://sdk4ed.eu/

4.4 Data Analysis Methodology

Since the main objective of the study is to examine whether classes that present similar levels of TD Principal, also produce a similar amount of TD Interest, classical approaches such as correlation analysis are not able to provide straightforward answers. Although traditional correlation coefficients can be used to explore the nature and strength of the pairwise relationship between variables of a multivariate dataset, they can only assess which subset of software metrics is associated to the TD Principal at the lowest level of the quality hierarchy. The current approach focuses on the similarity (dissimilarity) of classes to examine the association between TD Principal and TD Interest, by considering that *TD Principal* and *TD Interest* are multifaceted concepts that can be assessed through various aspects (see Section 3 and Figure 2) synthesizing in turn, multidimensional spaces of metrics. Through this approach, we believe that researchers and practitioners will be able to acquire significant knowledge in a more straightforward and intuitive manner, since the interpretation of the results resembles the way of human decision-making by comparing similar cases (i.e., classes), as in case-based reasoning (CBR) process.

4.4.1 Mantel Test

To this regard, we make use of a multivariate statistical methodology, namely the Mantel test [24] that has been extensively applied in many scientific areas such as: health, ecology, biology, population genetics. The rationale of the approach is to evaluate the association between the corresponding positions of all pairs of observations from two dissimilarity matrices computed by either univariate or multivariate data. The procedure is further augmented with a randomization mechanism based on the permutation of the rows and columns of one of the two dissimilarity matrices, to test the null hypothesis that the two dissimilarity matrices are uncorrelated. We have also to point out that the Mantel test has been used in software engineering: (a) to evaluate the difference between perspectives in the determination of the relative importance of impact analysis issues of software [33]; and (b) for the calibration of the analogy-based software cost estimation model [17]. An outline of the approach that we have used for applying the Mantel test, in our study, is described below:

1. Each analysis element (i.e., box in Figure 2) is represented by either a vector or a matrix, in which the rows correspond to the n classes of each project and columns comprise the metrics representing a specific element (see Table 6). A simple concept is represented by a column vector (Table 6), e.g. TD Principal, the measurements can be represented in the following form ${TD_{\text{Principal}}}^T = \left({TD_{\text{Principal}}}_{c_1}, \dots, {TD_{\text{Principal}}}_{c_n}\right)^T$. Regarding multifaceted concepts, e.g. the quality property of *Size* evaluated by two metrics (*Lines of Code* and *Number of Properties,* 3rd Level, Table 6), the measurements can be compiled in the form of a tabular matrix

$$\begin{bmatrix} SIZE1_{c_1} & SIZE2_{c_1} \\ \vdots & \vdots \\ SIZE1_{c_n} & SIZE2_{c_n} \end{bmatrix}$$

   where $SIZE1_{c_i}$ and $SIZE2_{c_i}$ represent the measurements of i class regarding the two metrics *Lines of Code* and *Number of Properties,* respectively. The followed approach provides us the ability to capture a relation between sets of metrics that quantify two concepts that can be either simple or multifaceted; whereas traditional univariate correlation analysis would be able to indicate the relationship between pairs of scalar metrics.

Table 6. Analysis Element Representation

Level	Concept	Element	Representation	Metrics
1	TD Principal	TD Principal	Vector	Total TD Principal
1	TD Interest	TD Interest	Vector	Total TD Interest
2	TD Principal	Types of TD Principal Issues	Matrix	TD Principal due to Understandability Issues TD Principal due to Security/Runtime Issues TD Principal due to Poorly Written Code TD Principal due to the Violation of Coding Standards
2	TD Interest	Maintainability Level	Matrix	Coupling, Cohesion, Complexity, Size Inheritance
2	TD Interest	Maintenance Effort	Vector	Average Number of Lines Changed between Commits
3	TD Interest	Coupling	Matrix	Message Passing Coupling

Level	Concept	Element	Representation	Metrics
				Response for a Class Data Abstraction Coupling
3	TD Interest	Cohesion	Vector	Lack of Cohesion of Methods
3	TD Interest	Complexity	Matrix	Cyclomatic Complexity, Weighted Method per Class
3	TD Interest	Size	Matrix	Lines of Code, Number of Properties
3	TD Interest	Inheritance	Matrix	Depth of Inheritance Tree, Number of Children Classes

2. From each analysis element we calculate a distance matrix for each pair (i, j) of n classes for a given project. For the case of *TD Principal* and *TD Interest* (first levels of hierarchies), the first step of the method involves the evaluation of two distance matrices, Distance(TD$_{\text{principal}}$) and Distance(TD$_{\text{interest}}$) representing the distances between the pair of classes (i, j) of the two vectors (1st Level, Table 6). In the case of a multifaceted TD concept (e.g. quality property of *Size*, 3rd Level, Table 6), the distance matrix is evaluated based on the measurements of all the associated metrics (see Table 6). We note that dissimilarities between each pair of classes are evaluated on the standardized measurements ($[0,1]$) to be immune to metrics' range. For example, the investigation of the relation between *TD Principal* (1st Level) and the element of *Size*, described by two metrics (*Lines of Code* and *Number of Properties*) is based on distance matrices evaluated through the following formulae:

$$a_{ij} = \sqrt{\left(TD_{\text{Principal}_{c_i}} - TD_{\text{Principal}_{c_j}}\right)^2} \tag{2}$$

$$b_{ij} = \sqrt{\left(SIZE1_{c_i} - SIZE1_{c_j}\right)^2 + \left(SIZE2_{c_i} - SIZE2_{c_j}\right)^2} \tag{3}$$

Table 7. Distance-Matrix Example for TD Principal (1st Level) and Size Element of TD Interest

Distance (TD$_{\text{Principal}}$)		**Classes**			
		C_1	C_2	…	C_n
Classes	C_1	0	α_{12}	…	α_{1n}
	C_2	α_{21}	0	…	α_{2n}
	…	…	…	…	…
	C_n	α_{n1}	…	…	0

Distance (TD$_{\text{Interest-Size}}$)		**Classes**			
		C_1	C_2	…	C_n
Classes	C_1	0	b_{12}	…	b_{1n}
	C_2	b_{21}	0	…	b_{2n}
	…	…	…	…	…
	C_n	b_{n1}	…	…	0

3. Then, we take separately for each project all possible combinations of analysis elements (i.e., pairs of boxes: one from the *TD Principal* and another from *TD Interest* hierarchies), and calculate the *Mantel's r correlation coefficient* between the corresponding matrices.
   a. for RQ$_1$, we use the vectors of TD Principal and TD Interest of the first level
   b. for RQ$_{2.1}$, we use the vector of TD Principal, the matrix of Maintainability Level and the vector of Maintenance Effort
   c. for RQ$_{2.2}$, we use the vector of TD Principal, the matrices of Coupling, Cohesion (the only vector), Complexity, Size, and Inheritance
   d. for RQ$_3$, we use the vector of TD Interest and the matrix of Types of TD Principal Issues

We have also to note that due to the symmetrical nature of a distance matrix, only the elements from the upper or lower triangle matrix are used.

4.4.2 Statistical Inferential Process

After the discovery of similarity patterns, the next critical issue is to investigate whether the observed phenomena can be generalized to the population of OSS projects. Given the fact that the case study of the paper is characterized as multiple (i.e., using many projects as subjects—see Section 4.2), there is an imperative need to adopt an appropriate statistical inferential mechanism in order to derive conclusions regarding the population of OSS projects with similar characteristics. Towards this direction, an aggregated dataset comprising the Mantel's correlation coefficients evaluated from the classes of each OSS project for paired analysis elements is constructed. For example, for the case of RQ$_2$, the dataset can be expressed via a long-format matrix as presented in Table 8. As we can observe, each row of the matrix comprises the Mantel coefficient evaluated from the distance matrices of *TD Principal* and the alternative two aspects of *TD Interest* (Maintainability Level and Maintenance Effort) for a given project of our experimental setup. The question is to examine the potential effects of aspects of *TD Interest* on *TD Principal*.

For this reason, we use the *Linear Mixed Effects* (LME) modelling statistical technique [31] that is able to model simultaneously two types of effects that are (a) the fixed and (b) the random effects. In the terminology of LME models, the term "*fixed effect*" is used to depict factors influencing the mean value of a response variable, whereas a "*random effect*" (i.e. *Project* in our experimental setup) may have an impact on the variance of the response variable. The reason to control the project as a random effect is that our dataset represents OSS projects that are selected from an infinite unknown population of projects.

Regarding RQ$_{2.1}$, our aim is to examine whether there is a difference between the 2nd Level Aspects of *TD Interest* on how they are related with the *TD Principal*. We essentially investigate which of the two aspects correlates better with TD Principal. This difference can be formally modeled as an effect of the factor "*TD Interest Aspects*" (with two discrete levels, namely *Maintainability Level* and *Maintenance Effort*) on the Mantel correlation in the presence of the project's random effect. So, we actually consider the Mantel r as response variable while the type of aspect (fixed effect) and the project (random effect) are the two explanatory variables (Table 8). As far as the second goal of RQ$_2$ concerns, which is the investigation of whether there are 3rd Level Aspects of *TD Interest* that are more related to *TD Principal*, a similar approach is followed. In this case, the repeated measures design is represented via a matrix, where the examined effect is the 3rd Level *Aspect of TD Interest*: *Inheritance*, *Coupling*, *Cohesion*, *Complexity*, and *Size*) with the same random effect, i.e. the *Project*. Finally, a similar process has been also applied for RQ$_3$. The rest of the repeated measure designs is presented in the Appendix.

Table 8. Repeated Measures Design (RQ$_{2.1}$)

Mantel r	**TD Principal**	**Aspect of TD Interest**	**Project**
r_{11}	Total TD Principal	$Maintenance\ Effort$	Project 1
r_{21}	Total TD Principal	$Maintainability\ Level$	Project 1
…	…	…	…
r_{1n}	Total TD Principal	$Maintenance\ Effort$	Project *n*
r_{2n}	Total TD Principal	$Maintainability\ Level$	Project *n*

5 RESULTS

In this section, we provide the answers to the RQs of this study. In Table 11 and Figure 3, we present the descriptive statistics for the aspects of TD Interest (second and third level) and TD Principal (second level), respectively.

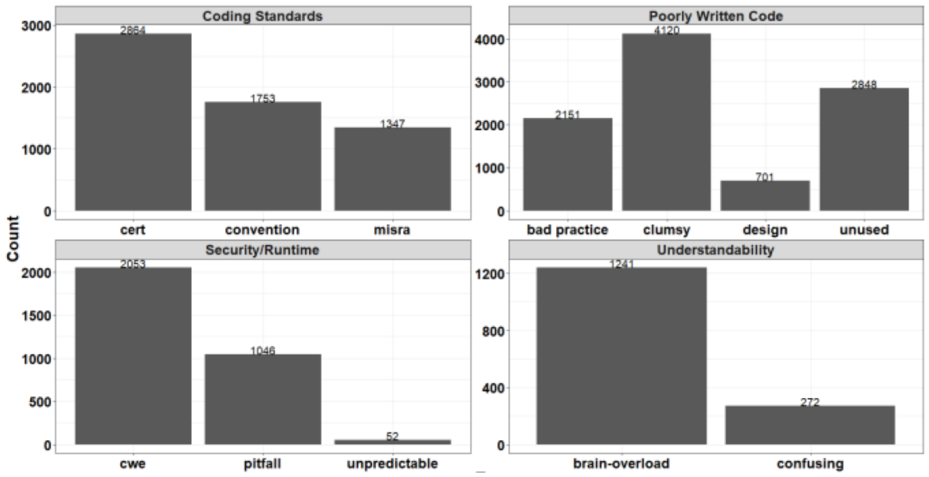

Fig. 3: Aspects of TD Principal Frequency

Note: *Aspects of the TD Principal are represented as counts of detected issues in all projects. Code Smell Tags that are not presented in Figure 3 (compared to Table 3) have zero instances in our dataset.*

Table 11. Descriptive Statistics for TD Interest and its Aspects

Metric		*N*	*M*	*SD*	*Min*	*Max*
TD Interest		3599	59.12	114.62	0.92	1698.10
Aspect of TD Interest	**Metric**	*N*	*M*	*SD*	*Min*	*Max*
Maintenance Effort – Historical Change	LOC	3599	46.04	83.90	0.20	1401.00
Maintenance Difficulty – Inheritance (Inh)	DIT	3600	2.17	1.40	1	9
Maintenance Difficulty – Inheritance (Inh)	NOCC	3600	0.76	2.55	0	41
Maintenance Difficulty – Coupling (Cpl)	MPC	3600	41.00	106.17	0	2531
Maintenance Difficulty – Coupling (Cpl)	RFC	3600	31.68	41.43	0	532
Maintenance Difficulty – Coupling (Cpl)	DAC	3600	0.29	1.05	0	20
Maintenance Difficulty – Cohesion (Coh)	LCOM	3389	75.27	356.25	0	8759
Maintenance Difficulty – Complexity (Com)	CC	2790	1.51	1.27	1	22.57
Maintenance Difficulty – Complexity (Com)	WMPC	3600	9.17	14.45	0	276
Maintenance Difficulty – Size (Size)	SIZE1	3600	56.50	91.80	1	1361
Maintenance Difficulty – Size (Size)	SIZE2	3600	11.12	17.16	0	238

Note-1: N refers to the number of classes in which the metric was calculated (e.g., CC cannot be calculated for interfaces / abstract classes). *M*, *SD*, *Min Max* stand for: mean, standard deviation, minimum and maximum of the metric
Note-2: The range of values for all the metrics that represent the aspects of TD Interest is [0, +∞) and TD Interest itself is measured in euros, using as Unit Cost of Maintenance the 1.8324 dollars per line of code (see [8])

5.1 Relation between TD Principal and Interest (RQ_1)

Mantel's correlation coefficients (r) for each pair of *TD Principal* and *TD Interest*, as evaluated from the classes of each OSS project, are summarized in Figure 4 ranging from weak ($r_{min} = 0.271$, ComNet) to very strong ($r_{max} = 0.820$, ComIO) correlation. The y-axis of Figure 4, demarcates the regions of "*no*", "*weak*", "*moderate*", "*strong*", or "*very strong*" relation [43], whereas the x-axis does not have a specific conceptual interpretation: it is just used for spreading projects in the full-width of the graph, to facilitate readability. Based on the evaluation of coefficients and their corresponding p-values, we can observe that there is noted a statistically significant correlation between the distance matrices of *TD Principal* and *TD Interest* for the whole set of the examined projects implying that classes with similar levels of TD principal have similar levels of TD interest. Based on the scales presented in Figure 4, for 80% of the projects the relation between TD Principal and TD Interest is at least moderate.

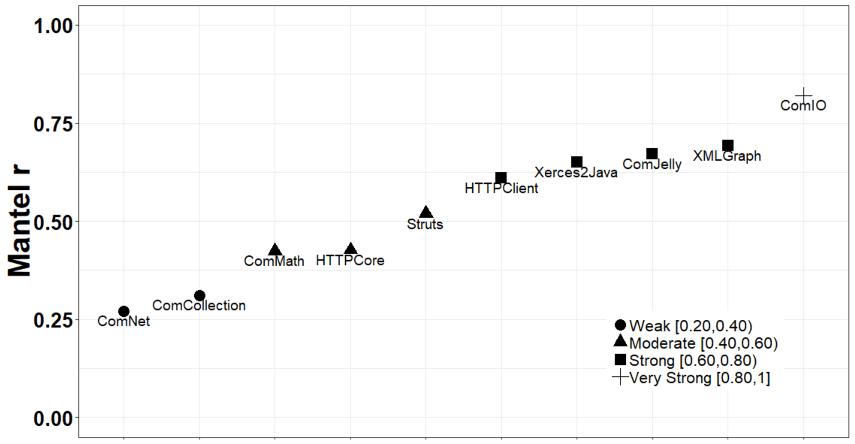

Fig. 4: Mantel's coefficients between Principal and Interest

In the second step of the analysis, an LME model is fitted to investigate the strength of the observed association to the population of OSS projects. The parameter of the LME model fitted on the accumulated results demonstrates a mean value of 0.540 signifying a moderate correlation between *TD Principal* and *TD Interest* aspects.

Classes with similar levels of TD Principal tend to have similar levels of TD Interest. The strength of this relation is at least moderate and statistically significant.

5.2 Relation between TD Principal and Aspects of TD Interest (RQ$_2$)

Comparing 2nd-level Aspects of TD Interest (RQ$_{2.1}$). Given the relation between principal and interest, we further drill down to investigate if either of the second-level aspects of interest, Maintainability Level (structure) or Maintenance Effort (history) presents a higher effect on this relation. Figure 5 presents the distributions of the correlation coefficients between *TD Principal* and the two aspects of *TD Interest* (Maintainability Level and Maintenance Effort), in which each asterisk denotes the sample coefficient for an examined project of the case study. The p-values of the tests revealed statistically significant correlations for all pairwise comparisons between *TD Principal* and both aspects of *TD Interest*. Regarding the strength of the association for the Historical aspect of *TD Interest (i.e., Maintenance Effort)*, the coefficients range, again, from weak ($r_{min} = 0.160$, for Xerces2Java) to strong ($r_{max} = 0.700$, for ComJelly), whereas the results are similar for the Structural aspect of *TD Interest* ($r_{min} = 0.230$, for ComIO,$r_{max} = 0.610$, for ComJelly).

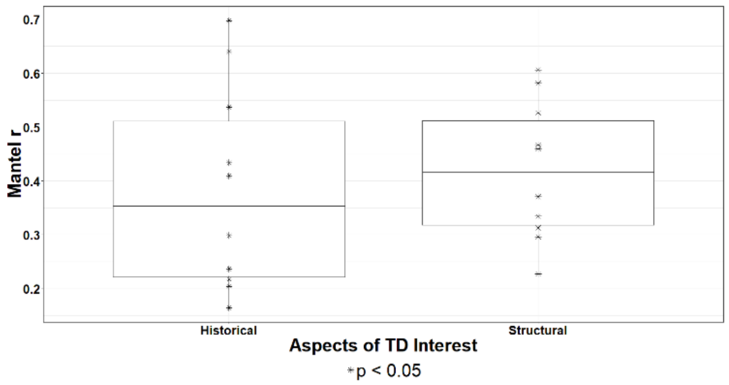

Fig. 5: Distributions of coefficients: TD Principal and Aspects of TD Interest (Maintainability Level / Maintenance Effort)

To investigate the effect of the *Aspects of TD Interest* on the evaluated correlation coefficients, we fitted again, an LME model. The model did not reveal a statistically significant main effect of the *Aspects of TD Interest* on the examined coefficients ($F = 0.415, p = 0.536$) denoting that the two aspects are correlated to *TD Principal* to the same extent. The parameter estimates for the population mean values of correlation are 0.384 and 0.418 for *Maintainability Level and Maintenance Effort* aspects, respectively. Given the fact that both aspects of interest have merit, and do not differ significantly, it is important to investigate if the aggregated measure of *TD Interest* is more related to *TD Principal* compared to the association of the two aspects (in isolation) to *TD Principal*. The results suggest that *TD Principal* seems to present a higher correlation to the aggregated *TD Interest* metric (1st level of the *TD Interest* hierarchy) compared to the aspects of *TD Interest* (second level of hierarchy).

To investigate the generalizability of the aforementioned results, we fitted an LME model incorporating the fixed effect of the factor *Hierarchy Level* (1st Level/2nd Level). Based on the results of the previous model (i.e. insignificant differences between the two *Aspects of TD Interest*), we have to clarify that the category *Second Level* aggregates the correlation coefficients evaluated from both *Maintainability Level and Maintenance Effort* aspects of *TD Interest*. The model revealed a statistically significant main effect of the factor *Hierarchy Level* on the mean values of the correlation coefficients ($F = 5.927, p = 0.025$).

Table 12. LME - Main Effect of Factor Hierarchy Level of TD Interest

	b	***SE***	***Df***	***t***	***p***
First Level	0.540	0.053	19	10.248	< 0.001
Second Level	-0.139	0.057	19	-2.435	0.025
Note: *b*, *SE*, *df*, *t*, *p* stands for parameter estimations, standard error of the estimates, degrees of freedom, t-statistic and p-value, respectively. The reference category for factor Hierarchy Level of TD Interest is First Level.					

Interpreting the parameter estimates of the LME model (see Table 12), the aggregated measure of *TD Interest* presents a higher mean correlation value to *TD Principal* compared to the two aspects in isolation. The negative sign of the parameter estimate for the *Second Level* of *TD Hierarchy* implies that the mean correlation coefficient is 0.139 lower than the corresponding value evaluated from the *First Level of TD Hierarchy* (0.540). For the rest of this study, we do not perform any analysis on the 2nd level of *TD Interest*.

The aggregated measure for TD Interest seems to be a more representative metric for capturing the divergence of classes in terms of their TD principal compared to the individual Historical or Structural metrics

Comparing 3rd-level Aspects of TD Interest ($RQ_{2.2}$). In this sub-section we investigate whether there are certain structural aspects of *TD Interest* (3rd level of the *TD Interest* hierarchy), that are more related to *TD Principal* ($RQ_{2.2}$). The main motivation for this, as explained in Section 2.2, is the fact that structural properties are directly linked to actionable results; e.g., a tentative importance of lack of cohesion, can underline the importance of conforming to the Single Responsibility Principle [25].

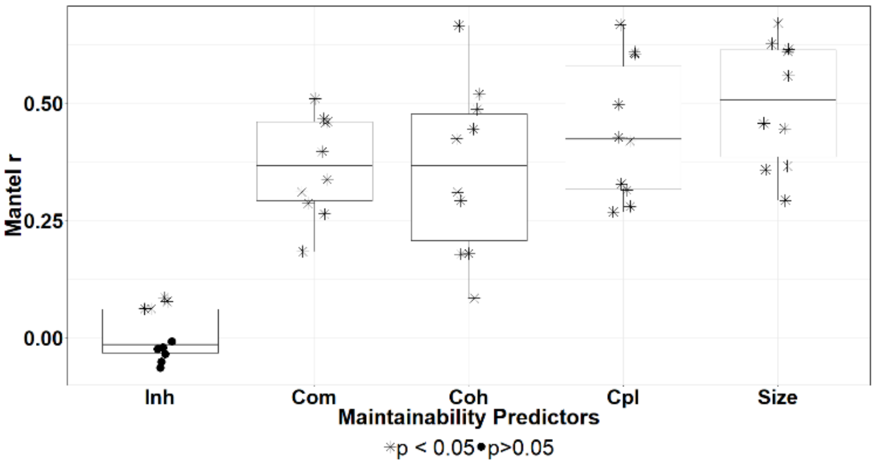

Fig. 6: Distributions of Mantel's coefficients between TD Principal and Maintainability Predictors

Figure 6 summarizes the Mantel's correlation coefficients (r) evaluated from the classes of each OSS project based on the dissimilarities for each pair of *TD Principal* and maintainability metrics (Inh: *Inheritance*, Com: *Complexity*, Coh: *Cohesion*, Cpl: *Coupling*). The results suggest that the strength is heavily dependent on the type of maintainability predictor, ranging from statistically significant very weak correlation (e.g. for *Inh* predictor, $r_{min} = 0.06$, ComMath) to strong correlation (e.g. for *Size* predictor, $r_{max} = 0.671$, HTTPClient). Generally, *Size* seems to be the most related maintainability predictor of *TD Interest* to *TD Principal*. In contrast, *Inheritance* presented low coefficients that are statistically significant for 4 out of 10 cases.

Table 13. LME - Main Effect of Aspect of TD Interest

	b	*SE*	*df*	*t*	*p*
Intercept	0.061	0.062	30	0.988	0.331
Complexity	0.306	0.063	30	4.844	< 0.001
Cohesion	0.297	0.063	30	4.705	< 0.001
Coupling	0.381	0.063	30	6.021	< 0.001
Size	0.439	0.063	30	6.947	< 0.001
Note: *b*, *SE*, *df*, *t*, *p* stands for parameter estimations, standard error of the estimates, degrees of freedom, t-statistic and p-value, respectively. The reference category for factor maintainability predictor of TD Interest is Inh					

The findings of the LME for the accumulated results present a statistically significant coefficient, demonstrating a significant main effect of the maintainability predictors on the response variable, $F = 13.061$ and $p < 0.001$. The parameter estimates of the model (Table 13) reveal that the mean value of correlation between *TD Principal* and *Inheritance* (reference category of the FITTED model) is very weak 0.061 ($r_{Inh} = b_{Intercept} = 0.061$) and significantly lower than the corresponding population mean values of *Coupling* ($r_{Cpl} = b_{Intercept} + b_{CpI} = 0.061 + 0.381 = 0.442$), *Cohesion* ($r_{Coh} = b_{Intercept} + b_{Coh} = 0.061 + 0.297 = 0.359$), *Complexity* ($r_{Com} = b_{Intercept} + b_{Com} = 0.061 + 0.306 = 0.367$), and *Size* ($r_{Size} = b_{Intercept} + b_{Size} = 0.061 + 0.439 = 0.500$).

The post-hoc analysis through Tukey's HSD test signifies statistically significant differences ($p < 0.001$) in all pairs between *Inheritance* and the other four quality properties (Figure 7—cases in which the error bar does not cross the vertical dashed line on 0.0). Regarding the *Size*, there is a statistically significant difference with *Complexity* ($p = 0.030$) and *Cohesion* ($p = 0.017$), but no difference compared to *Coupling* ($p = 0.702$). Therefore, we consider the strength of the relation of *Coupling* to *TD Principal*, similar to the strength of the relation between *Size* and *TD Principal*. Finally, the rest pairwise comparisons do not indicate statistically significant differences.

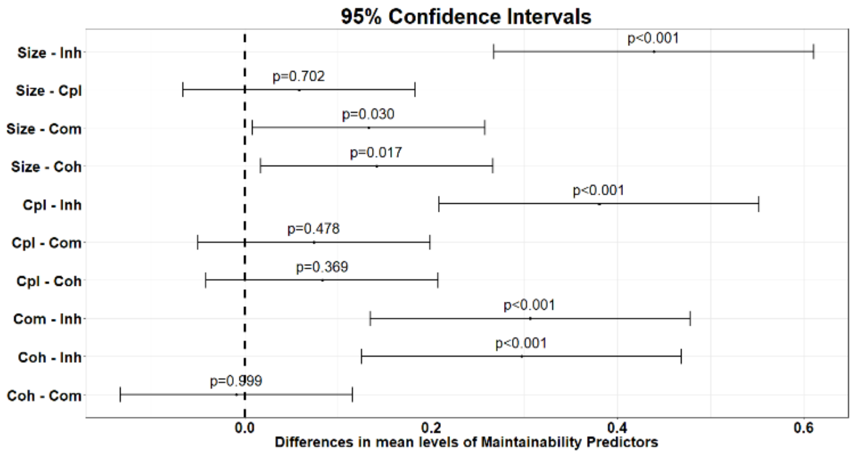

Fig. 7: Post-hoc analysis for LME model (main effect of Maintainability Predictors of TD Interest)

Size and Coupling are the maintainability-related properties that are most closely related to TD Principal, followed by Cohesion and Complexity.

5.3 Relation between Aspects of TD Principal and TD Interest (RQ$_3$)

In this section, we present the results on the relation between aspects of TD Principal and TD Interest (RQ$_3$). We remind that the aspects of TD Principal correspond to the four tag categories (Understandability, Poorly Written Code, Security/Runtime, and Coding Standards), that group the nineteen tags of code smells (see Section 3.1). Subsequently we evaluate the correlation of those four aspects to the first level of the *TD Interest Hierarchy*. *TD Principal* aspects present statistically significant correlations to *TD Interest* for the majority of the examined projects (37 out of 40 cases—see Figure 8) ranging from weak ($r_{min} = 0.137$, ComCollections) to strong ($r_{min} = 0.699$, ComJelly). Regarding the LME model incorporating the factor aspects of TD Principal, the findings did not reveal a statistically significant main effect on the examined coefficients ($F = 1.126$, $p = 0.356$).

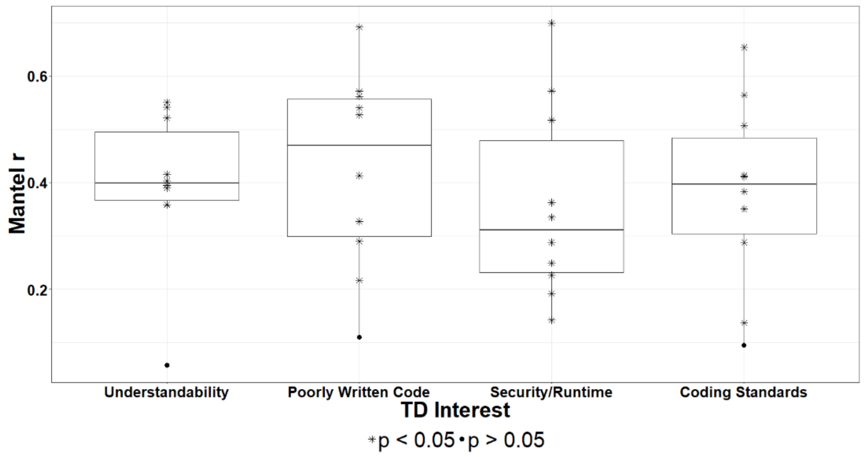

Fig. 8: Distributions of Mantel's coefficients between Tag Categories of code smells and TD Interest

All four categories of code smells (aspects of TD Principal) present a moderate relation to TD Interest.

6 DISCUSSION

In this section we discuss the main findings of this paper, first interpreting the obtained results, and then providing useful implications for researchers and practitioners.

Interpretation of results. The goal of this study is to explore the relation between *TD Principal* and *Interest*, as well as the relation between the aspects of both concepts. First, we have been able to provide empirical evidence on the existence of a relation between principal and interest. Therefore, although no causal effect can be assumed between principal and interest, we have provided the first well-based indications on the existence of a relation between interest and principal: classes with similar levels of TD Principal tend to produce similar levels of TD Interest. We note that identifying the form of the relationship (linear or any other type) would require a different kind of analysis. Thus, the existence of a relation similar to the one of economics still needs investigation, probably through a different study setup that can assess causality.

We have also unveiled a relation between TD Principal and 4 (out of 5) 3rd level aspects of *TD Interest* (namely: size, coupling, cohesion, and complexity). This relation appears to be stronger for size and coupling, and less strong for cohesion and complexity:

- *Size*: The relation between TD Principal and size (i.e., classes of similar size tend to have similar TD Principal) is intuitive in the sense that the more lines of code exist in the system, the more rules are expected to be violated. However, in most cases, this is not an actionable result as refactoring only for the sake of reducing size is very rarely done. In contrast, the size of software presents a linear growth over time; thus, it is of paramount importance that new code inserted into the system has as few rule violations (TD principal) as possible. Nevertheless, this observation provides two interesting implications: (a) TD Principal normalization (by size) makes sense for comparing classes of different sizes; and (b) the identification of design hotspots (in terms of TD Principal) should not be performed at a system-wide, but between similar (in terms of size) neighborhoods of classes; in the sense that artifacts of different size will not be directly comparable.
- *Coupling****,*** *Cohesion* and *Complexity*: The relation of TD Principal with these three quality attributes (Aspects of TD Interest—i.e., classes with similar TD Principal tend to have similar levels in these three properties), is indirect: we conjecture that developers who pay attention to software design (e.g., improved modularity, or low complexity) are also careful not to violate source code programming rules. The fact that coupling is more strongly related to TD Principal compared to cohesion and complexity denotes that a property reflecting the design rather than the implementation, is more important for maintenance. We remind that coupling, as calculated in this study (i.e., MPC, RFC, and DAC), can be calculated from design level artifacts (e.g., UML class or sequence diagrams); whereas complexity (CC—count of iteration and selection statements) and cohesion (LCOM—attributes used in method bodies) can only be captured by parsing source code artifacts.

Finally, regarding the specific aspects of TD Principal, the results suggest that the four aspects do not differ statistically significantly, in terms of the TD Interest that they are associated with. We would expect that TD interest (in the way that it is assessed in this study) as a structural property, is conceptually closer to two of the TD Principal aspects: code understandability and design practice violations. However, the other two TD Principal aspects (coding standards and run-time and security violations) seem equally related to TD interest. This is not intuitive especially for the relation between run-time and security violations and TD interest; a study investigating a possible causality between the two would be particularly interesting.

Implications to Researchers and Practitioners. Regarding *practitioners*, there has always been demand for an accurate calculation of interest to drive the prioritization of repaying TD. While there are relatively mature ways to calculate TD principal (mostly through source code analysis), TD interest is more elusive as it depends on knowing future changes. The establishment of a relation between TD principal and interest, implies that TD principal can be safely used for TD prioritization: paying back the TD items with the highest principal will very likely also reduce the TD interest paid in the system. In practice, prioritization of TD items is required whenever a development team receives an intractable number of refactoring suggestions from a TDM tool (which is the usual case when large rulesets are applied on large software systems). Despite the fact that practitioners could either prioritize based on TD Principal or TD Interest; in fact, they have long been prioritizing TD items with large principal. Our results provide empirical evidence that this is a sound practice, since the amount of *TD Principal* is related to the amount of *TD Interest, in the sense that classes with similar levels of TD Principal tend to have similar levels of TD Interest*. However, this observation does not downgrade the significance of TD Interest assessment: we have also provided evidence that especially for TD repayment, emphasis should be placed on improving specific quality properties (i.e., coupling and size), which have proven to be linked to the concentration of more TD Principal. Additionally, based on the findings of this study, some interesting future work opportunities have been identified. Therefore, we encourage *researchers* to:

- ***Study of causality***. The establishment of a correlation between TD interest and principal and their various aspects, begs the question whether causality also exists between them. This is especially interesting for third-level aspects of interest and second level aspect of principal. Does complexity, for example, cause understandability rules violations? Does low cohesion and high coupling cause poorly written code? Such a causal study could be designed and performed through a controlled experiment, in which the researchers would control the amount of TD Principal in several variations of a system, and seek the actual maintenance time (TD Interest) for different amounts and types of TD Principal. The necessity (and difficulty) of targeting causation, instead of correlation has been discussed in detail in Dagstuhl 2014 on "Software Development Analytics" [44][45].
- ***Replicate and Generalize***. The results of this study have been obtained by studying well-known and high-quality Java projects. Therefore, there is a need to replicate the case study in other languages and projects of different levels of quality, so as to ensure the generalizability of our results. Similarly, the results need to be confirmed in different programming paradigms (other than object-oriented). We note that such a study, supposing that both extensions are made, would require the addition of two new factors (programming language and paradigm) and an inferential analysis that would target the exploration of the effect of these two factors in the identified relations
- ***Extend the concepts of TD Principal and TD Interest to other types of TD***. By considering that this study limits the calculation of TD to code, we believe that an interesting extension would be towards other types of TD, such as architecture, requirements or documentation TD. For instance, requirements debt could concern costs such as the delay of developing features, whereas architecture debt could involve the impossibility to evolve the system, or the effect on other quality attributes or even on the social aspects of the organization. Nevertheless, given the level of abstraction of these concepts, we see this research work more qualitative (e.g. involving experts) than quantitative.

7 THREATS TO VALIDITY

In this section, we discuss potential threats to the validity of our case study: construct validity, reliability, and external validity [34]. Internal validity is not considered, since causal relations are not in its scope.

7.1 Construct Validity

Construct validity is related to the way in which the selected phenomena are observed and measured. In this study the investigated concepts are *TD Principal* and *TD Interest*. On the one hand *TD Principal* is quantified through SonarQube. SonarQube is the most frequently used tool for measuring *TD Principal* [3], [21], in the sense that is the most widely used in research and practice. Although SonarQube is an established tool, it focuses on code TD, neglecting other types of TD, like architecture debt, requirements debt, etc. According to Martini et al. currently in industry static analyzers (such as SonarQube) are used to analyze the source code in search of TD. Only in few cases out of the respondents in their survey (15 companies) practitioners built their own metrics tools for checking (language-specific) rules or patterns that can warn the developers of the presence of TD [26]. In a similar discussion, Yli-Huumo et al. [40] discuss SonarQube as the mostly used tool for TDM in the eight development teams that they have involved in their case study. Despite the identified limitations, especially in the level of Architectural Technical Debt (ATD), SonarQube is considered as extremely useful for code TD identification, monitoring, measurement and prioritization. Additionally, although SonarQube could be configured to provide more accurate results (e.g., change remediation times), such a practice is not prominent in the literature, where researchers do not perform any re-configuration of the tool [13], and [36].

On the other hand, in the literature there is no established way to measure *TD Interest*. This is due to the fact that an accurate measurement of interest would require the simultaneous maintenance of two software solutions: an optimal and an actual one, and the anticipation of future maintenance activities. Besides the inability to forecast future changes, such an approach is unrealistic for two reasons: (a) there is no way to define a universally accepted optimal system, and (b) it is cost inefficient to maintain two real systems just aiming to accurately measure technical debt interest. According to industrial practitioners, acknowledge that there are no indicators that show the amount of interest paid or predicted if the refactoring. Research prototype tools for interest assessment are not employed in practice yet and should be integrated to provide overall indicators to provide help to the stakeholders to estimate and prioritize TD. Thus, the TD research community shall intensify their work on introducing such tools and indicators [26]. Therefore, as the current state-of-the-art stands *TD Interest* can only be assessed through proxies. In this study, we selected metrics that assess maintainability as a proxy of interest. More specifically, we selected ten object-oriented metrics (grouped in 5 categories/aspects of *TD Interest*) measured at source code, although, in literature, maintainability has been linked to various metrics. Metrics' selection was based on empirical evidence in the literature suggesting that a combination of these metrics is the optimal maintainability predictor [32]. The model for synthesizing the aforementioned values in a unified value for TD Interest relies on solid mathematical calculations, given the assumption that maintenance effort is inversely proportional (linearly) to maintainability. This assumption, although it cannot be validated without a controlled experiment, relies on previous studies [38], [42] and is considered as intuitive by the authors of this paper.

Additionally, we need to note that both *TD Principal* and *TD Interest* are measured at source code level. However, TD is a wider concept that represents inefficiencies at the whole software development lifecycle, and therefore the source code analysis is not comprehensively studying the phenomenon. Thus, our results are not representative of TD holistically as a phenomenon, but only of a subset of it. Nevertheless, code TD is the most studied type of technical debt in the state-of-research [2] and one of the most important in the industry [5]. Finally, with respect to RQ_3, we note that the results heavily rely on the classification schema that we have proposed for Aspects of TD Principal. Although the schema has not been validated and relies on the expert opinion of the senior researchers of this study, it is developed in a systematic way. Thus, also given the fact that such an endeavor could not be conclusive, we believe that it serves the goal of this study, since it is explicitly presented and acknowledges all of its inherent limitations.

7.2 Reliability

With regard to reliability, we consider any possible researchers' bias, during the data collection and data analysis process. The design of the study, concerning data collection, does not contain threats, since all data are automatically extracted by tools, without any subjective configuration. Moreover, with respect to the data analysis process, to mitigate any potential threats to reliability, three researchers were involved in the process, aiming at double checking the work performed and thus reducing the chances of reliability threats. Furthermore, the detailed case study protocol presented in Section 4 enables the repetition of the study, as well as the provision of a replication package[8]. However, we need to note that the clustering of code smells under specific tag categories is subjective and could have been differently performed. Nevertheless, we believe that the clustering is intuitive and forms a well-justified decision.

7.3 External Validity

Concerning external validity, a potential threat to generalization is the possibility that performing the study on different projects of different languages might affect the retrieved correlations. However, we believe that the selected projects, given their size and complexity, represent a realistic real-world system. Additionally, the results of the study are not applicable to non-object-oriented systems, in the sense that *TD Interest* in such systems could not be assessed through properties such as inheritance, coupling and cohesion, which are applicable only in OO software modules. Finally, we note that since the interpretation of the results is based solely on the understanding of the authors on the TD concepts, and not through an additional qualitative study with industrial stakeholders, they cannot be generalized to an industrial context, without additional validation.

8 CONCLUSIONS

This study aims to investigate the interrelation between *TD Principal* and *TD Interest* from two perspectives: (a) to understand the underlying relations between the two concepts, and (b) to provide a way for efficient TD management. To achieve these goals, we have performed a case study on 3600 classes retrieved from 10 Apache projects. The concepts of *TD Principal* and *TD Interest* have been decomposed to multiple aspects that assess different views of the concepts. Given the hierarchical structure of the concepts (*TD Principal* and *TD Interest*) the Mantel test has been used for the examination of their relation and Linear Mixed Effects models for assessing the generalizability of the obtained results. The results of the analysis suggested that *TD Principal* is related to *TD Interest*, and that *TD Principal* is more closely related to the interest aspects of size and coupling, followed by cohesion and complexity. Regarding *TD Principal* aspects, the one that appears to be more strongly interrelated to higher levels of interest is code smells, whereas by further focusing on code smells, we have collected evidence that smells that hinder source

[8] https://se.uom.gr/wp-content/uploads/interest-principal-empirical-relation/replication.zip

code understandability are the ones that are more urgent to resolve in the sense that they are related to higher levels of *TD Interest*. Given the aforementioned outcomes, various implications for researchers and practitioners have been drawn. In particular, regarding practitioners we have suggested a strategy for technical debt prevention, repayment, and prioritization, based on technical debt interest amount.

ACKNOWLEDGMENT

Work reported in this paper has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No. 780572 (project: SDK4ED).

REFERENCES

[1] E. Allman, "Managing technical debt", *Communication*, ACM, 55 (5), pp. 50-55, May 2012.

[2] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study", *Information and Software Technology*, Elsevier, vol 70, pp. 100-121, 2016.

[3] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review", *Information and Software Technology*, Elsevier, vol. 64, pp. 52–73, Aug. 2015.

[4] Ar. Ampatzoglou, Ap. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "Establishing a framework for managing interest in technical debt", *5th International Symposium on Business Modeling and Software Design (BMSD 2015)*, Italy, 2015.

[5] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun and K. Systa, "The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study", *8th International Workshop on Managing Technical Debt (MTD' 16)*, IEEE, Raleigh, NC, USA, Oct. 2016.

[6] Ar. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, Ap. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A framework for managing interest in technical debt: an industrial validation", Proceedings *of the 2018 International Conference on Technical Debt (TechDebt 2018)*, ACM, Gothenburg, Sweeden, pp. 115-124, May 2018.

[7] T. Besker, A. Martini, and J. Bosch, "The Pricey Bill of Technical Debt: When and by Whom will it be Paid?", *33rd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Shangai, China, pp. 13-23, Sept. 2017.

[8] A. Chatzigeorgiou, Ap. Ampatzoglou, Ar. Ampatzoglou, and T. Amanatidis, "Estimating the breaking point for technical debt", *7th International Workshop on Managing Technical Debt (MTD' 15)*, IEEE, Bremen, Germany, pp.53-56, Oct. 2015.

[9] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis", *Transactions on Software Engineering*, IEEE Computer Society, 24 (8), pp. 629-639, Aug. 1998.

[10] J. M. Conejero, R. Rodríguez-Echeverría, J. Hernández, P. J. Clemente, C. Ortiz-Caraballo, E. Jurado, and F. Sánchez-Figueroa, "Early evaluation of technical debt impact on maintainability", *Journal of Systems and Software*, Elsevier, 142, pp. 92-114, 2018.

[11] W. Cunningham, "The WyCash Portfolio Management System", *7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '92)*, 1992

[12] R. J. Eisenberg "A threshold-based approach to technical debt", ACM SIGSOFT Software Engineering Notes, 37 (2), pp. 1 - 6, ACM, 2012.

[13] M. García-Valls, J. Escribano-Barreno, and J. García-Muñoz. "An extensible collaborative framework for monitoring software quality in critical systems", *Information and Software Technology*, Elsevier, 107, pp. 3-17, 2019.

[14] M. Harman, "The current state and future of search-based software engineering.", In *Future of Software Engineering (FOSE'07)*, pp. 342-357. IEEE, 2007.

[15] ISO/IEC 9126-1:2001, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland, 2001.

[16] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt", *37th International Conference on Software Engineering (ICSE)*, IEEE/ACM vol. 2, pp. 179-188, Florence, Italy, May 2015.

[17] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery, "Analogy-X: Providing statistical inference to analogy-based software cost estimation", *Transactions on Software Engineering*, IEEE, 34(4), pp. 471-484, 23 May 2008.

[18] M. V. Kosti, A. Ampatzoglou, A. Chatzigeorgiou, G. Pallas, I. Stamelos, and L. Angelis, "Technical Debt Principal Assessment Through Structural Metrics", *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria, 2017.

[19] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice", *Software*, IEEE, 29 (6), pp. 18-21, 2012.

[20] J. L. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method", *Software*, IEEE 29 (6), pp. 44–51, 2012.

[21] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management", *Journal of Systems and Software*, Elsevier, v. 101, pp. 193-220, March 2015.

[22] W. Li and S. Henry, "Object-oriented metric that predict maintainability", *Journal of Systems and Software*, Elsevier, 23 (2), pp. 111-122, November 1993.

[23] A. MacCormack, and D. J. Sturtevant, "Technical debt and system architecture: The impact of coupling on defect-related activity", *Journal of Systems and Software*, Elsevier, 120, pp. 170-182. 2016.

[24] N. Mantel, "The detection of disease clustering and a generalized regression approach", *Cancer research*, 27 (2), pp.209-220, 1967.

[25] R. Martin, "Agile Software Development, Principles, Patterns, and Practices", *Prentice Hall PTR*, 3rd edition, 2003.

[26] A. Martini, T. Besker, and J. Bosch, "Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations" *Science of Computer Programming*, Elsevier, 163, pp. 42-61, 2018.

[27] S. H. Misra, "Modeling design/coding factors that drive maintainability of software systems", *Software Quality Journal*, Springer 13(3), pp. 297-320, 2005.

[28] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest", *2nd International Workshop on Managing Technical Debt (MTD' 11)*, ACM, pp. 1 - 8, Hawaii, USA, May 2011.

[29] M. O' Keeffe and M. O. Cinnéide, "Search-based refactoring for software maintenance", *Journal of Systems and Software* 81, no. 4 (2008): 502-516.

[30] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, "Multi-criteria code refactoring using search-based software engineering: An industrial case study', *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25(3), pp. 1-53, 2016.

[31] J. Pinheiro and D. Bates, "Mixed-effects models in s and s-plus", Springer Science and Business, 2006.

[32] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics", 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE, Florida, USA, pp. 367-377, 2009.

[33] P. Rovegård, L. Angelis, and C. Wohlin, "An empirical study on views of importance of change impact analysis issues", *Transactions on Software Engineering*, IEEE, 34(4), pp. 516-530, 2008.

[34] P. Runeson, M. Host, A. Rainer and B. Regnell, "*Case Study Research in Software Engineering: Guidelines and Examples*", Wiley, 2012.

[35] K. Schmid, "On the limits of the technical debt metaphor some guidance on going beyond", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 63 - 66, San Francisco, USA, 18 - 26 May 2013.

[36] M. Schnappinger, M.H. Osman, A. Pretschner, and A. Fietzke, "Learning a classifier for prediction of maintainability based on static analysis tools" *Proceedings of the 27th International Conference on Program Comprehension*, IEEE Press, pp. 243-248, 2019.

[37] A. A. Tsintzira, Ar. Ampatzoglou, O. Matei, Ap. Ampatzoglou, A. Chatzigeorgiou, and R. Heb, "Technical Debt Quantification through Metrics: An Industrial Validation", *15th China-Europe International Symposium on Software Engineering Education (CEISEE' 19)*, IEEE TEMS, Lisbon-Caparica, Portugal, May 2019.

[38] C. van Koten and A. Gray, "An application of Bayesian network for predicting object-oriented software maintainability" *Information and Software Technology*, Elsevier, 48 (1), pp. 59-67, 2006.

[39] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, "Identifying and quantifying architectural debt", *38th International Conference on Software Engineering (ICSE)*, IEEE/ACM, pp. 488-498, Austin, TX, USA, May 2016.

[40] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt?–An empirical study" *Journal of Systems and Software*, Elsevier, 120, pp. 195-218, 2016.

[41] N. Zazworka, A. Vetró, C. Izurieta, S. Wong, Y.Cai, C. Seaman and F. Shull, "Comparing four approaches for technical debt identification", *Software Quality Journal*, Springer, 22 (3), pp. 403 – 426, Sept. 2014.

[42] Y. Zhou and H. Leung, "Predicting Object-Oriented Software Maintainability using Multivariate Adaptive Regression Splines", *Journal of Systems and Software*, Elsevier, 80 (8), pp. 1349-1361, 2007.

[43] J. D. Evans, "Straightforward Statistics for the Behavioral Sciences", Brooks / Cole Publishing, Pacific Grove, California, USA, 1996.

[44] M. Di Penta, "Combining quantitative and qualitative methods (when mining software data)", Perspectives on Data Science for Software Engineering, Morgan Kaufmann, pp. 205-211, 2016.

[45] T. Menzies, "Correlation is not causation (or, when not to scream "Eureka!")", Perspectives on Data Science for Software Engineering, Morgan Kaufmann, pp. 327-330, 2016.