A Framework for Managing Interest in Technical Debt: An Industrial Validation

Areti Ampatzoglou[1,4], Alexandros Michailidis[2], Christos Sarikyriakidis[3], Apostolos Ampatzoglou[1,4], Alexander Chatzigeorgiou[4], and Paris Avgeriou[1]

[1] Department of Mathematics and Computer Science, University of Groningen, Netherlands
[2] School of Science and Technology, International Hellenic University, Thessaloniki, Greece
[3] Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Kozani, Greece
[4] Department of Applied Informatics, University of Macedonia, Greece

areti.ampatzoglou@rug.nl, alex.michailidis.dev@gmail.com, xsarikiriakidis@gmail.com, a.ampatzoglou@rug.nl, achat@uom.gr, paris@cs.rug.nl

ABSTRACT

Technical debt management entails the quantification of principal and interest. In our previous work we had introduced a framework for calculating the Technical Debt Breaking Point (TD-BP), which is a point in time where the accumulated interest becomes larger than the principal; thus the debt of the company is no longer sustainable after this point in time. In this paper, we instantiate this framework and validate its ability to assess the breaking point of source code modules in an industrial setting. The results of the validation suggest that the calculated TD-BP is strongly correlated to experts' opinion on the sustainability of modules, and that it can accurately rank components, based on their maintenance difficulty.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; *Designing software*; Software design engineering • **Social and professional topics → Professional topics**; *Management of computing and information systems*; Quality assurance

KEYWORDS

Technical debt, interest, empirical study

TechDebt '18, May 27–28, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5713-5/18/05…$15.00
https://doi.org/10.1145/3194164.3194175

ACM Reference format:
A. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, A. Ampatzoglou, A. Chatzigeorigiou, and P. Avgeriou. 2018. *Proceedings of Technical Debt 2018 International Conference (TechDebt 2018), Gothenburg, Sweden, May 2018*, 10 pages.

1 INTRODUCTION

In the last years, the technical debt research community has been spending increasingly large effort on the different activities of Technical Debt Management (TDM). To a large extent, this research is focusing on issues related to cost and benefit analysis of software quality management [3]. An overview of TDM research is presented in, among others, two literature reviews [4, 5]. Although research on TDM has been intense in the last years, the community still faces major challenges, two of which are the focus of this paper: (a) a sound estimation of the amount of TD (i.e., the quantification of interest and principal [5]), and (b) the monitoring of the TD amount, as it increases due to the accumulation of interest [6].

In our previous work [6, 7], we begun to address these challenges, by defining a theoretical framework, named FITTED, that can be used for the long-term management of Technical Debt. FITTED considers that interest accumulating during software evolution, can potentially outgrow the amount required for repaying the principal of TD. Therefore, it is critical for project managers to be able to estimate the point in time where the accumulated interest will be equal to the TD principal. This point in time is called the "breaking point" (TD-BP), in the sense that any benefit deriving from the decision to take on technical debt is being neutralized after that point, i.e. the cost becomes higher than the benefit [8].

The aim of this paper is twofold: (a) we instantiate the FITTED framework by providing details on the calculation of principal and interest, and (b) we validate the proposed instantiation of the FITTED framework in an industrial setting. We note that both the instantiation of the framework and its subsequent validation are at the source code level; instantiation could also take place for technical debt at other levels but that is out of the scope of this paper. The instantiation of the framework is supported by a software tool which can extract data on: (a) the principal of technical debt, (b) the interest accumulated in every version of the system, and (c) the occurrence of the breaking point for every version of the system.

The validation of the framework was conducted in collaboration with a software development company. Specifically the tool was applied on two of their current software development projects, and its results were analyzed together with two software engineers. Compared to the state-of-research (see Section 2.1), this is the first automated approach that calculates the Breaking Point of Technical Debt and validates it within an industrial setting.

The rest of the paper is structured as follows: In Section 2, the FITTED framework is presented. In Section 3, we present our approach for validating FITTED, and in Section 4, we explain the design of our study. Next, we present the results of our analysis, in Section 5, and we discuss them, in Section 6. In Section 7 we refer to the threats to validity, and finally, in Section 8, we present our conclusions and future work.

2 BACKGROUND INFORMATION

2.1 Interest Theories in Technical Debt Management

Section 2.1 presents the results of several studies that have investigated interest in Technical Debt Management (TDM). Ampatzoglou et al. [5] and Li et al. [4] have suggested that interest is the most frequently financial term used in TDM research. It should be mentioned here that in economic theory, the level of interest rate (not interest per se) is the main subject of research, since interest is calculated upon interest rate. Nevertheless, in Technical Debt Management, interest is estimated according to a variety of proposed theories, presented later in this section and it is not calculated based on interest rate, which is a term not clearly defined in TD literature.

Ampatzoglou et al. [6] suggest that in literature, interest is described either as the extra effort during maintenance (in approximately 31% of primary studies), or as the extra maintenance cost (in 51% if primary studies). Therefore, as software economics mainly refer to cost as a function of effort, we can presume that in 82% of primary studies interest is defined as the extra cost/effort occurring during maintenance, because of the accumulated technical debt. In the rest of the literature, interest is approached through more high-level definitions─as in [10] and [18]─or through references to economic theory, i.e., it is defined as the increase rate of the amount of TD [11], or through software engineering concepts, i.e., it is defined as a change in a design-time quality attribute─e.g., [12] and [13]. Moreover, it is observed that almost 28% of primary studies recognize the effect of interest probability. Two of these studies [14, 15] approach TDM with a financial risk management theory and define interest probability as the standard deviation of interest rate, i.e., they consider it as the probability of TD to occur.

Additionally, in Ampatzoglou et al. [6] it is suggested that almost 21% of studies refer to the evolution of interest along time. Under this perspective, interest is characterized either as compound or continuously increasing. On the other hand, Chin et al. [16] suggest that there is one type of interest which is simple. In particular, they propose that the cost of the organization to restrict TD neither increases nor decreases, but it is stable across time. Finally, only 17% of the studies suggest a specific methodology for measuring interest. In most of the cases, the estimation is performed by using historical data, documentation, and maintenance.

2.2 FITTED: A Framework for Managing Interest in Technical Debt

To develop a theory on managing technical debt interest, we borrowed the rationale of the economic interest theories on the equilibrium achievement. More particularly, we adopt the idea of the breaking point at the money market, where money supply is equal to money demand. As long as accumulated interest is lower than the principal, the benefit derived by the initial decision to save effort is more than the cost generated by the extra effort to maintain the software. On the other hand, when the accumulation of interest overcomes the principal, then the cost needed for the system's maintenance becomes more than the money saved when technical debt incurred. Consequently, the point where the accumulated interest is equal to the principal (breaking point) is very critical in technical debt management and can help project managers in their decision making. Under this perspective, we map principal to money supply – since it represents the amount of money available – and the accumulated amount of interest with money demand – since it embodies the extra amount of money that will be necessary for future maintenance activities caused by the TD.

Figure 1 represents the FITTED Interest Theory: the horizontal axis depicts time, while the vertical axis stands for money amount. The blue curve $\Sigma(I)$ denotes the increasing cumulative interest, as the software develops. We consider cumulative interest as continuously increasing, because it consists of the accumulation of interest in every successive version of the project and because TD interest is compound [6]. Principal, on the other hand, is represented by the green line P and is also considered as increasing, in the sense that the introduction of new functionality to the software generates new TD items. Concerning principal, we have to point out that in our original analysis of the

FITTED framework [6], we have defined it as an amount of effort saved once while taking on technical debt and considered it stable throughout software evolution. However, as software grows, technical debt keeps accumulating. Therefore, in this study we take into account the development of principal and we calculate it during software development, so that we are able to reach a more accurate assessment of the version where breaking point occurs.

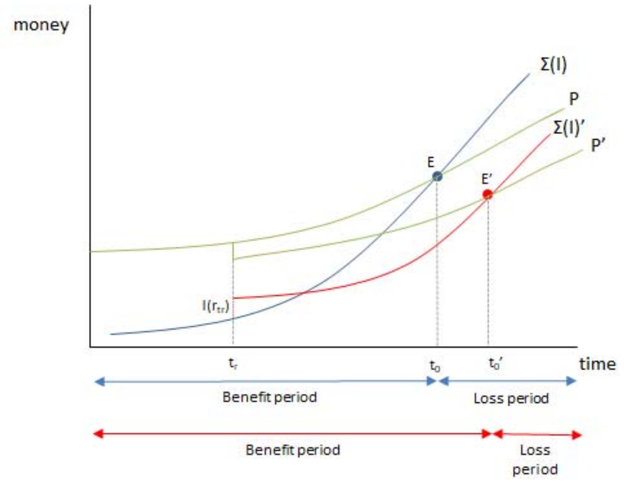

Figure 1: **FITTED Interest Theory**

As described by the figure, until time stamp t_0, interest is lower than the principal and taking on technical debt can be perceived as beneficial for the project. After time point t_0, interest accumulates and becomes larger than the principal and extra maintenance costs exceed the initial money saving. Therefore, the equilibrium point E, denotes the point in time t_0 where the complete amount of money saved by incurring TD (i.e., the principal) equals the money spent on extra maintenance activities caused by the incurred TD [8]. If the development team proceeds with some repayment activity, e.g. at timestamp t_r, accumulated interest increases, because of the addition of the repayment effort, and line Σ(I) shifts upwards to Σ(I)´. However, the slope of the line decreases, since lower maintenance activities are expected in the future due to the repayment. On the other hand, principal decreases at time stamp t_r, because of the repayment activity, and then follows the course of line P´. Consequently, the equilibrium point moves to the right E´, and the benefit period increases to time stamp t_0´ [8].

3 PROPOSED APPROACH

In order to instantiate the aforementioned approach, we need to describe the way that the two components of TD-BP can be calculated. To this end, in Section 3.1 we describe the employed way for calculating principal at source code level, whereas in Section 3.2, we present the proposed approach for assessing interest on source code artifacts.

3.1 Principal Estimation

As proposed in literature, TD principal can be calculated as a function of three variables [17]. The first variable is the number of problems that must be fixed, the second one is the time required to fix each one of these problems, and the third one is the cost for fixing each problem. Regarding the number of must-fix problems. In our previous work [7] we have presumed that there is an actual design quality for any object-oriented software, which can be estimated by a proper fitness function. Under this perspective, the 'spread' between the optimal and the actual design, as it can be derived by the difference in their fitness function values, can be mapped to the principal, i.e., the effort needed to convert the actual system to the correspondent optimum one. This notion of 'distance' is depicted in Figure 2, as $\mathtt{Effort}_r$.

For the purpose of this study, we have decided to estimate principal, based on the computation of a widely used OSS platform, SonarQube—formerly known as Sonar, which is a state-of-the-art tool for calculating TD principal. According to its documentation[1], SonarQube aims at the continuous evaluation of software quality. SonarQube can assess the quality of software on a plentitude of programming languages, generating documentation on quality measures and issues, such as coding rule violations. The analysis has been performed according to the platform's default format, without any further configuration. The platform algorithm is based upon an adopted version of the SQALE method proposed by Letouzey [18], in which a remediation index is obtained for 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, for example, 20 minutes per uncovered line of code. Finally, for each artifact, the remediation index relating to all the characteristics of the Quality Model is obtained by adding all remediation indices linked to all quality requirements. The resulting SQALE Quality Index is considered to represent the principal of the TD for the assessed source code.

3.2 Interest Estimation

In this section we discuss the way we calculate interest in the proposed approach. In particular, in Section 3.2.1 we present the theoretical background of the approach,

[1] https://docs.sonarqube.org/display/SONAR/Documentation

whereas in Section 3.2.2, we present the metrics and the method that we have used for its instantiation at the source code level, also known as code TD.

3.2.1 FITTED Interest Theory

Assuming the optimal and actual systems of Figure 2, in the common case, maintaining the optimum system requires less effort than maintaining the actual system. As shown in the figure, adding a new feature A to the optimum system needs 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 interest that is accumulated during this maintenance activity, i.e., the addition of feature A.

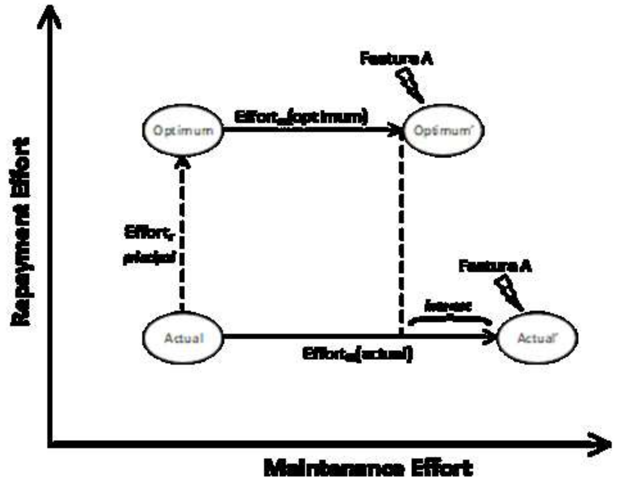

Figure 2: Increased maintenance effort for TD item

However, 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 historic data, by considering past effort spent on maintenance activities. More specifically, although many measures can be used for the calculation of past effort, we have selected to use the average number of lines of code added between sequential releases. Added lines of code indicate the amount of effort required for the addition of new functionality and in a way the effort needed for changing existing modules. Supposing that an average of `k` lines of code are added in the transition from one version to another, then we can accept that adding `k` lines to a high design quality system (as denoted by the assigned fitness function value) is easier than making the same addition to a lower design quality system. At this point we suppose that maintenance effort is dependent on the design quality, as in (1):

$$Effort_m = c \cdot FitnessValue \tag{1}$$

where `c` is an arbitrary constant. Based on the abovementioned approach, the fitness value for both the optimum and the actual systems can be calculated and therefore we can capture the proportion of the theoretical over the actual effort, as in (2):

$$\begin{gathered}\frac{Effort_m(optimum)}{Effort_m(actual)} = \frac{c \cdot FitnessValue(optimum)}{c \cdot FitnessValue(actual)} \Rightarrow \\ Effort_m(optimum) = \frac{FitnessValue(optimum)}{FitnessValue(actual)} \cdot Effort_m(actual)\end{gathered} \tag{2}$$

Hence, if past maintenance effort (i.e., the number (`k`) of added lines) is used to define the actual maintenance effort, then the optimum effort can be directly obtained. Consequently, the amount of interest accumulated between any two sequential versions can be calculated as the distance between the optimum and the actual effort and is given by (3) and (4).

$$Interest = \Delta Effort = k \cdot \left(1 - \frac{FitnessValue(optimum)}{FitnessValue(actual)}\right) \tag{3}$$

$$Interest = \Delta Effort = k \cdot \left(\frac{FitnessValue(optimum)}{FitnessValue(actual)} - 1\right) \tag{4}$$

Since *the optimum value for each one of the applied metrics could be either the minimum or the maximum value*, interest is given by the actual effort (`k`) multiplied by `1` minus the ratio of the optimum fitness value to the actual fitness value, if the optimum value is the maximum one—as in (3). Otherwise, if the optimum value is the minimum one, interest is given by the actual effort (k) multiplied by `1` minus the ratio of the actual fitness value to the optimum fitness value—as in (4).

3.2.2 Interest in Source Code Debt

Given the fact that interest is closely related to the system's maintainability [19], our analysis concentrates on the estimation of interest, based on well-known maintainability models. Maintainability models are sets of low-level metrics that quantify design-time quality attributes, such as inheritance, cohesion, coupling, complexity and size, which are somehow aggregated in one maintainability index. Therefore, given the relation between maintainability and interest, in this study we define the fitness value used for the estimation of the interest, using the set of metrics that are accurate maintainability predictors. Riaz et al. [20], in their systematic review have studied all existing maintainability models, and reported on their accuracy. Based on the results of that systematic literature review, a list of ten metrics stand out as successful maintainability predictors (see Table 1)—and this is the set of metrics that we use in our study. Table 1 depicts these metrics and the quality attribute to which they refer, as presented by Arvanitou et al. [21]. Nevertheless, we need to note that in

our case no aggregation function is used for these metrics, since it is out of the scope of this paper.
More specifically, in order to assess the optimum design, our analysis consists of the following steps. Firstly, classes of high similarity level with the investigated class are found (see Section 3.4). The identification and comparison of a class with structurally similar ones or the calculation of the fitness function is crucial, in the sense that the maintainability of a class of tens of LoC cannot be comparable with one with hundreds of LoCs. Secondly, the optimum value for each one of the aforementioned metrics is identified, the difference between the actual and the optimum fitness values is calculated and the average distance between the actual and optimum design is assessed. Subsequently, the interest is calculated as described above, in equations (3) and (4). Finally, taking into account the number of LoC that can be added in a unit of time (according to past data) and the developer's hourly rate, interest is estimated in terms of money.

Table 1: Object-Oriented Metrics

Metric	Description	Quality Attribute
DIT	Depth of Inheritance Tree: Inheritance level number, 0 for the root class.	*inheritance*
NOCC	Number of Children Classes: Number of direct sub-classes that the class has.	*inheritance*
MPC	Message Passing Coupling: Number of send statements defined in the class.	*coupling*
RFC	Response For a Class: Number of local methods plus the number of methods called by class methods.	*coupling*
LCOM	Lack of Cohesion of Methods: Number of disjoint sets of methods (a set of methods that do not interact with each other), in the class.	*cohesion*
DAC	Data Abstraction Coupling: Number of abstract types defined in the class.	*coupling*
WMPC	Weighted Method per Class: Average cyclomatic complexity of all methods in the class.	*complexity*
NOM	Number of Methods: Number of methods in the class.	*size*
SIZE1	Lines of Code: Number of semicolons in the class.	*size*
SIZE2	Number of Properties: Number of attributes and methods in the class	*size*

3.3 Breaking Point

According to the abovementioned analysis, an estimate of the interest in USD can be achieved, based on the difference between theoretical effort (maintenance performed on the optimum system) and the actual effort, on past data concerning the time required for the addition of lines of code, and the per hour cost of a developer's work. Therefore, given the principal, as it comes from SonarQube, and knowing the interest accumulated in one version of the software, the ratio of the principal over the interest gives the number of versions after which the company will reach the breaking point, as shown in (5).

$$TD\_BP = \frac{Principal(\$)}{Interest(\$)} \quad (5)$$

3.4 Tool Support

To boost the applicability of the aforementioned methodology, and moreover to validate the FITTED framework, we created a desktop application, developed in Java, called Breaking Point Calculator (BPC). The procedure is separated into two phases, the analysis phase and the computation phase. During the analysis phase, the source code of the project is analyzed and the metrics needed for the calculation of the results are retrieved. Each project is analyzed with the aid of two software tools, SonarQube and Percerons Client:

- ***SonarQube*** is used for the extraction of five metrics which are used as a factor of similarity and technical debt principal. The metrics used for identifying similar classes are depicted in Table 2, accompanied by their description. The analysis of a project by SonarQube can take several hours, thus it takes place on our server in UoM (University of Macedonia) where it is installed. After the analysis is complete the metrics are stored in SonarQube database, which we access to retrieve them.
- ***Percerons Client*** is a toolset[2] developed to support empirical software engineering research. Percerons is able to calculate metrics for every class contained in a .JAR file and produces a .CSV output [22]. Percerons Client is used to extract the object-oriented metrics for each class, on every version of the investigated project. The output document is parsed by BPC and the metrics of each class are stored into a database.

Table 2: Similarity Metrics

Metric	*Description*
Classes	Number of classes (including nested classes, interfaces, enums and annotations).
Complexity	The complexity calculated based on the number of paths through the code.

[2] http://www.percerons.com

Metric	***Description***
Functions	Number of functions.
NCLOC	Number of physical lines that contain at least one character which is neither a whitespace nor a tabulation nor part of a comment.
Sqale Index	Effort needed to fix all maintainability issues.
Statements	Number of statements.

Once the analysis phase is completed, we proceed with the computation of the breaking point, by selecting the project under investigation, along with the computation scope, i.e., whether we want to perform the computation on "class" or "package" level. Classes and packages selected for our scope are being referred to as artifacts. The goal of the computation phase is the calculation of the breaking point. To this end, we retrieve all the artifacts for each version of the project selected, along with the similarity and object-oriented metrics that characterize each one. For every artifact under study, we detect the most similar ones by using the aforementioned similarity metrics. Thereafter, we calculate the optimal artifact by selecting the best values among the grouped artifacts.

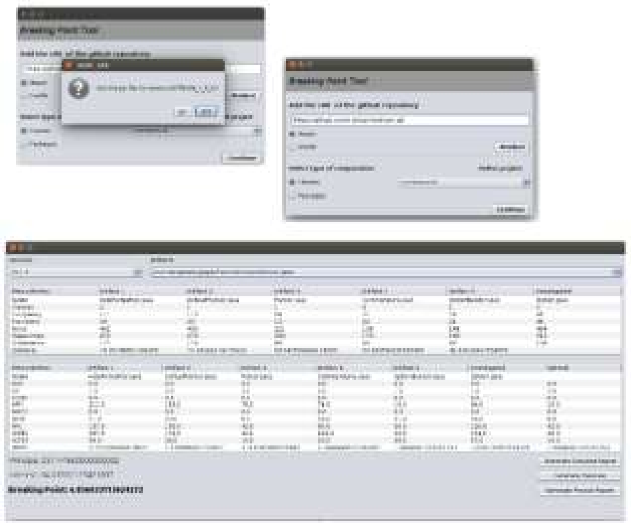

Figure 3: **Breaking Point Calculator**

On the completion of this phase, we are able to proceed with the calculation of the fitness value, the principal, the interest and the breaking point. To sum up, by the time the computation phase is completed, the following information can be extracted for any artifact from any available project version:

1. The five most similar artifacts to the investigated one, along with the values of the similarity metrics and the similarity rate.
2. The values of the object-oriented metrics for the most similar artifacts and the investigated one.
3. The values of the quality metrics for the optimum artifact.
4. The principal in USD.
5. The interest in USD.
6. The breaking point in versions.

Finally we are producing overview reports for the project in .CSV documents in order to get a more comprehensive picture of it, as well as to enable the comparison of multiple artifacts. Some screenshots of the tool are provided in Figure 3.

4 CASE STUDY DESIGN

To validate the proposed framework and its ability to estimate the TD-BP, we performed an industrial case study in a small-medium enterprise (SME) in Greece, which is active in the domain of mobile applications. The company wants to keep its anonymity, thus all records in the dataset have been anonymized, and no personalized information can be provided, either about the company and its projects, or the case study participants. The case study is designed according to the guidelines by Runeson et al. [23]. In the next sections we present the four parts of our research design, i.e., objective and research question (see Section 4.1), case selection and units of analysis (see Section 4.2), data collection (see Section 4.3), and analysis (see Section 4.4).

4.1 Objective and Research Question

The goal of the case study is to validate the FITTED approach, regarding its ability to assess the breaking point in an industrial environment. Therefore, our study focuses in the estimation of the accuracy of the framework in assessing the breaking point. To this end, we derived the following research question:

RQ: Is the calculated breaking point metric correlated with the effort required for maintenance activities?

This Research Question aims at validating the proposed instantiation method of FITTED, and in particular it assesses if the TD-BP is correlated with maintenance efforts.

4.2 Case Selection and Units of Analysis

To collect data for our case study, we executed the Breaking Point Calculator Tool on the source code of two projects of the collaborating SME: (a) a lottery engine, and (b) an online betting system. Since both systems are medium scale and a manual inspection of the code might be required from the participants, we preferred to scope our analysis at high-level packages. An additional reason for this decision is that by scaling up the level of granularity for our reporting, we come closer to the architectural debt notation that has higher impact than code TD, since larger amount of TD and interest are expected to occur.

Therefore, the units of our analysis are 18 packages, contained in these two different android application projects. The projects have been developed in different context and under dissimilar circumstances; hence they are expected to have different levels of accumulated TD and interest. Moreover, prerequisites for the selection of the projects were that the projects contained adequate number of versions and that the developers would be available to answer a short questionnaire concerning the development of the projects.

4.3 Data Collection

To answer the aforementioned research question, we performed a three phase analysis:

- ***TD Estimation***. We applied the Breaking Point Tool on the source code of the two applications, as described in Section 3.4, and obtained data on 8 versions of the two projects. From this analysis, we recorded three main variables: *interest* (I), *normalized principal per LoC* (NrP), and *breaking point (*TD-BP*)*.
- ***Extraction of Control Variables***. We compiled a list of packages, in a random ordering, and asked the developers to assign a Likert-scale score (1-5) to each one of these packages, with respect to the effort required to maintain them, and the extent to which they consider them sustainable (i.e., need to completely re-write them, due to heavy maintenance effort). The data extraction method for this phase was a questionnaire-based approach.
- ***Discussion and Interpretation of Obtained Results.*** During this phase we conducted a focus group with all developers (i.e., case study participants) and discussed their rationale for ranking specific packages with high sustainability scores, and others with a lower one.

4.4 Data Analysis

To answer the research question, we performed a Spearman Rank correlation, between the TD estimated and the control variable. The reason for performing a non-parametric test is that the dataset was not normally distributed, whereas the reason for performing rank correlation instead of actual values correlation is that we are not interested in the predictive power of the metrics (i.e., how well they can predict the values of the sustainability), but on the consistency of the proposed metric. According to the 1061-1998 IEEE Standard for Software Metrics [24], ***consistency*** assesses whether there is consistency between the ranks of the quality characteristic and the ranks of the metric under study. Consistency determines if a metric can accurately rank artifacts in terms of quality. The criterion is quantified by the coefficient of rank correlation.

5 RESULTS

In this section we present the results of the empirical study, through presenting some demographics and subsequently answering the research question.

5.1 Study Demographics

In Table 3, we present the average descriptive statistics for the TD estimation variables and the average number of lines per class per version, between versions (a proxy of maintenance effort per version).

Table 3: Dataset Descriptive Statistics

TD Estimator	Min	Max	Mean	Std. Dev
NrP	0,138	1,163	0,690	0,302
I	0,188	179,616	28,228	55,721
TD-BP	0,085	366,279	49,602	88,572
AVG(LoC)	0,250	281,889	21,296	65,785

Furthermore, in Figure 4, we present the distribution of participants' responses with respect to the control variable, i.e., the sustainability of each examined packages. Finally, in Figure 5, we present the evolution of the three TD estimators. In order to fit all estimators in the same graph (since they have different range of values), we normalize all of them against their value in the first of the examined versions (for that version all values equal 1), and then plot in a line chart their evolution.

Based on the results of Figure 4, we observe that the majority of the examined packages have been assigned an average sustainability, whereas the packages with high sustainability are more than those with low sustainability. Another observation that can be made, based on the evolution analysis of TD estimators (see Figure 5) is that Principal is a more stable metric [14], i.e., provides less fluctuations, compared to interest (ranges from 100%-400% of the value of the first version) and the breaking point (ranges from 75% - 138% of the first version). Also, we can observe that the interest which increased a lot in the 2nd examined version is then decreasing along time, suggesting an improvement in the structural quality of the software. When developers were asked if they could explain the reason behind the enhancement of quality along evolution, one of them replied that: "*the quality of the code is a top priority for us. We are trying to structure our code in a modular manner, applying well-known patterns for that, like Model-View-Presenter (MVP)*[3]".

[3] MVP is a well-known pattern for mobile web development that is conceptually relevant to MVC. Online reference.

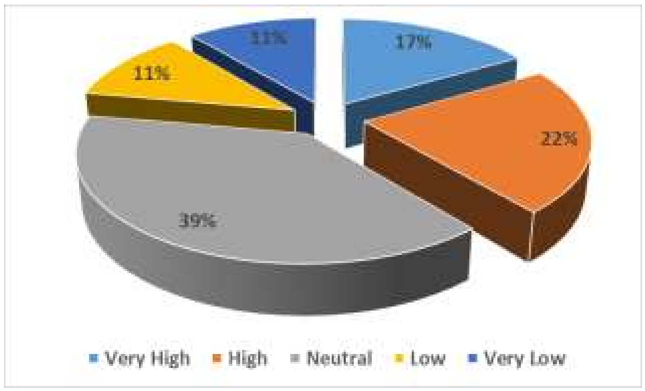

Figure 4: Control Variable Demographics

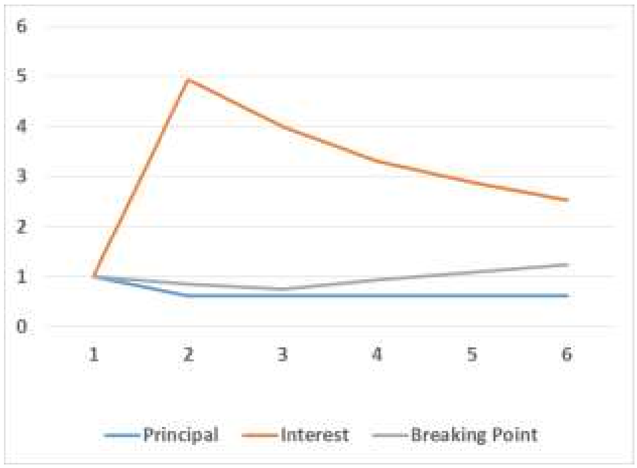

Figure 5: TD Estimators Evolution

5.2 Answering the Research Question

As mentioned in Section 4.4, the research question is answered using correlation analysis. The results of the analysis are presented in Table 4. In the table, statistically significant correlations are denoted with grey cell shading.

Table 4: Spearman Rho: Assessment Power

Statistical Measure	NrP	I	TD-BP
Correlation Coefficient	-0,010	-0,747	,773
Sig. (2-tailed)	0,970	0,000	,000
N	18	18	18

The results of the correlation analysis suggest that both interest and normalized principal are negatively correlated with software artifacts sustainability which is considered an intuitive outcome. However, only the relation between interest and sustainability is strong and statistically significant. By assessing the correlation between TD breaking point and sustainability, we can observe that the relation is statistically significant and the strongest among the three examined TD estimators. This outcome, suggests that the compound metric that we have proposed in this study is effective. This is an interesting finding in the sense that although principal in isolation is not correlated to sustainability, its inclusion in the calculation of TD-BP has increased its assessing power. The main reasons that the developers used for explaining high and low values of sustainability are summarized below:

- ***High Sustainability:***
  1. "*Provision of limited functionality*"
  2. "*Only shows some data that never change*"
  3. "*Presents a static user interface as well*"
  4. "*Low coupling class which manipulates small amount of data*"
  5. "*Obeys to the Single responsibility*"
  6. "*Used to present static details about the app*"
- ***Low Sustainability***
  1. "*High Coupling*"
  2. "*Handles feed from different data sources. Needs extra effort to be maintained*"
  3. "*Fundamental class of the app which handles many data and has many responsibilities*"

6 DISCUSSION

In this section, we interpret the case study results and discuss their implications for researchers and practitioners.

6.1 Explanation of Obtained Results

According to the results of the study interest (as calculated by the proposed approach) appears to be more sensitive than principal (as calculated by SonarQube) in the sense that it exhibits more fluctuations over the course of the project's evolution. This could be attributed to the fact that principal depends on the number of rule violations that SonarQube identifies, and these violations do not change drastically from one version to the other. On the other hand, interest as calculated by the proposed FITTED approach, takes indirectly into account many structural characteristics of software, as the 'optimum' design is assessed by identifying structurally similar classes. For example, even a change in the control flow structure of a method will be reflected on interest, whereas it could go unnoticed by the principal calculation. Thus, interest calculation is subject even to minor changes in the code leading to a more sensitive measure.

Considering the results of the correlation analysis between principal, interest, breaking point and software artifacts sustainability, it appears that the consideration of the structural characteristics on top of the TD issues identified by SonarQube has been successful. In other words, the combination of interest (which has a statistically significant correlation to sustainability) and principal for the extraction of the TD breaking point, leads to even stronger, and statistically significant correlation between the breaking point and sustainability.

It should be noted that in this study sustainability is related to the difficulty of performing maintenance or conversely the need to completely re-write software modules. A module has low sustainability if it undergoes frequent and extensive maintenance and at the same time its TD hinders maintenance. According to the responses of the study typical symptoms of a poor design, such as high coupling and violation of the single responsibility principle contribute to low sustainability. The fact that the proposed TD breaking point captures to a sufficient extent the notion of sustainability as perceived by the developers implies that TD issues combined with structural properties of the code can serve as a valid indicator of maintainability.

6.2 Implications for Researchers

Considering the need to assess the sustainability of software modules in an objective and accurate manner, the promising results regarding the correlation between TD-BP and sustainability should be checked against more projects. Given that the case study has been executed on two projects from the mobile domain, extension of the study to projects with different characteristics in terms of application domain, programming language and development process can be very valuable for tuning the calculation of the breaking point. It should be noted that the proposed application of FITTED targets code TD; however, the underlying theory can be extended to other levels such as architecture. Considering the architectural TD has a larger impact on system maintainability than code TD, obtaining the interest of an architectural element based on the notion of 'distance' to similar, 'optimum' elements can remove the existing barriers of effectively calculating a pragmatic TD measure.

6.3 Implications for Practitioners

The results of the study and the developers' feedback on the causes of low and high sustainability indicate that the adoption of rather 'classic' well-known practices (such as the use of patterns, conformance to the Single Responsibility Principle, low coupling and high cohesion) can improve sustainability. FITTED confirms that addressing code and design inefficiencies can move the breaking point, i.e. the time at which increased maintenance costs exceed any short-term benefits, further away to the future.

The proposed toolset in this study and especially the consideration of structurally similar modules that exhibit a higher quality can be of help to novice software engineers. Assessing principal, interest and breaking point in a relative manner that is by comparing a software module to that of peers can yield feasible opportunities for improvement. Moreover, the proposed toolset offers a viable way of continuous monitoring of TD critical levels, so as to act as a Quality Gate in the context of contemporary continuous integration practices.

7 THREATS TO VALIDITY

The results regarding the correlation between interest and breaking point and sustainability are subject to external validity threats meaning that they cannot be generalized to other application domains, programming languages or project sizes. Furthermore, TD principal estimation with the use of SonarQube has to be reported as a possible validity threat. Since the tool does not consider architecture when calculating TD principal, this may impact the precision of the estimation. Moreover, it should be noted that the developers in the study might have perceived the concept of sustainability in different ways. Since sustainability is one of the main variables in the analysis, construct validity threats arise, although an attempt to clarify the concept prior to the collection of results has been made.

8 CONCLUSION

Principal and Interest are key concepts in the Technical Debt literature as they provide a monetary representation of detectable inefficiencies. However, their value has to be demonstrated by proving their correlation to external software qualities. In this paper we demonstrate the feasibility of instantiating a theoretical framework which allows the calculation of the breaking point, which is a time stamp in which the accumulated interest becomes larger than the principal. The results of a validation in an industrial setting are very promising since they suggest that breaking point is strongly correlated to experts' opinion on the sustainability of modules.

ACKNOWLEDGMENTS

Work reported in this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780572 (project SDK4ED). Also, we would like to thank the anonymous (due to the request of the company) case study participants for their efforts in conducting this work.

REFERENCES

[1] W. Cunningham. 1992. The WyCash Portfolio Management System. In Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications. NY, USA, 29‑30.

[2] H. J. Harrington. 1987. Poor-Quality Cost: Implementing, Understanding, and Using the Cost of Poor Quality. CRC Press, Book 11 (February 1987).

[3] E. Lim, N. Taksande, and C. Seaman. 2012. A Balancing Act: What Software Practitioners Have to Say about Technical Debt. IEEE Softw., Vol. 29, No. 6, 22–27, 2012.

[4] Z. Li, P. Avgeriou, and P. Liang. 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw., Vol. 101, 193–220 (Mar. 2015).

[5] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou. 2015. The financial aspect of managing technical debt: A systematic literature review. Inf. Softw. Technol., Vol. 64, 52–73 (Aug. 2015).

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

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

[8] Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou. 2016. A financial approach for managing interest in technical debt. In Business Modeling and Software Design, Lecture Notes in Business Information Processing, Springer International Publishing, 117-133. DOI: 10.1007/978-3-319-40512-4.

[9] N. Ramasubbu and C. F. Kemerer. 2014. Managing Technical Debt in Enterprise Software Packages. IEEE Trans. Softw. Eng, Vol. 40, No. 8, 758-772 (August 2014).

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

[11] Ernst N., 2012. On the role of requirements in understanding and managing technical debt. *3rd International Workshop on Managing Technical Debt (MTD '12)*. IEEE.

[12] Zazworka N., Seaman C., Shull F., 2011. Prioritizing design debt investment opportunities. *2nd International Workshop on Managing Technical Debt (MTD' 11).* ACM.

[13] Guo Y., Seaman C., 2011. A portfolio approach to technical debt management. *2nd International Workshop on Managing Technical Debt*. ACM.

[14] Snipes W., Robinson B., Guo Y., Seaman C., 2012. Defining the decision factors for managing defects: A technical debt perspective. *3rd International Workshop on Managing Technical Debt (MTD' 12)*. IEEE Computer Society.

[15] Guo Y., Seaman C., Gomes R., Cavalcanti A., Tonin G., da Silva F., Santos A.L., Siebra C., 2011. Tracking technical debt - An exploratory case study. *27th International Conference on Software Maintenance (ICSM '11)*. IEEE Computer Society.

[16] Chin S., Huddleston E., Bodwell W., Gat I., 2010. The Economics of Technical Debt, *Cutter IT Journal*.

[17] Curtis, J. Sappidi, and A. Szynkarski. 2012. Estimating the size, cost, and types of Technical Debt. In 3rd International Workshop on Managing Technical Debt (MTD '12). IEEE, Zurich, Switzerland, 49 – 53.

[18] J. L. Letouzey and T. Coq. 2010. The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code. In Second International Conference on Advances in System Testing and Validation Lifecycle (VALID '10). IEEE, Nice, France, 43-48.

[19] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun ,and K. Systa. 2016. The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study. In 8th International Workshop on Managing Technical Debt (MTD '16). IEEE, Raleigh, NC, USA, 9-16.

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

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

[22] A. Ampatzoglou, A. Gkortzis, S. Charalampidou, and P. Avgeriou. 2013. An embedded multiple-case study on oss design quality assessment across domains. In Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM ' 13). ACM/IEEE, Baltimore, MA, USA, 255-258.

[23] P. Runeson, M. Host, A. Rainer, and B. Regnell. 2012. Case study research in software engineering: Guidelines and examples (1st ed.). John Wiley & Sons.

[24] 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, re-affirmed Dec. 2009).