Assessing TD Macro-Management: A Nested Modelling Statistical Approach

Nikolaos Nikolaidis[1], Nikolaos Mittas[2], Apostolos Ampatzoglou[1], Elvira-Maria Arvanitou[1], Alexander Chatzigeorgiou[1]

Abstract—Quality improvement can be performed at the: (a) micro-management level: interventions applied at a fine-grained level (e.g., at a class or method level, by applying a refactoring); or (b) macro-management level: interventions applied at a large-scale (e.g., at project level, by using a new framework or imposing a quality gate). By considering that the outcome of any activity can be characterized as the product of *impact* and *scale*, in this paper we aim at exploring the impact of Technical Debt (TD) Macro-Management, whose scale is by definition larger than TD Micro-Management. By considering that TD artifacts reside at the micro-level, the problem calls for a nested model solution; i.e., modeling the structure of the problem: artifacts have some inherent characteristics (e.g., size and complexity), but obey the same project management rules (e.g., quality gates, CI/CD features, etc.). In this paper, we use the Under-Bagging based Generalized Linear Mixed Models approach, to unveil project management activities that are associated with the existence of HIGH_TD artifacts, through an empirical study on 100 open-source projects. The results of the study confirm that micro-management parameters are associated with the probability of a class to be classified as HIGH_TD, but the results can be further improved by controlling some project-level parameters. Based on the findings of our nested analysis, we can advise practitioners on macro-technical debt management approaches (such as "*control the number of commits per day*", "*adopt quality control practices*", and "*separate testing and development teams*") that can significantly reduce the probability of all software artifacts to concentrate HIGH_TD. Although some of these findings are intuitive, this is the first work that delivers empirical quantitative evidence on the relation between TD values and project- or process-level metrics.

Index Terms — Technical Debt, Metrics / Measurement, Quality Analysis and Evaluation, Software maintenance

1 Introduction

Technical Debt (TD) is evident and can be accumulated during every software development activity [1]. Due to its multifaced nature (architecture, code, build TD, etc. [2]) there is a variety of causes that have been related to TD accumulation [3][4]; constituting effective TD prevention or repayment a non-trivial task. According to the literature, upon the identification of high-TD and high-risk artifacts (i.e., high interest amount or probability of paying interest), the software engineer can initiate a refactoring process to reduce the amount of TD [2] in the specific software artifact. For example, in the case of code TD, if a *class* is very *long and undergoes frequent maintenance*, the *extract class* refactoring can be applied, to lead to more maintainable code—i.e., producing less TD interest [5].

We consider such repayment activities which are focused on design hotspots and the impact of the solutions is local—as ***TD Micro-Management***. Additionally, these solutions are *a-posteriori* ones, i.e., the system has suffered from TD consequences for some time, and afterwards some reactive measures are taken. On the other hand, by studying the causes that can lead to TD accumulation (e.g., *there is no adoption of quality control and monitoring tools*), one could identify proactive management practices, which can be considered *a-priori* and prevent the accumulation of TD [3]. We consider activities which: (a) have a global impact (all artifacts of the system are affected and (b) prevent TD accumulation as ***TD Macro-Management***.

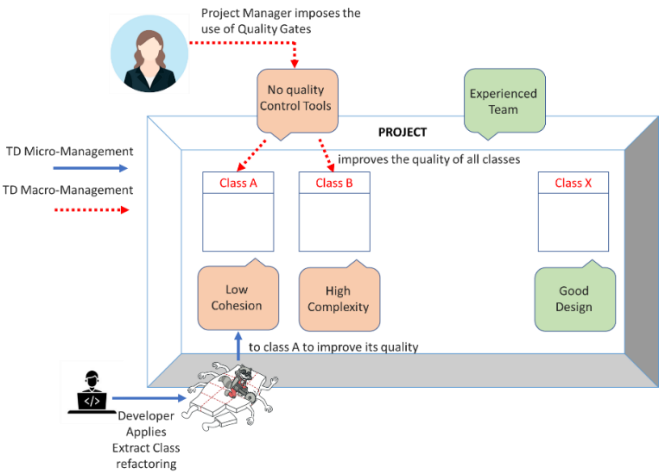

FIG. 1: CONTEXT DESCRIPTION & MOTIVATION

Figure 1 visualizes this context. Suppose a software project that comprises of X classes. At the project level, we visualize two possible causes of TD: "*No quality control tools*" that is not satisfied by current practices (orange bubble) and "*Having an Experienced Team*" which holds (green bubble). At the class level, we suppose that Class A

- *Nikolaos Nikolaidis is with the Department of Applied Informatics, University of Macedonia, Greece. E-mail: nnikolaidis@uom.edu.gr*
- *Nikolaos Mittas is with the Department of Chemistry, International Hellenic University, Greece. E-mail: nmittas@chem.ihu.gr*
- *Apostolos Ampatzoglou is with the Department of Applied Informatics, University of Macedonia, Greece. E-mail: a.ampatzoglou@uom.edu.gr*
- *Elvira-Maria Arvanitou is with the Department of Applied Informatics, University of Macedonia, Greece. E-mail: e.arvanitou@uom.edu.gr*
- *Alexander Chatzigeorgiou is with the Department of Applied Informatics, University of Macedonia, Greece. E-mail: achat@uom.edu.gr*

suffers from "*Low Cohesion*" (orange bubble), Class B suffers from "*High Coupling*" (orange bubble), whereas Class X is of "*Good Quality*" (green bubble). On the one hand, working with a ***TD Micro-Management*** technique (blue continuous arrow), a developer can resolve the TD item of Class A, by applying the "*Extract Class*" refactoring—however, the other classes are not affected. On the other hand, the project manager can impose the use of "*Quality Gates*" and affect the code quality of all classes of the system—applying ***TD Macro-Management***. Given the aforementioned context, in this work, we aim at answering the following general research question: "*Which combination of TD Micro- and Macro-Management methods can be the most efficient for preventing the accumulation of TD*"?

As a first step in studying TD Macro-Management, in this work, we quantitively explore the relation between a pre-determined list of TD causes [3] and TD accumulation, through an empirical study on 100 open-source software projects. We note that HIGH_TD artifacts are extracted from a TD Benchmarker, identifying the agreed HIGH_TD artifacts, as indicated by three top TDM tools[1]: SonarQube, CAST, and Squore [6]. In other words, artifacts are considered as HIGH_TD ones, when there is a consensus by three tools that their level of TD places them among the most problematic artifacts. According to the threshold set in the TD Benchmarker [6], only 12% of the artifacts in the benchmark are characterized as such, while the rest are NOT_HIGH_TD artifacts, which do not raise any alarms about the level of TD. Out of the most prevalent causes of TD, we are able to ***state management guidelines that can be useful for practitioners to prevent the accumulation of TD in all software artifacts***. However, identifying the most prevalent causes of TD accumulation is challenging. Solving this problem yields for a novel methodological / analysis approach, for two reasons:

- ***nested nature of the problem***. By definition TD Macro-Management parameters (i.e., project-level metrics) have equal values for all the artifacts of the same project, whereas TD Micro-Management parameters (e.g., class metrics) have different values for every artifact. In other words, units of analysis (classes) are nested into contextual/aggregate units (projects) resulting in multi-level data. In that sense, traditional statistical analysis would fail and a multi-level statistical modeling approach would be required.
- ***need for identifying project-level activities that can prevent the accumulation of TD***. Black-box machine learning approaches that produce accurate models for predicting if an artifact has accumulated high levels of TD or not (given the values of TD Micro- and Macro-management parameters) are not suitable. The reason for this is that Machine Learning models suffer from low interpretability [7], whereas the goal of the study is to explore the TD Micro- and Macro-Management parameters and identify the most prevalent ones.

By considering both the need for a multi-level statistical approach, as well as the need for reaching an explainable model, we propose the use of Generalized Linear Mixed Models (GLMMs) models; i.e., a branch of statistical techniques that uniformly models fixed and random effects (***Fixed***: TD Macro- and Micro-Management parameters, ***Random***: the variance attributed to the nested design (classes are nested within projects).

Given the above, and by considering the main research question of this work, as well as the required steps to achieve it, the main contributions of this study are:

- ***the compilation of a list of TD Micro- and Macro-management parameters***, by studying the literature on causes of TD accumulation—we note that this is the first study that attempts to map causes of TD accumulation to specific metrics;
- ***a dataset of values of these parameters for 100 open-source software projects***, extracted by mining software repositories (GitHub & Jira)—this dataset is the first one in the TD community that involves project-level measurements of product and process quality;
- the ***introduction of an explanatory data analysis methodology*** that is able to ***handle the nested nature of the software***—to the best of our knowledge, despite the indisputable nested nature of software systems, this the first time that a nested solution is used;
- ***the provision of empirical quantitative evidence***, based on a large-scale dataset, that can ***unveil the most important management activities for TD prevention***—this is the first study that brings evidence on the relevance of project management level parameters that can be used to prevent TD accumulation.

2 Related Work

In this section, we present related work and background information required to facilitate the understanding of our work. First, we present related work, i.e., studies that aim at performing TD items identification (see Section 2.1); and in Section 2.2, we present the main causes of Technical Debt accumulation that will set the basis for defining our metric-based approach.

2.1 Technical Debt Identification

In the literature, we have identified two systematic mapping studies related to *TD identification*—i.e., the process of understanding which artifacts suffer from TD. First, Alves et al. [8] collected and investigated approximately 100 studies. They highlight several TD indicators that can be used to identify the amount of TD of a system. Some of the top mentioned indicators are "code smells", "software architecture issues", "automatic static analysis issues", etc. Additionally, in this study, we observe that almost all the papers use source code artifacts to identify TD items (e.g., even for architectural TD), most probably because of the tools that exist for TD measurement. Additionally, Ben Idris et al. [9] examined 43 empirical studies, published in a period of 4 years. They suggest that the most

[1] We note that while the use of three different tools in a development process is theoretically feasible, we believe that this would hardly be possible in practice. Employing more than one tool would be very costly, since most of the existing tools are commercial ones, and require significant effort to deploy, configure and familiarize with. Even if a development team employs more than one tool, the identification of HIGH_TD classes does not simply consist in finding the union of all findings (leading to an unrealistic number of suggestions, rendering the process intractable), but on archetypal analysis (see [6]).

common TD indicator is "code smells", while the second most commonly used indicators are "code comments" and "Defect/Bugs". Finally, regarding the identification of TD items, the authors highlight the variety of available tools, denoting as the most commonly used: SonarQube.

This strategy leaves an important gap in the TD identification, since the project-level is completely neglected.

By focusing one some striking primary studies out of these works (we cannot focus on all, since the literature is vast), Zazworka et al. [10] tried to evaluate the TD items that different stakeholders report and the differences with three automated approaches for the TD identification. From this research, we can observe that there is not a lot of overlap with the TD items that the tools and the stakeholders reported. This points out the importance of the role of the developers in the identification of TD, and how different stakeholders know and consider different debts in their projects. Finally, Zazworka et al [11] analyzed multiple versions of Apache Hadoop with four different TD identification techniques namely, modularity violations, grime buildup, code smells, and automatic static analysis. They found out that there is a limited degree of overlapping, when using the different approaches: different techniques tend to point to different classes and therefore to different problems.

There is a need for multiple TD indicators, especially in the non-source code part of a project, because to the best of our knowledge this option has not been studied yet.

2.2 Causes of Technical Debt

In this section, we present the most common causes of TD based on the literature. The four studies that are dealing with identifying possible causes of TD accumulation are outlined below. Martini et al. [4] performed a multiple-case embedded study in seven sites at five large organizations to investigate the current causes for the accumulation of architectural TD (ATD). As a result of this study, the authors provided a taxonomy of causes and their influence in the accumulation of ATD. Martini and Bosch [12] conducted a case study to investigate: (a) the most dangerous ATD items in terms of effort paid later; (b) the effects triggered by such ATD items; and (c) if there are sociotechnical patterns of events that trigger the creation of ATD. The results suggested that TD items can be contagious, causing other parts of the system to be contaminated with the same problem, which may lead to nonlinear growth of interest. The authors also presented a model of ATD effects for TD repayment prioritization.

Yli-Huumo et al. [13] performed a case study to investigate the role of technical debt management in software development. In particular, the goal of this study was to explore the causes of TD accumulation, as well as its effects, and the strategies that are being used for technical debt management. The results of this study suggested that the reasons for incurring TD were management decisions that were made in order to reach deadlines, or unknowingly due to lack of knowledge. Finally, as a more recent work in this area, Rios et al. [3] conducted an industrial survey in different countries in order to investigate the trends in the TD area including the causes and the effects of TD. The survey design reached a large number of targeted responded: eventually 107 practitioners from 11 countries have reacted positively to the survey. The results of this study suggested that most of the practitioners were familiar with the concept of TD. As a final outcome, Rios et al. [3] identified 78 causes that lead to TD occurrence. Out of them, we focus on the most cited causes that lead to the accumulation of TD. Based on Rios et al. [3], the top-14 most cited causes of TD referring to TD Macro-Management are:

- ***Change in Project*** — TD can be accumulated due to the need for continuous changes in the project — the need for velocity will lead to neglecting the quality of produced code. Usually, large volumes of change bring more pressure. *Example*: "Constant request for changes in the project".
- ***Close Deadlines*** — Having close and unrealistic deadlines can lead to promoting quick and dirty implementation solutions, increasing TD. Close deadlines are usually accompanied by frequent releases. *Example*: "The rush of managers (customers) that want to receive something working asap".
- ***Dependencies to External Components*** — TD accumulation can occur when the project depends on an external component; either due to the quality of external components or due to less control over the external project. *Example*: "The fact that Angular 2 functionality is not yet stable, even with a deadline to fix the bugs".
- ***Inaccurate or Complex Requirement*** — TD can occur due to failure, lack of clarification, complexity or poor definition in the collected requirements. Inaccurate requirements can lead to many changes (interest probability), whereas complex requirements will lead to more complex code. *Example*: "Lack of clarification of requirements".
- ***Inappropriate Testing*** — A project that is poorly tested, or even when the tests were poorly planned or do not have good coverage, can lead to more corrective maintenance requests; thereof, more TD accumulation. *Example*: "Lack of testing";
- ***Lack of Interest in Acquiring Knowledge*** — Refers to the lack of interest of the team in seeking knowledge to develop new skills. Getting stuck with old technologies, might lead to unnecessary complexity that might be resolved with new technologies and skills. *Example*: "Lack of interest and willingness of the team to acquire knowledge".
- ***Lack of Specific Teams*** — Occurs when there is no specific team to perform any software process activity, such as test team, development team, maintenance team, documentation team, etc. In practice, having the code being tested or reviewed with a "fresh look" by a different group would lead to more accurate TD identification. *Example:* "Lack of a separate testing team".
- ***Lack of Team Communication*** — TD can occur when there is communication problem between team mem-

bers. Lack of communication might lead to more effort to understand code, and additional introduction of bugs. *Example*: "Lack of integration of business areas, analysts of systems, developers and the test area".

- ***No Adoption of Good Practices*** — TD can be accumulated, when there is no use of good practices (e.g., patterns, refactorings, quality gates, etc.) that would facilitate the accomplishment and maintenance of activities in the project. *Example*: "Employment of bad design practices".
- ***No Awareness of the Importance of Testing & Refactoring*** — TD can occur when the team do not recognize the importance of documenting, refactoring, and testing the software. *Example*: "To think that certain tasks (tests, refactoring) are not important".
- ***No Focus on Documentation*** — When a company has no culture on documentation and thus does not recognize its importance will probably accumulate more TD, since the effort to understand the codebase will be higher. *Example*: "The manager's vision that documentation is non-productive time".
- ***Pressure*** — TD accumulation occurs when there is high pressure on team members to meet deadlines and speed deliveries — this usually drifts the focus from quality improvement. *Example*: "Pressure to meet short deadlines".
- ***Structural Change in Organization*** — When companies undergo structural changes, the development process might be altered, leading to producing code of poor or deteriorated quality. *Example*: "Changes in the structure of a company lead to postponement of refactoring sessions".
- ***Team Overload*** — When the development team has accumulated activities, either because of a lack of adequate management or because members have left the team, will lead to time pressure, leading to TD accumulation. *Example*: "The accumulation of activities did not allow a loophole for extensive refactoring, so this was being pushed to the end of the project".

In addition, several other causes that can be related to TD Micro-Management parameters have been identified:

- ***Poor Documentation*** — Refers to the lack of any kind of documentation (from documents to comments). *Example*: "Lack of code documentation".
- ***Experience of Contributors*** — Refers to the experience of the developers in specific software activities. *Example*: "Lack of experience of programmers".
- ***Poor Design*** — Refers to poorly designed projects, with bad quality metrics. *Example*: "Poorly designed class with high cohesion or coupling".

3 METHODOLOGY

In this section, we describe the methodology that enables the analysis, given the nested design of the problem. In Section 3.1, we present the necessary background information for enabling the understanding of the proposed analysis; in Section 3.2 the data collection process; whereas in Section 3.3 the proposed data analysis methodology.

3.1 Methodology Overview

In this section, we present background information necessary for facilitating the understanding of the proposed data analysis methodology (see Section 3.3). To this regard, we discuss the inherent challenges posed by the nature of the problem under investigation and the data collected for empirically assessing TD Micro- and Macro-Management opportunities, as well as, the advanced statistical and ML approaches adopted in a unified methodology to overcome these hurdles.

The first challenge concerns the research design of the experimental setup, in which the units of analysis (i.e., artifacts) are organized as a two-level hierarchy, representing: (a) TD Macro-Management (or project-level) parameters; and (b) TD Micro-Management (or class-level) parameters. The hierarchical structure of the data imposes the need for the application of appropriate statistical procedures, since class-level measurements, corresponding to the classes of the same project are expected to be more similar, compared to classes of a different project, due to the fact that they share common characteristics. To this regard, traditional statistical approaches ignore dependencies within grouped data or they lack sophisticated mechanisms able to take into consideration the multi-level structure of the data leading, in turn, to statistical validity issues and erroneous decision-making.

Due to the above reasons, we decided to make use of an advanced modeling technique belonging to the general branch of *Mixed Effects Models* (MEMs) [14] that are able to deal with both the hierarchical structure of the data and the random variation associated with sampling higher-level units, projects in our case. Described briefly, MEMs investigate simultaneously two types of effects: (a) the *fixed* and (b) the *random* effects. The former type of effects (fixed effects) is associated to factors affecting the mean value of the response and they are constant (or fixed) across all observations. In simple words, fixed effects can be thought as factors of primary interest from the practitioner point of view that deserve a thorough investigation to understand how each factor may affect the changes in the response. In contrast, random effects are associated to the sources of variance in an experimental setup modeling how the total variation is decomposed into different variance components. Although random effects may be used in a wide range of experimental designs (e.g., longitudinal data, repeated measures etc.) for providing insights about the variability at different levels of interest, in our case, this type of effects takes into consideration the two-level data structure of the experimental units that is *classes are nested within projects*. Given the nature of the response (a dichotomous variable — $\boldsymbol{I}_i$) indicating if a TD artifact is characterized by the TD Benchmarker tool as HIGH_TD or NOT_HIGH_TD, we decided to make use of the *Generalized Linear Mixed Models* (GLMMs) with the *logit link function*[2].

The second challenge stems from the sparsity of HIGH_TD classes. According to Amanatidis et al. [6], the

[2] $g(\cdot) = ln(p/(1-p))$, where p is the probability of an artifact to be identified as having accumulated high levels of TD

level of agreement for classes that are identified as problematic by all three used tools is 7.7% (for the examined dataset in their study). This finding is considered intuitive in the sense that different TD measurement tools rely on different rulesets for calculating whether a class suffers from TD or not. Indeed, in our sample, the investigation of the distribution of the response variable (an expected dichotomous variable) indicates that the HIGH_TD classes are under-presented (6.3% of classes) compared to NOT_HIGH_TD classes, leading to a well-known phenomenon in ML, namely the *class imbalance problem*. The highly-skewed response distribution poses significant barriers to statistical and ML algorithms leading to poor performances, especially for the minority class, since most of these approaches are developed under the assumption of equal sample sizes for both levels of the response.

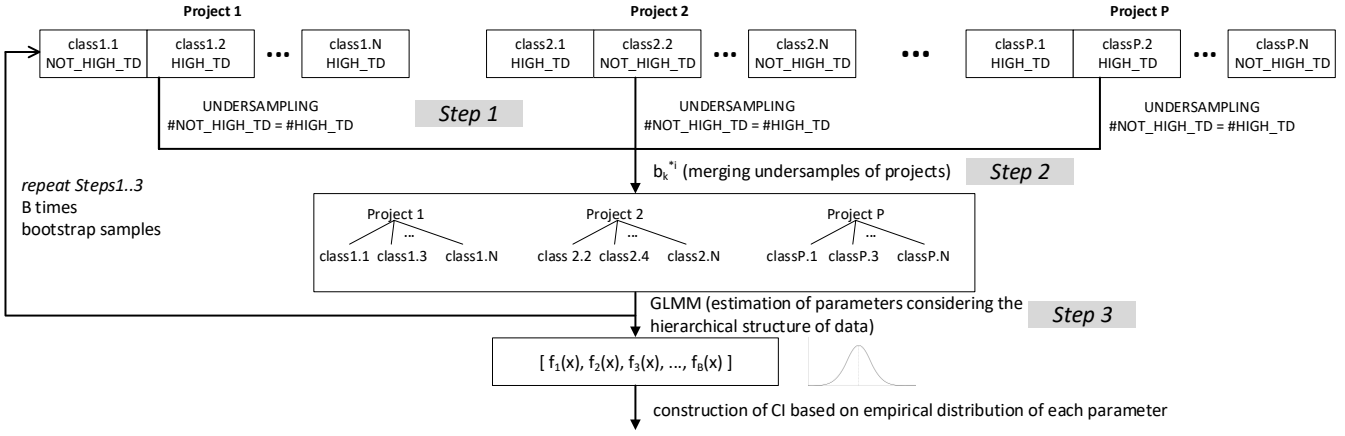

FIG. 2. METHODOLOGICAL FRAMEWORK

To alleviate this inherent limitation for the vast majority of approaches to deal with the class imbalance problem, we designed a simulation resampling technique, named: *Under-Bagging based GLMM* (UBGLMM) [15], combining the merits of: (a) *bootstrap aggregating* (*bagging*) [16], a well-known type of *ensemble learning*; and (b) *random under-sampling* [17]. The general idea behind the proposed approach is the building of an ensemble consisting of a large number (B) of GLMMs fitted on random subsamples drawn independently from the original dataset following an under-sampling strategy, which aims at balancing the class distribution of the response through the random elimination of cases belonging to the majority class. The whole framework along with its main components and the necessary steps are summarized in Figure 2.

In the first step of the approach (Step 1), each project (p_i), from the collection of the total P projects, is subjected to a random under-sampling process for balancing the number of NOT_HIGH_TD classes to the number of HIGH_TD classes in order to mitigate the class imbalance problem. The set of P projects consisting of an equal number of NOT_HIGH_TD and HIGH_TD classes are merged into a unified dataset with TD Macro-Management (project-level) parameters and TD Micro-Management (class-level) parameters (Step 2). The merged dataset constitutes the basis for the fitting of a GLMM taking into consideration both fixed and random effects, so as to appropriately handle the hierarchical nature of the experimental data (Step 3). This step, essentially, results into the estimated coefficients of the GLMM providing to us a straightforward inference on the fixed effects of both TD Macro- and Micro-Management parameters on the response (NOT_HIGH_TD / HIGH_TD classes) and random effects that is the variance decomposition into project- and class-levels.

Algorithm 1: Pseudo code of the proposed methodology

```
Data: Number of repetitions: B, Project List: pList
for (Project p: pList)
    C1.add(p.getHighTDClasses) // C1: Minority class
    C2.add(p.getNotHighTDClasses) // C2: Majority class
end
for (i=0; i<B; i++)
    Balanced_List.clear()
    for (k=0; k<C1.size(); k++)
        p = C2.get(random)
        Balanced_List .add(p)
    end
    Balanced_List.add(C1)
    GLMM.fit(Balanced_List)
end
```

Steps 1-3 are repeated for a large number of times (B), leading, in turn, to the formation of an empirical distribution b_k^{*i} $(i = 1, \dots B)$ for the set of K parameters (class- and project-level metrics), which can be used to construct $(1-a)\%$ *under-bagged percentile confidence intervals* (UBCIs), inferring the fixed effects of TD Micro- and Macro-Management parameters on the response (Step 4). More specifically, the $(1-a)\%$ UBCI for the k parameter based on the B estimates derived from UBGLMM approach can be evaluated through the following formula:

$$\left[b^*_{k_{a/2}}, b^*_{k_{(1-a/2)}}\right] \quad (1)$$

where, a represents the prespecified significance level, whereas $b^*_{k_{a/2}}$ and $b^*_{k_{(1-a/2)}}$ are the lower and upper limits corresponding to the $100(a/2)$-th and the $100(1-a/2)$-th percentiles of the empirical distribution, respectively.

Last but not least, the empirical distribution of the under-bagged estimates b_k^{*i} $(i = 1, \dots B)$ can be also used for calculating an approximation of the unknown population parameter β_k via the following equation

$$b_k^{UB} \frac{1}{B} = \sum_{i=1}^{B} b_k^{*i} \quad (2)$$

To highlight the above process related to the construction of the UBCI (Eq. (1)) and the estimates of the examined parameters (Eq. (2)), we, indicatively, demonstrate an example related to the examination of the *No Adoption of Good Practices* Macro-Management metric. Following the above under-sampling strategy and the fitting of a UBGLMM model for $B = 1000$ repetitions (Steps 1-3), the process resulted into 1000 under-bagged estimates for the *No Adoption of Good Practices* parameter. The set of $B = 1000$ under-bagged estimates of the fitted models' forms, in turn, an empirical distribution that is graphically displayed in Figure 3. Hence, the evaluation of the 95% UBCI (lower and upper bounds represented by the black vertical lines, respectively) is a straightforward task through the identification of the 2.5-th and the 97.5-th percentile values of the constructed empirical distribution (see also the results in the second column of Table 2 in Section 2). Moreover, the mean value of this empirical distribution (represented by the red line) is the final under-bagged estimate (Eq. (2)) concerning the coefficient of the *No Adoption of Good Practices* Macro-Management TD metric of the model.

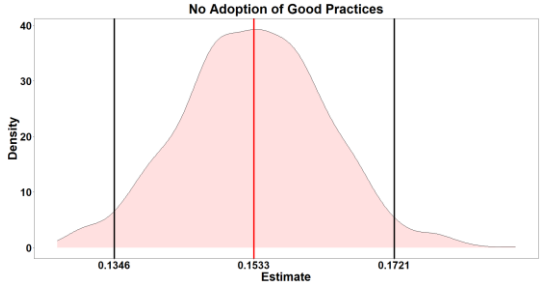

FIG. 3: EXAMPLE DISTRIBUTION

3.2 Data Collection

For the purpose of this study, we have analyzed 100 open-source software projects. To have a common source of information for project retrieval and to be sure that all of them are being developed using the same techniques, we selected the Apache Software Foundation organization from GitHub. To be able to calculate all the selected metrics for each project, we have applied the following two restrictions in the project selection process. First of all, the projects must be developed in the Java programming language, to be able to calculate the majority of the TD Micro-Management parameters. Secondly, the projects should also rely on Jira as an issue tracking system. This was vital for the calculation of some TD Marco-Management parameters. The final list of selected projects can be found in Appendix A, along with some descriptive characteristics.

To be able to calculate all of the TD Micro- and Macro-Management Metrics, we had to use a variety of existing tools, or create our own. On the one hand, concerning ***TD Micro-Management***, we use established class-level metrics used for assessing software assets maintainability—see first part of Table 1. The selection of metrics relied on a secondary study by Riaz et al. [34], suggesting that the metrics suites of CK and Li and Henry are the most commonly used maintainability predictors. On top of that, regarding developers experience per class we relied on the metric by Tsoukalas et al. [25], and for "Poor Design" we used TD Interest [21], as the most relevant for the TD phenomenon. The metrics are calculated relying on four different tools: CKJM [18], Metrics Calculator[3] [19], PyDriller [20], and SDK4ED Interest Calculator [21].

On the other-hand, to the best of our knowledge, there are neither metrics, nor tools, for assessing the ***TD Macro-Management*** parameters. For this reason, we had to define our own metrics for each TD cause, described in Section 2.2, and are presented in the second part of Table 1, along with their calculation method. The presentation of the TD Macro-Management metrics follows the order of TD project-level causes (see Section 2.2). For example, the cause "*Change in Project*" has led us to consider a metric: "*Average LoC changes between commits*", or the "*Close Deadlines*" TD accumulation cause has inspired us for studying the: "*Average days between releases*" metric. Nevertheless, we need to clarify that, since for each TD accumulation cause, a number of different metrics could have been introduced, the obtained results are coupled to the defined metrics and not to the TD causes that have been used as an inspiration. Therefore, any interpretations stand closer to the metrics rather than the corresponding cause. Despite the employed interpretation strategy to avoid severe construct validity threats, in Section 6, we further discuss the possible complications more extensively.

The dichotomous response variable expresses if a class of the dataset is classified as HIGH_TD or not. The assessment is performed with a tool[4] based on Machine Learning classifiers that uses source code metrics and repository activity information as predictors [33], validated with a large number of open-source software classes [25]. As ground truth for the development of the employed classification framework an empirical benchmark of classes that exhibit high levels of TD was used [6], based on the convergence of three widely adopted TD assessment tools, namely SonarQube, CAST, and Squore. In other words, a class that is classified as HIGH_TD corresponds to an artifact which would probably be identified as problematic by three leading TD analysis tools, thereby expressing a commonly agree TD item.

3.3 Data Analysis

3.3.1 Data Pre-processing

The dataset of this study, encompasses artifacts assessed as HIGH / NOT_HIGH_TD by the TD Benchmarker tool [25] along with the class- and project-level metrics evaluated by the examined tools. The initial dataset comprised 90,669 classes and their associated measurements at both TD Macro- and Micro-Management.

[3] https://github.com/dimizisis/metrics_calculator
[4] http://160.40.52.130:3000/tdclassifier

TABLE 1: METRICS USED FOR THE TD MICRO- AND MACRO-MANAGEMENT

	Metric	Calculation method / Description	*M*	*SD*	*Mdn*	*Min*	*Max*
Micro-TDM	Coupling Between Objects (CBO)	Number of dependencies a file has	9.22	8.09	7	0	43
	Weighted Method per Class (WMC)	Counts the number of branch instructions in a class	15.23	21.15	8	0	325
	Depth of Inheritance Tree (DIT)	Counts the number of "fathers" a class has	2.03	1.89	1	1	144
	Response for a Class (RFC)	Counts the number of unique method invocations in a class	17.49	22.41	10	0	561
	Lack of Cohesion of Methods (LCOM)	Subtracts the number of cohesive from the non-cohesive pairs of methods	43.06	172.71	3	0	3257
	Lines of Code (LoC)	Counts the lines of code	77.64	108.12	41	1	3876
	Data Abstraction Coupling (DAC)	Counts the number of user-defined classes as class properties	0.34	0.93	0	0	9
	Message Passing Coupling (MPC)	Total number of methods called	20.91	35.97	6	0	276
	Number of Classes (NoC)	Measures the number of immediate descendants of the class	0.48	3.97	0	0	365
	Complexity	Counts the amount of decision logic in a source code	11.50	21.76	2	0	188
	Documentation	Percentage of comment lines in comparison with the source code lines	11.21	17.32	2.10	0	97
	Experience of Contributors	Percentage of the lines authored by the highest contributor to a class	88.29	17.81	100	16.12	100
	Poor Design / TD Interest	Amount of money that has to be paid as an overhead when changes are being made, due to the code quality of the codebase.	0.91	2.26	0.26	0	180.31
Macro-TDM	Changed Lines	The average changed lines of code per commit	470.96	334.66	399.28	137.89	2316.58
	Release Distance	Days that passed between each release in Github	173.04	120.44	139.74	4.33	515.82
	Number of Dependencies	Number of dependencies of the project.	49.78	57.23	31.5	0	251
	Jira Issues Size	Average size (in words) of the Jira issues description	98.18	54.4	85.68	32.42	284.1
	Test Coverage (%)	Test coverage as a percentage of the functions covered by test by the total number of functions	0.24	0.33	0.14	0.01	1.91
	Commits since a new technology is added	Number of commits that pass since the addition of a new technology (obtained by counting the file extension)	419.46	334.31	316.42	98	1533
	Distinction of Development & Testing	Check if there is an intersection between team members reporting and resolving issues in Jira	No: 33,867 (47.42%)		Yes: 37,552 (52.58%)		
	Emails among team members	Average number of emails the developers exchanged from the official mail-archives of Apache	206.47	179.19	140.22	13.25	714.8
	Code inefficiencies	Average number of code inefficiencies (with the help of Checkstyle tool) for each line of code	1.15	1.58	0.7	0.06	11.37
	Number of Refactoring	Average number of refactoring that appear in a commit (based on Refactoring Miner)	5.36	9.02	3.12	0.66	61.84
	Size of Javadoc's	Average size of each list item in the Javadoc	9.73	5.72	7.92	1	23.56
	Commits per Day	Average number of commits per day	1.77	1.97	1.01	0.24	10.57
	Changes in Board Meetings	Changes of the members in the official board meetings of Apache	3.77	16.67	1.51	0.68	128.44
	Number of Members	Number of official team members of a project	19.98	11.73	17.5	4	57

Note *M, SD, Mdn, min, max* represent the mean, standard deviation, median, minimum, maximum values, respectively

The descriptive statistics and the graphical inspection of the measurements for the set of independent variables (class- and project-level metrics) indicated highly-skewed distributions due to the existence of extreme outlying points. For this reason, we decided to identify and filter out these outliers through the evaluation of the *z*-scores for each class-level metric using a threshold value of 3.0, since they will strongly affect the fitting process of the final model. This pre-processing step resulted into the removal of 2,670 classes. Finally, the limited number of HIGH_TD classes within specific projects led us to exclude projects presenting less than 20 HIGH_TD classes. We decided to exclude from any further analysis projects containing only a limited number of HIGH_TD classes, since they would not be representative cases from the unknown population that we wish to infer about. For example, *clerezza* and *commons-io* accumulated only seven HIGH_TD classes, and thus, any inference about the random effects through the insertion of these projects into the analysis would be covered by a significant amount of variability. Applying the aforementioned decisions, yielded a final dataset consisting of 71,419 classes from 58 Java projects. Out of those 3,675 (5.15%) were characterized as HIGH_TD (response variable).

3.3.2 Model Building

Regarding the building of the GLMMs, the fitting phase was based on a two-step process for the selection of the set of the independent variables that will form the TD Micro- and Macro-Management parameters, corresponding to class- and project-level metrics, respectively. As far as TD Micro-Management parameters are concerned, we followed a feature selection strategy based on predefined criteria synthesized via the careful examination of the class-level metrics and the interpretation / calculation formula for each metric. Such an approach is aligned with the general guidelines for selecting fixed effect factors in MEMs suggesting the insertion of independent variables obtained from the researchers' expertise and knowledge and the understanding of the problem rather than a naïve approach that takes into consideration the whole set of the collected features [22]. The selection of class-level metrics was conducted satisfying the following criteria:

C1. Class-metrics that are linear combinations of other class-metrics should be excluded. This criterion has been set due to limitations of statistical methods that use the covariance matrix and related problems (e.g., problem of ranks, irreversibility, etc.);

C2. Class-metrics that are highly correlated to other class-metrics, i.e., presenting a statistically significant Spearman correlation coefficient higher than 0.6 should be excluded. This criterion has been set since multicollinearity causes problems in statistical methodologies; and

C3. Class-metrics that are not highly correlated with the response, i.e., Spearman correlation coefficient lower than 0.1 should be excluded. This criterion has been set due to the fact that an independent variable that is very weakly correlated to the response variable can offer very limited interpretability.

The adaption of the above selection strategy resulted into a set of five metrics (*CBO*, *LCOM*, *Complexity*, *Experience of Contributors*, *MPC*) that will be used into the Micro-Management component of the model. Due to C1 we excluded *RFC*, since its calculation involves both *MPC* and *WMC*. Due to C2, we have excluded *LoC*, which is very strongly correlated to *CBO*, *WMC*, *MPC*, and *TD Interest*. Also, we excluded *Focus on Producing more at the Expense of Quality*, which is correlated to *TD Interest; WMC* which is correlated to *LCOM*, and *DAC* being correlated to *CBO*. Finally, in the dilemma of excluding either *TD Interest* or *MPC*, we preferred to retain *MPC* since it is a primary metric, whereas *TD Interest* is a compound one. Based on C3, we excluded *DIT*, *NOCC*, and *Documentation*.

Concerning the Macro-Management part of the model, we made use of a forward selection strategy, since, to the best of our knowledge, this is the first research attempt focusing on the investigation of the potential effects for an extensive set of Macro-Management parameters on TD accumulation. Thus, there is a relatively limited body of knowledge and empirical evidence that could guide the process. Described briefly, the model incorporating the class-level metrics (TD Micro-Management parameter component) is fitted, while in each iteration, the algorithm adds in project-level metrics belonging to the TD Macro-Management parameters list one by one. The TD Macro-Management parameter selected for an entry into each iteration is the one leading to a model that presents a statistically significant difference compared to the model of the previous iteration based on the *Wald* test, whereas among the candidates of statistically significant project-level metrics, the metric resulting to the model with the lowest *Akaike Information Criterion* (AIC) value is finally selected. The process is iteratively repeated until there is no further improvement by the insertion of project-level metrics from the set of candidates that do not participate in derived solution of the previous iterations. The execution of the forward selection algorithm revealed a set of three project-level metrics (*pressure, non-adoption of good practices,* and *lack of specific team*) that will be inserted into the TD Macro-Management part of the model along with the five metrics of the TD Micro-Management part.

4 Experimental Results

This section presents the findings extracted from the proposed multi-level statistical approach. Table 2 summarizes the results obtained from the execution of the UB-GLMM algorithm on the examined dataset. The second column ("UB Estimate") presents the estimated coefficients by which each metric participates into either the micro- or macro-management parts of the final model along with the corresponding 95% UBCIs.

In addition, Figure 4 visualizes the empirical under-bagged distributions for each class- and project-level metric obtained from $B = 100$ repetitions of the UBGLMM algorithm, whereas the vertical red line indicates the zero value for facilitating hypothesis testing. In our case, the findings (Table 2 and Figure 4) reveal that the null hypothesized value (the zero value) falls outside of the constructed 95% UBCIs for all class- and project-level met-

rics, which practically means, that the identified TD Micro- and Macro-Management activities present statistically significant fixed effects on the response.

TABLE 2: RESULTS OF UBGLMM ALGORITHM

Metric	UB Estimate 95% UBCI	OR 95% CI for OR
CBO	0.1429 [0.1346, 0.1512]	1.1536 [1.1441, 1.1632]
LCOM	0.0014 [0.0009, 0.0020]	1.0014 [1.0009, 1.0020]
Complexity	0.0384 [0.0360 ,0.0410]	1.0391 [1.0367, 1.0419]
Contributors Experience	-0.0081 [-0.0108, -0.0053]	0.9919 [0.9893, 0.9947]
MPC	0.0165 [0.0148, 0.082]	1.0166 [1.0149, 1.0184]
Pressure	-0.1825 [-0.2066, -0.1605]	0.8332 [0.8133, 0.8517]
No Adoption of Good Practices	0.1533 [0.1346, 0.1721]	1.1657 [1.1441, 1.1878]
Lack of Specific Teams: Yes	-0.3283 [-0.4207, -0.2446]	0.7201 [0.6566, 0.7830]

To provide an intuitive and straightforward interpretation of the effects for the final set of Micro- and Macro-Management parameters on TD accumulation, we computed also the *Odds Ratio* (OR) (3rd column, first line of Table 2) and their associated 95% CIs (3rd column, second line of Table 2) for the class-level and project-level metrics, respectively, which quantifies the ratio of the probability of a TD-artifact to be assessed as HIGH_TD to the probability of being assessed as NOT_HIGH_TD. Generally, the larger the OR, the higher odds that the class will be assessed as HIGH_TD, whereas, odds ratios smaller than one indicate the TD-artifact has fewer odds of being assessed as HIGH_TD. For example, the value of OR for the *CBO* metric indicates that as the number of dependencies for a class file increase by one unit, the odds for the class to be assessed as HIGH_TD increases by a factor of 1.1536, which practically means an increase of 15.36%. Regarding the interpretation of the OR for the categorical project-level metric *Lack of Specific Team (i.e., No Distinction among Development and Testing Team)*, the value is lower than one, signifying that a TD-artifact to be assessed as HIGH_TD is more likely at level *No* (i.e., when there is no distinction between the Development and Testing roles), since 'No' is the reference category for this specific factor (Table 1).

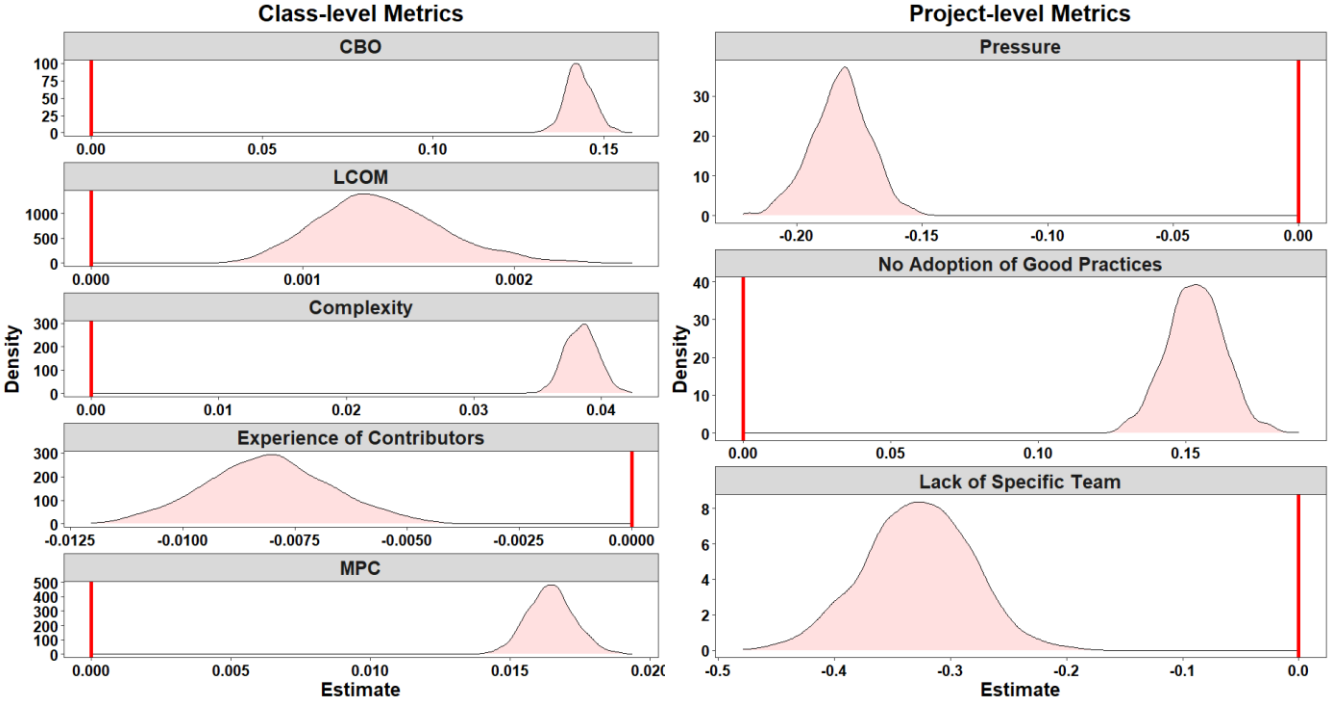

FIG. 4: IMPORTANCE OF MICRO- AND MACRO-TD PARAMETERS

Interpretation of Micro TD-Management Results: When interpreting the OR values, the range of the metrics, as well as the ease with which the metrics can be changed by one unit, must be considered. By considering that: (a) *LCOM* is a sensitive metric, whose values can change even by the addition of only one method or splitting a class [23]; whereas (b) *coupling* and *complexity* metrics are fluctuating less along evolution and the improvement of their metric scores requires more complex refactoring opportunities, e.g., replacing conditionals with polymorphism or re-arranging methods in classes; we can suggest that managing cohesion is a promising way for reducing the probability of a class to classified as HIGH_TD. The sign in these relations is positive, as expected, since an increase in the levels of these qualities hurts the maintainability of the system [24]. Interestingly, an increase in

the *Experience of Developers* tends not to reduce the probability of writing lowered-TD code, but rather increases it. A tentative interpretation of this, which is also supported by the literature [25], is that experienced developers are more reluctant to change their development habits or follow standards in their way programming.

> The findings of the study suggest that proper object-oriented design, dealing with modularity (i.e., balance between coupling and cohesion), abstractness, and unnecessary complexity; can lead to improved TD Micro-Management.

Interpretation of Macro TD-Management Results: Regarding project-level metrics, we can also make some interesting observations. First, we can conclude that the sign of the relations (as presented in Table 2) is intuitive. In particular, putting pressure on the developers to make frequent commits (*Pressure*) is a good practice in software development [27], supported by many modern development methodologies. The frequent committing dictates a careful development process, possibly implying *Continuous Integration* strategies, and frequent testing, since committed code must pass the functional and quality tests. Additionally, the *Non-Adoption of Quality Practices* (e.g., not performing quality checks, leading to inefficiencies creeping into the code) is expected to lead to accumulation of code TD, especially by considering that the majority of TD measurement tools identify TD items, by counting quality rule violations. Finally, our results suggest that the use of *Different Teams for Development and Testing* is a practice that can aid in improving quality, since a different perspective and viewpoint is always beneficial for effective quality audits [29].

> By considering the range of values for the TD Macro-Management parameters, as well as the ease of changing their values by one unit, we can claim that committing code daily is a practice that must be promoted among practitioners.

Regarding the random effects of the final model (see Table 2), the variance decomposition showcases that the proportion of the variance attributed to classes differences is lower ($\sigma^{2\,UB}_{class} = 0.067, 95\%\ UBCI\ [0.046, 0.103]$) compared to the variance attributed to project differences $\sigma^{2\,UB}_{project} = 0.237, 95\%\ UBCI\ [0.189, 0.289]$. Therefore, an interesting follow-up investigation would be to explore whether a practitioner should invest in TD Macro-Management to prevent TD accumulation. To provide a straightforward answer to the above critical issue, we decided to perform a sensitivity analysis by examining, if the insertion of project-level metrics contributes to the better understanding of the examined phenomenon—suggesting that the TD Macro-Management parameters play an important role on if a class is classified as a HIGH_TD one. To this end, we repeated the analysis by applying the proposed UBGLMM approach relying, solely, on the set of class-level metrics and compare the findings to the model incorporating both class- and project-level metrics. To compare the performances of Model A (containing both Micro- and Macro-Management parameters) versus Model B (containing only Micro-Management parameters), we made use of the Wald test followed by a *win-tie-loss* strategy [30]. In other words, we counted the number of times the Wald test resulted into a statistically significant difference between the fitting of the two models, while for selection purposes, the model with the lowest AIC value is finally chosen. The findings indicated that Model A presented lower AIC values compared to Model B for the whole set of repetitions. Regarding the results of the hypothesis testing procedure, the p-value of the Wald test was below the alpha level of 0.05 for the entirety of the repetitions.

> Based on our empirical evidence we can claim that TD Macro-Management can contribute to an effective TD prevention.

5 Implications to Researchers and Practitioners

The findings of this study (as presented in Table 2, Figure 4, and the boxed text in Section 4) can be summarized as follows: (a) TD Macro-Management showed to be an efficient way for preventing classes of a project to accumulate TD; (b) Coupling, Cohesion, and Complexity metrics are the most important parameters of TD Micro-Management—as also supported by the literature; and (c) Number of Commits, Use of a Different Team for Testing and Development, and the Adoption of Good Quality Practices are the most important TD Macro-Management parameters. These findings can be important for both researchers and practitioners.

Implications to Researchers: First, based on the fact that the proposed approach was able to provide empirical evidence, considering the nested nature of the problem, we encourage researchers to apply the proposed approach for various software engineering problems—we expect that almost all software engineering research problems can be approached as such. The uniform analysis at both levels, aids in surpassing important mining software repository problems (e.g., the overrepresentation of projects that contribute a large portion of the dataset [31]) advancing the state-of-research, which until now focused only at one of the two levels. Applying a nesting approach can: (a) limit the need of aggregating data from the class-level to the project-level, so as to work with project metrics; (b) limit the unhandled dependence of data that rely on the same project, being treated as independent observations; and (c) enables the development of models combining information at both levels.

Additionally, based on finding (a) we confirm the relevance of the TD causes literature [3][4][12][13], providing the first quantitative and independent empirical evidence on their relation to TD accumulation. Interesting future work in this direction would be to assess more causes, and provide additional metrics for their quantification. Additionally, another extension to this work would be to explore project metrics in isolation so as to explore their relation to the percentage of classes in a system that are classified as HIGH_TD. Finally, based on

finding (c) we can encourage researchers to propose approaches or processes that will integrate TD accumulation prevention measures. An important step in this direction would be the adoption and assessment of such approaches by software practitioners, mostly evaluating their long-term costs and benefits. For instance, how does the cost of "*Adopting Good Quality Practices*" compare to the benefit that they can bring (e.g., "*Reduced Maintenance Cost*", "*Reduced Bug Fixing Costs*", etc.)? An analysis of the point at which process improvements are over-engineered is very important in order not to over-spent resources (increasing cost) for gaining a minimal benefit. Similar approaches are well-established in the manufacturing domain, namely: Poor Quality Cost Management [32].

Implications to Practitioners: Based on finding (a) we encourage project managers to pay attention to technical management practices that can prevent the accumulation of TD, since there seems to be an important association between managerial decisions and TD costs. This implication becomes even more important by considering, apart from the impact, also the wide scope of such interventions. Additionally, by synthesizing findings (b) and (c), we encourage technical managers to adopt into their processes the following steps:

- Impose developers to commit their code at least once per day;
- While committing, the adoption of good quality practices must be automatically assessed, through tools.
- Quality gates must check the coupling, cohesion, and complexity of the code. In case of a need for a quality compromise class cohesion must be advised: i.e., opt for more and small classes, that might present some additional coupling;
- Upon code commit, a separate team must check the code manually, both from a functional or a non-functional perspective.

Despite the fact that the above steps are strongly supported by empirical evidence, we need to highlight (as mentioned before) that the ordering, effectiveness, and cost feasibility of such approach needs to be studied in future work.

6 Threats to Validity

This study exploring the importance of TD Micro- and Macro-Management with regard to the probability of software artifacts to exhibit high values of TD is based on specific metrics outlined in Table 1 and Table 2. TD Micro-Management metrics, such as size, complexity, coupling and cohesion have been widely studied and their relation to software quality is proven. On the other hand, the field of TD Macro-management metrics reflecting project-wide strategies and policies, is less studied. As a result, the findings are subject to construct validity threats in the sense that the examined class probability of accumulating high or low levels of TD might be loosely connected to the selected metrics. Furthermore, even if the causes of TD are assumed to be valid, the TD metrics for expressing them and the associated calculation method are not unique. A set of different metrics for capturing TD Macro-Management (e.g., extract through the list of worst reasons of TD smells [35]) strategies might have led to different findings. Nevertheless, we believe that the use of a forward selection strategy for the macro-management metrics resulted in a set of statistically significant metrics which are sound, quantifiable and diverse. For the response variable expressing whether a class is suffering from HIGH_TD or not, concerns could be raised on its accuracy, considering the challenges in objectively identifying and measuring TD. However, for the employed dataset, the response variable reflects the level of agreement among three leading TD analysis tools, meaning that a high TD class has a high probability of being unanimously designated as such by three different approaches, mitigating the relevant construct validity threat.

The fact that the analysis has been performed on 100 open-source Java projects threatens the external validity of our findings. In particular, we cannot generalize the observations regarding the importance of TD Micro- and Macro-Management approaches to industrial software or programs written in different programming languages. Nevertheless, the longevity, number of projects and the good practices followed by the Apache Software Foundation regarding the organization and management of their projects, partially mitigate the generalization threats as these open-source projects follow strict guidelines and closely monitor the progress. In any case, replication studies on projects with different characteristics can shed light into the value of TD Macro-Management processes.

Finally, reliability threats for the kind of study that we have presented are associated to the ability of replicating it and reaching the same results. To mitigate this threat, the study protocol is extensively described in Section 3 explicitly listing all data collection and analysis steps. Researcher bias has been avoided since the dataset has been subject only to automated analysis with no subjective interpretation by the researchers. A replication package consisting of all metrics values for each unit of analysis is available and we encourage the independent replication of the investigation in similar settings[5].

7 Conclusion and Future Work

In this study, we aimed at identifying possible causes of TD accumulation, so as to provide explicit suggestions for avoiding the existence of HIGH_TD classes in a software system, by applying with Micro- or Macro-Management interventions. The nature of the problem (many classes belong to the same project, and share some common characteristics) led us in using a nested modelling approach to surpass the problems that traditional correlation analysis would face. In particular, we proposed the *Under-Bagging based Generalized Linear Mixed Models* (UB-GLMM) data analysis methodology that can use class- and project-level metrics in the same model and discriminate their fixed and random effects.

The results of the study suggested that TD Micro-Management (methods that manage class-level parameters) and TD Macro-Management approaches (methods that manage project-level parameters) can both contribute

[5] https://users.uom.gr/~a.ampatzoglou/aux_material/TD_Nested.xlsx

to efficient prevention of TD accumulation. In particular, we have provided quantitative empirical evidence that Reducing Coupling Between Objects, Limiting Lack of Cohesion of Methods, Reducing Complexity, Adopting Good Quality Practices, Controlling Commit Frequency, and Using Different Teams for Development and Testing can be efficient ways for preventing the accumulation of TD into classes. The results have been interpreted and contrasted with existing literature (see Section 4), and various implications for researchers and practitioners have been stated (see Section 5).

ACKNOWLEDGEMENTS

The work of Dr. Arvanitou was financially supported by the action "Strengthening Human Resources Research Potential via Doctorate Research" of the Operational Program "Human Resources Development Program, Education and Lifelong Learning, 2014-2020", implemented from State Scholarship Foundation (IKY) and co-financed by the European Social Fund and the Greek public (National Strategic Reference Framework (NSRF) 2014–2020). The work of Mr. Nikolaidis is funded by the University of Macedonia Research Committee as part of the "Principal Research 2020" funding program. The work of Dr. Chatzigeorgiou and Dr. Ampatzoglou has been financed from the European Union's Horizon 2020 Research and Innovation Programme through SmartCLIDE project under Grant Agreement No. 871177.

REFERENCES

[1] E. M. Arvanitou, A. Ampatzoglou, S. Bibi, A. Chatzigeorgiou, I. Stamelos, "Monitoring technical debt in an industrial setting", 23rd International Conference on the Evaluation and Assessment in Software Engineering (EASE' 19), Copenhagen, Denmark, ACM (2019), 14-17 April

[2] Z. Li, P. Avgeriou, P. Liang, "A systematic mapping study on technical debt and its management", J. Syst. Softw., 101 (2015), Elsevier, pp. 193-220.

[3] Rios, N., Spínola, R. O., Mendonça, M. and Seaman, C. (2020). The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. Empirical Software Engineering. pp. 1-72.

[4] Martini, A., Bosch, J. and Chaudron, M. (2014). Architecture technical debt: Understanding causes and a qualitative model. 40th Conference on SEAA. pp. 85-92.

[5] V. Lenarduzzi, V. Mandić, A. Katin, and D. Taibi. 2020. "How long do Junior Developers take to Remove Technical Debt Items?", 14th International Symposium on Empirical Software Engineering and Measurement (ESEM '20). ACM, 1–6.

[6] T. Amanatidis, N. Mittas, A. Moschou, A. Chatzigeorgiou, A. Ampatzoglou, L. Angelis, "Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities", Empir. Softw. Eng., 25 (5) (2020), pp. 4161-4204

[7] Carvalho DV, Pereira EM, Cardoso JS. Machine Learning Interpretability: A Survey on Methods and Metrics. Electronics. 2019; 8(8):832

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

[9] Ben Idris, M., 2020. Investigate, identify and estimate the technical debt: a systematic mapping study. SSRN: 3606172.

[10] Zazworka, N., Spínola, R.O., Vetro', A., Shull, F. and Seaman, C., 2013, April. A case study on effectively identifying technical debt. 17th International Conference on Evaluation and Assessment in Software Engineering (pp. 42-47).

[11] Zazworka, N., Vetro, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C. , Shull, F., 2014. Comparing four approaches for technical debt identification. Software Quality Journal, pp.403-426.

[12] Martini, A. and Bosch, J. (2017). On the interest of architectural technical debt: Uncovering the contagious debt phenomenon. Journal of Software: Evolution and Process, 29(10).

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

[14] Jiang, J. Linear and generalized linear mixed models and their applications. Springer Science & Business Media, 2007.

[15] Raghuvanshi, B. S., & Shukla, S. (2019). Class imbalance learning using UnderBagging based kernelized extreme learning machine. Neurocomputing, 329, 172-187.

[16] Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.

[17] Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. SIGKDD explorations newsletter, 6(1), 20-29.

[18] M. Aniche, Java code metrics calculator (CK), 2015.

[19] A. Ampatzoglou, A. Gkortzis, S. Charalampidou and I. Stamelos, "An Embedded Multiple-Case Study on OSS Design Quality Assessment across Domains", 7th International Symposium on Empirical Software Engineering and Measurement (ESEM'13), ACM, 10 - 11 October 2013, Baltimore, USA.

[20] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python Framework for Mining Software Repositories," 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2018.

[21] A. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A framework for managing interest in technical debt: an industrial validation," in Proceedings of the 2018 International Conference on Technical Debt, 2018, pp. 115–124.

[22] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, "Software metrics fluctuation: a property for assisting the metric selection process", Information and Software Technology, vol. 72, pp. 110-124, 2016.

[23] Harrison, X. A., Donaldson, L., Correa-Cano, M. E., Evans, J., Fisher, D. N., Goodwin, C. E., ... & Inger, R. (2018). A brief introduction to mixed effects modelling and multi-model inference in ecology. PeerJ, 6, e4794.

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

[25] D. Tsoukalas, N. Mittas, A. Chatzigeorgiou, D. Kehagias, A. Ampatzoglou, T. Amanatidis, L. Angelis "Machine Learning for Technical Debt Identification," IEEE Trans. Softw. Eng., pp. 1–1, 2021.

[26] Van Vliet, H., and Van Vliet, J.C., 2008. Software engineering: principles and practice (Vol. 13). Hoboken, NJ: Wiley & Sons.

[27] Asklund, U. and Bendix, L., 2002. A study of configuration management in open-source software projects. IEE Proceedings - Software, 149(1), pp.40-46.

[28] K. Herzig and A. Zeller, "The impact of tangled code changes," in 2013 10th Working Conference on Mining Software Repositories (MSR), May 2013, pp. 121–130

[29] Ammann P., and Offutt J., "Introduction to Software", Cambridge Press, 1st Edition, 2008.
[30] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research, 7, 1-30.
[31] N. Saarimäki, S. Moreschini, F. Lomio, R. Penaloza, and V. Lenarduzzi, "Towards a Robust Approach to Analyze Time-Dependent Data in Software Engineering", IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER '22), Hawaii, USA, April 2022.
[32] Ampatzoglou A, Arvanitou E-M, Ampatzoglou A, Avgeriou P, Tsintzira A-A, Chatzigeorgiou A. 2021. Architectural decision-making as a financial investment: an industrial case study. Information and Software Technology 129:106412
[33] D. Tsoukalas, A. Chatzigeorgiou, A. Ampatzoglou, N. Mittas, D. Kehagias, "TD Classifier: Automatic Identification of Java Classes with High Technical Debt", 5th International Conference on Technical Debt (TechDEBT' 22), ACM, Pennsylvania, USA, May 2022
[34] Riaz, M., Mendes, E. and Tempero, E. "A systematic review on software maintainability prediction and metrics", 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09), IEEE Computer Society, Florida, USA, pp. 367-377, 15-16 October 2009.
[35] D. Falessi and R. Kazman, "Worst Smells and Their Worst Reasons," 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), 2021, pp. 45-54.

Nikolaos Nikolaidis is an PhD student at the Department of Applied Informatics in the University of Macedonia, Greece. He received a BSc in Applied Informatics from the same university in 2018. He is currently employed as a research associate in the Software Engineering group of University of Macedonia, working on multiple research projects. His research interests are technical debt management, mining software repositories, and software quality assurance.

Dr. Nikolaos Mittas is an Assistant Professor in the Department of Chemistry at the International Hellenic University. He received the BSc degree in Mathematics from the University of Crete and the MSc and PhD degrees in Informatics from the Aristotle University of Thessaloniki (AUTH). His current research interests are focused on the application of statistics and data analytics in the area of Software Engineering and Project Management.

Dr. Apostolos Ampatzoglou is an Assistant Professor in the Department of Applied Informatics in University of Macedonia (Greece), where he carries out research and teaching in the area of software engineering. Before joining University of Macedonia, he was an Assistant Professor in the University of Groningen (Netherlands). He holds a BSc on Information Systems (2003), an MSc on Computer Systems (2005) and a PhD in Software Engineering by the Aristotle University of Thessaloniki (2012). He has published more than 100 articles in international journals and conferences, and is/was involved in over 15 R&D ICT projects, with funding from national and international organizations. His current research interests are focused on technical debt management, software maintainability, reverse engineering software quality management, open-source software, and software design.

Dr. Elvira-Maria Arvanitou is a Post-Doctoral Researcher at the Department of Applied Informatics, in the University of Macedonia, Greece. She holds a PhD degree in Software Engineering from the University of Groningen (Netherlands, 2018), an MSc degree in Information Systems from the Aristotle University of Thessaloniki, Greece (2013), and a BSc degree in Information Technology from the Technological Institute of Thessaloniki, Greece (2011). Her PhD thesis has been awarded as being part of the top-3 ICT-related in Netherlands for 2018. Her research interests include technical debt management, software quality metrics, and software maintainability.

Dr. Alexander Chatzigeorgiou is a Professor of Software Engineering in the Department of Applied Informatics and Dean of the School of Information Sciences at the University of Macedonia, Thessaloniki, Greece. He received the Diploma in Electrical Engineering and the PhD degree in Computer Science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999 he was with Intracom S.A., Greece, as a software designer. His research interests include object-oriented design, software maintenance, technical debt and evolution analysis. He has published more than 150 articles in international journals and conferences and participated in a number of European and national research programs. He is a Senior Associate Editor of the Journal of Systems and Software.