Noname manuscript No.
(will be inserted by the editor)

A Clustering Approach towards Cross-project Technical Debt Forecasting

Dimitrios Tsoukalas · Maria Mathioudaki · Miltiadis Siavvas · Dionysios Kehagias · Alexander Chatzigeorgiou

Received: date / Accepted: date

Abstract Technical debt (TD) describes quality compromises that can yield short-term benefits but may negatively affect the quality of software products in the long run. A wide range of tools and techniques have been introduced over the years in order for the developers to be able to determine and manage TD. However, being able to also predict its future evolution is of equal importance in order to avoid its accumulation, and, in turn, the unlikely event of making the project unmaintainable. Although recent research endeavors have showcased the feasibility of building accurate project-specific TD forecasting models, there is a gap in the field regarding cross-project TD forecasting. Cross-project TD forecasting is of practical importance, since it would

Dimitrios Tsoukalas (Corresponding Author)
Centre for Research and Technology Hellas, Information Technologies Institute, Thessaloniki, Greece
University of Macedonia, Department of Applied Informatics, Thessaloniki, Greece
E-mail: tsoukj@iti.gr
ORCID: https://orcid.org/0000-0001-9986-0796

Maria Mathioudaki
Centre for Research and Technology Hellas, Information Technologies Institute, Thessaloniki, Greece
E-mail: mariamathi@iti.gr

Miltiadis Siavvas
Centre for Research and Technology Hellas, Information Technologies Institute, Thessaloniki, Greece
E-mail: siavvasm@iti.gr
ORCID: https://orcid.org/0000-0002-3251-8723

Dionysios Kehagias
Centre for Research and Technology Hellas, Information Technologies Institute, Thessaloniki, Greece
E-mail: diok@iti.gr

Alexander Chatzigeorgiou
University of Macedonia, Department of Applied Informatics, Thessaloniki, Greece
E-mail: diok@iti.gr

enable the application of pre-existing forecasting models on previously unknown software projects, especially new projects that do not exhibit sufficient commit history to enable the construction of project-specific models. To this end, in the present paper we focus on cross-project TD forecasting, and we examine whether the consideration of similarities between software projects could be the key for more accurate forecasting. More specifically, we propose an approach based on data clustering. In fact, a relatively large repository of software projects is divided into clusters of similar projects with respect to their TD aspects, and specific TD forecasting models are built for each cluster, using regression algorithms. According to our approach, previously unknown software projects are assigned to one of the defined clusters and the cluster-specific TD forecasting model is applied to predict future TD values. The approach was evaluated through several experiments based on real-world applications. The results of the analysis suggest that the proposed approach comprises a promising solution for accurate cross-project TD forecasting.

Keywords technical debt · technical debt forecasting · cross-project prediction · data clustering

1 Introduction

Technical Debt (TD), a term that has been inspired by the financial debt of economic theory, has been introduced by Ward Cunningham in 1922 [14] to describe in monetary terms the cost of additional software maintenance effort caused by technical shortcuts taken usually in favor of shorter time-to-market. Initially, TD was linked only to the software implementation process but soon enough it was extended to the whole software development cycle [8]. As in financial debt, TD incurs interest payments in the form of increased future costs owing to the earlier quick and dirty design and implementation choices. In that sense, TD can be considered as a precious tool that enables the understanding of when a software product becomes unmaintainable.

Numerous techniques, methods, and tools have been proposed over the last years for managing TD, providing a variety of options to the developers and project managers of software applications [40]. These approaches however, are mostly focusing on estimating the TD of a software project at its current state. On the other hand, predicting the future TD during the software evolution is a new emerging field of study that can be considered equally important, as the software system and its TD emerge in parallel [17]. TD forecasting could give project managers and developers the opportunity to timely react on the accumulation of TD by employing appropriate repayment activities (e.g., code refactoring) promptly [63] and thus, maintain a software application to a satisfactory quality level and avoid the unlikely event of reaching a point at which the software project becomes no longer maintainable. Whereas various researchers have addressed the topic of forecasting the evolution of various aspects directly or indirectly related to the TD of a software project, such as code smells [20], fault-proneness [4] and evolution trends [10], not too many

concrete approaches have been proposed so far regarding the forecasting of TD itself. Hence, it is of paramount importance to produce and develop tools that will enable on-time decision making by predicting TD's future evolution.

In our previous work [60,61], we have initially approached the TD forecasting challenge by employing statistical time series models, which have been widely used for predicting software evolution trends. In addition to statistical time series, we have also introduced and investigated other more sophisticated approaches that can be applied for TD forecasting, such as machine learning (ML) and causal models. The results of our studies revealed that the proposed approaches are suitable and are able to provide accurate results. However, these approaches assume that reliable and sufficient historic data does exist in order for the models to be applied to a specific software project and provide accurate forecasts. More specifically, we expect that a rather long history of past commits of a given project is available for initiating a proper model-training process, as opposed to short series of historical data, which expose limited capacity for training forecasting models, due to insufficient information that they provide. This obviously affects the practicality of the produced models, as they cannot be applied to new software projects without a sufficient number of past commits. Cross-project TD forecasting, that is, building a forecasting model based on data retrieved from one project and using it to get reliable forecasts for a new, previously unknown software project, would allow project managers and developers leverage the benefits of TD forecasting in cases where a long history of commits is not available, e.g., from the very early stages of the development. To this end, in the present paper, in order to overcome the aforementioned problem, we introduce a novel clustering-based approach that aims to improve the performance of TD forecasting models in cross-project prediction. More specifically, according to our approach, a relatively large repository of software projects is utilized and its projects are grouped into clusters based on their similarity with respect to important TD aspects (e.g., size, complexity, Object-oriented metrics, etc.). Consequently, TD forecasting models are built using regression algorithms for each one of the produced clusters. When a new project becomes available, it is assigned to one of the defined clusters and the dedicated pre-trained forecasting model that is associated to this cluster is used to predict its future TD. Hence, this approach enables the provision of TD forecasts, for software projects that do not exhibit a sufficiently long history of commits. In other words, it enables accurate TD forecasting from the early stages of software development. In brief, the overall problem that the present work attempts to address can be summarized in the following research question:

RQ: *Can data clustering improve the accuracy of cross-project TD forecasting?*

For the purposes of the present study, a relatively large repository of real-world open-source software projects is utilized. This repository is used as the basis for the construction of the clusters, as well as of the TD forecasting

models for each cluster. It is also utilized for the evaluation of the ability of the proposed approach to provide accurate cross-project TD forecasting.

The cross-project TD forecasting approach that is proposed in the present paper is part of the SDK4ED[1] European project. The main goal of the SDK4ED project is to minimize cost, development time, and complexity of low-energy software development processes, by providing a set of innovative solutions (i.e., toolboxes) integrated into the form of an easy-to-use platform for automatic optimization and trade-off calculation among important design-time and run-time software quality attributes. More specifically, the outcome of the project so far showcases numerous research and technical achievements with respect to the optimization of the targeted quality attributes, namely Maintainability [16,3,37,5,50], Dependability [54,53,51,52,32,33], Energy Consumption [67, 42,24,23], as well as Quality Forecasting [63,60,61] and Decision Support [47, 55–57] throughout the overall software development cycle. The SDK4ED TD Forecasting tool, integrated into the TD Management (TDM) framework, aims to provide predictive forecasts regarding the evolution of the TD quality attribute. As mentioned previously, the existing work that forms the basis of this tool is built upon methods that assume the availability of a long history of past commits of a given project for initiating a proper model-training process. Therefore, the cross-project approach introduced in this paper aims to set the foundations towards covering the existing gap in the field by proposing a method able to deliver cross-project TD forecasts and therefore assist practitioners (i.e., SDK4ED users) in performing TD management activities from the very early stages of the development.

The remainder of this paper is structured as follows. Section 2 discusses the related work in the field of TD forecasting and cross-project prediction, focusing on the open issues that the present work attempts to address. In Section 3, the proposed methodology is described, along with experimental setup, whereas in Section 4, a discussion of the experimental results is provided. Finally, Section 5 concludes the paper and discusses directions for future work.

2 Related Work

2.1 TD Forecasting

Nowadays, software companies have numerous methods at their disposal, usually accompanied by tools, that allow them to estimate and manage the TD of the software applications they develop. However, besides managing the TD of a software application at its current state, gaining an insight into its future evolution is considered equally important, as such information could allow developers and project managers to prioritize and repay TD items not only based on their current TD values, but also based on their potential future TD accumulation. This section summarizes and discusses the research work that has been conducted until today in the TD forecasting field.

[1] https://sdk4ed.eu/

In his popular study about the laws of software evolution, Lehman stated that software systems need to constantly evolve over time to adapt to the environment [38]. However, an evolving software is almost unavoidably characterized by the tendency of its complexity to increase and its quality to decrease. On this account, the need to analyze, understand and predict the evolution of a software system has increased considerably in the last years [26] and nowadays constitutes an active and challenging area of research [22]. Within this scope, evolution models can be proven to be a valuable tool, since they can provide estimates on the evolution of a software product and consequently, an insight for its quality as well. Towards this effort, various researchers have addressed the topic of forecasting the evolution of various aspects directly or indirectly related to the TD of a software project, such as code smells [20], fault-proneness [4,28,27,35], software defects [48,45], vulnerabilities [49], and evolution trends [10,68,34].

In a recent study by Chaikalis and Chatzigeorgiou [10], the authors attempt to forecast the software evolution trends of 10 open-source Java projects by employing Network Models. The results of their approach demonstrate a promising model accuracy when predicting various network and software properties. Time series models are also a widely-used method of software evolution analysis. In a relevant study, Yazdi et al. [68] apply autoregressive moving average (ARMA) time series to predict future changes of various software systems. The authors conclude that time series models, if tuned correctly, can generate forecasts with a sufficient accuracy. In a similar study by Raja et al. [48], the authors employ time series analysis for defect prediction during the software evolution of eight open source projects. Their approach leads to the conclusion that time series models constitute an effective way to support time- and budget-related decision making activities. Likewise, Goulao et al. [28] try to forecast the evolution of change requests of the Eclipse application by using time series models. To do so, they model the evolution's seasonal patterns and conclude that using seasonal information leads to a significant improvement of the estimation accuracy. Finally, Kenmei et al. [34] employ time series models for future changes' forecasting. They validate their produced models on three open source applications and come to the conclusion that time series are suitable for predicting change requests and therefore can be proven to be a valuable tool for project planning.

Besides time series analysis, ML techniques are also employed by a multitude of researchers while approaching the challenge of modelling the evolution of various aspects of a software project. Chug and Malhotra [13] compare 17 ML techniques for predicting the maintainability of seven open source systems, using OO metrics as predictors. They conclude that genetically adaptive learning models outperform the rest of the investigated models. In another study by Elish and Elish [18], the authors compare different ML algorithms for maintainability prediction and conclude that the TreeNet model significantly improves of the estimation accuracy. In a study by Fontana et al. [20], the authors employ 16 supervised ML techniques to detect code smells on 74 software projects. They report that J48 and Random Forest algorithms demonstrate

the highest performance. In their study, Arisholm et al. [4] propose a multivariate model for fault-proneness prediction using historical change and fault data obtained from various object-oriented systems. Similarly, Gondra et al. [27] train an ANN based on software metrics to predict future fault-proneness. In another study, Nagappan et al. [45] accurately predict the probability of post-release defects using regression models trained on code metrics. Finally, in their study [35], Khoshgoftaar et al. employ classification and regression trees (CART) to distinguish between non-fault and fault-prone modules on large scale legacy software projects, concluding that their approach results in satisfactory accuracy.

The multitude of research endeavors aimed at addressing the challenge of predicting various software quality evolution aspects reveals the importance of this topic in the software engineering community. Nevertheless, as a software system evolves, so does the TD that accompanies it. While several researchers have studied the evolution of TD and its implications for the software development process [17, 16, 59, 2, 12, 3], the small amount of existing research works on TD evolution forecasting [60, 61, 58] highlights the gap in the field.

The significance of developing forecasting methods for accurate prediction of TD evolution as well as their potential impact while integrated into accompanying tools has been brought to light by a related study [63], in which the authors state that it would be interesting to examine if such methods could result in the development of higher-quality software products. In a first study towards addressing this challenge, a study by Skourletopoulos et al. [58] proposes a method for predicting the TD of Software as a Service (SaaS) applications using the COCOMO software cost model [7]. In another study by Tsoukalas et al. [60], the authors use time series analysis to model and forecast TD evolution. They evaluate their approach on long-lived, open-source software projects and conclude that a single autoregressive integrated moving average (ARIMA) model can generate accurate forecasts over a fairly long period. However, they notice that the performance of their model decreases significantly for forecasting horizons longer than eight weeks. In an attempt to present a more complete approach on the concept of TD forecasting, the same authors perform a new study [61] where they empirically evaluate ML methods for their ability to predict the evolution of TD, based on a dataset of fifteen open source projects. To do so, they investigate a large set of popular ML algorithms for various forecasting horizons. Through their study, they observe that while linear models are better candidates for shorter forecasting horizons, the non-linear Random Forest regression achieves better performance for longer forecasting horizons. They conclude that ML models constitute a suitable and trustworthy TD forecasting method, able to capture the future value of TD with a sufficient level of accuracy.

2.2 Cross-project Prediction Approaches

To investigate the effectiveness of cross-project validation, several studies have been published over the years, indicating that this is an active and challenging area of research. Towards an attempt to improve generalizability of the software maintainability prediction, Malhotra and Lata [41] propose a cross-project approach to predict software maintenance effort. To meet their objective, the authors use three open source projects written in java language and evaluate the performance of their models using statistical tests and conclude that cross project validation can be successfully applied to predict software maintenance effort of open source software. In the field of cost estimation, Kitchenham et al. [36] perform a systematic review of studies that compared predictions from cross-company cost models to predictions from within-company cost models based on analysis of project data. Their results however are inconclusive, as their research shows that some organizations would benefit from cross-company models while others would not.

Regarding cross-project defect prediction, Turhan et al. [64] investigate the applicability of cross-company data for building localized defect prediction models using static code features. They observe that models trained on within-company data outperform the ones trained on cross-company data. However, their analysis also shows that models trained using Nearest Neighbor on cross-company data are performing similarly to within-company models. In a similar study, Canfora et al. [9] propose an approach for cross-project defect prediction based on a multi-objective logistic regression model. They evaluate their model on 10 datasets and conclude that the results indicate the superiority and usefulness of the multi-objective approach. Similarly, Watanabe et al. [66] use metrics and bug data computed from C++ and Java projects to build cross-project models for fault prediction and evaluate the results based on open source projects. In another study, He et al. [30] investigate cross-project defect predictions by conducting three large-scale experiments on 34 datasets and conclude that in the best cases, training data from other projects can provide better prediction results than training data from the same project. Furthermore, they note that cross-project predictions are related to the distributional characteristics of datasets which are used for training. Finally, Menzies et al. [44] propose a clustering approach to perform cross-project or within-project defect prediction. Their approach clusters together similar classes of different projects and then, classes of the same cluster are used to train a prediction model, which is then used to predict defect-proneness of classes belonging to another project of the same cluster. The results indicate that cross-project prediction performance on classes belonging to the same cluster is typically superior when compared to that of classes belonging to another cluster.

Existing TD forecasting models have not demonstrated satisfactory results in cross-project forecasting. Under those circumstances, taking advantage of cross-project forecasting approaches in order to forecast the TD evolution of projects with limited historical data is of paramount importance. Through this work, we aim to examine whether a clustering approach could be the key for

more accurate cross-project TD forecasting. Such a contribution would enable software companies to support decision-making early enough in the software development process and arrange accurate payback plans to manage TD on time and avoid unforeseen conditions long-term.

3 Methodology and Experimental Setup

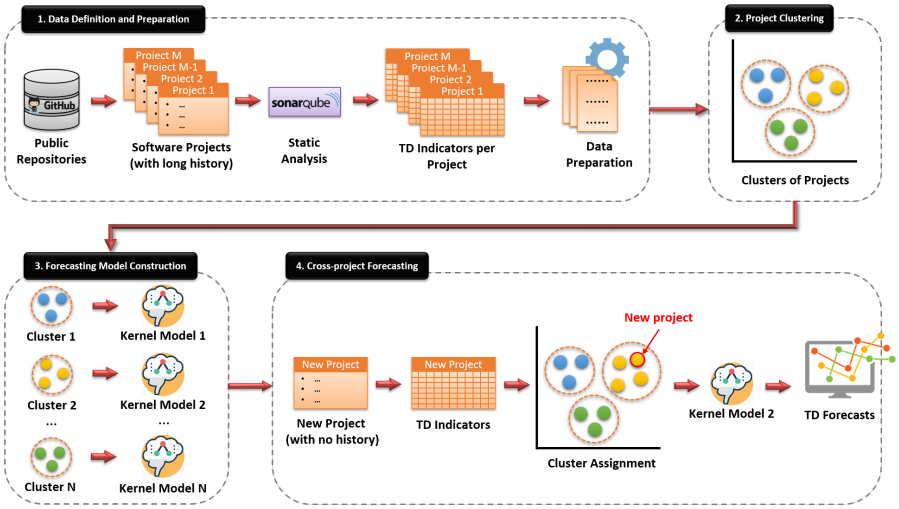

Fig. 1 Paper Roadmap

In Figure 1, a high-level overview of the overall approach that is adopted in the present paper for examining whether clustering of software projects could be the key for accurate cross-project TD forecasting is illustrated. As can be seen in Figure 1, the overall approach comprises four steps, which are briefly described below:

1. **Data Definition and Preparation**: During the first step of our approach, the source code of multiple software projects is retrieved from a code repository along with its history (i.e., past commits). Subsequently, the source code of the retrieved projects is analyzed using a static analysis platform and several TD measures are calculated for the construction of the initial dataset. Next, a data preparation process follows, which is responsible for applying feature selection techniques and bringing the dataset in a form ready to be used for model training purposes.
2. **Project Clustering**: During the second step of our approach, a selected clustering algorithm takes as input the dataset of analyzed projects, as produced during Step 1, and after finding the optimal number of clusters, it creates groups based on project "similarity" measures. These measures

include various software metrics and TD indicators, such as size, complexity, code smells and bugs, which were computed during the static analysis performed in Step 1.

3. **Forecasting Model Construction**: During the third step of our approach, for each cluster generated during Step 2, the TD-related data of the projects assigned into the specific cluster are used to train a single cluster-representative TD forecasting model, which within the context of this work is referred to as "kernel" model. Therefore, the outcome of this step is the creation of N kernel models, where N is the total number of clusters produced by the clustering algorithm.
4. **Cross-project Forecasting**: During the fourth and final step of our approach, we assume that a new, previously unseen project becomes available. After performing the required static analysis in order to extract its TD measures, the clustering algorithm assigns it to the appropriate similarity cluster and the dedicated pre-trained TD forecasting model for that cluster (produced during Step 3) is used to predict its future TD without requiring prior knowledge and historical data.

More details about each step of the proposed approach are provided in the rest of this section.

3.1 Dataset

As can be seen in Figure 1, the first step of the proposed approach starts with the construction of the dataset, that is, a relatively large code repository comprising the source code of multiple software projects along with their history (i.e., past commits). Subsequently, the source code of the retrieved projects needs to be analyzed using a static analysis platform in order to extract TD measures that will be used as input to the clustering algorithm.

For the purposes of the present study, we utilized a TD dataset provided by Lenarduzzi et al. [39]. This dataset is publicly available[2] and contains TD measurement data from 33 real-world open-source Java projects retrieved from the Apache Software Foundation. To collect TD measurements, the authors analyzed all available commits for each selected project using SonarQube[3], a popular open source platform that offers continuous code quality inspection and TD measurement mechanisms. Each project in the original dataset is provided as a csv file. The columns of the csv file correspond to 61 software-related metrics for a specific commit, as extracted from SonarQube across the whole commit history of each application. Respectively, each row of the csv file represents a specific commit of that project. For the present work, 27 of these projects were used during the first three steps of the approach, while three of the remaining projects were randomly selected and used during the last step of the approach as case studies in order to evaluate its usefulness

[2] https://github.com/clowee/The-Technical-Debt-Dataset
[3] https://www.sonarqube.org/

in practice. The 27 projects that were utilized for clustering and forecasting model construction purposes (i.e., Step 1-3) are presented in Table 1, along with additional information regarding the total number of analyzed commits and the analysis timeframe.

Table 1 Projects of the TD Dataset

Project Name	**SonarQube**	
	Analyzed Commits	**Analysis Timeframe**
Atlas	2336	12/14 - 06/18
Aurora	4012	04/10 - 06/18
Batik	2097	10/00 - 06/06
Beam	2865	12/14 - 07/16
Commons-bcel	1324	10/01 - 04/18
Commons-beanutils	1192	03/01 - 06/18
Commons-cli	896	06/02 - 02/18
Commons-collections	2982	04/01 - 09/18
Commons-configuration	2895	12/03 - 05/18
Commons-daemon	980	09/03 - 08/11
Commons-dbcp	1861	04/01 - 06/18
Commons-digester	2143	05/01 - 08/17
Commons-exec	617	08/05 - 01/16
Commons-fileupload	922	03/02 - 10/17
Commons-io	2118	01/02 - 06/18
Commons-jexl	1551	04/02 - 05/18
Commons-jxpath	597	08/01 - 11/15
Commons-net	2088	04/02 - 08/17
Commons-ognl	608	05/11 - 09/13
Commons-validator	1339	01/02 - 04/18
Commons-vfs	2067	07/02 - 05/18
Felix	596	08/05 - 10/06
Httpcomponents-client	2867	12/05 - 06/18
Httpcomponents-core	1941	02/05 - 08/17
Mina-sshd	1370	12/08 - 06/18
Santuario	2697	10/01 - 06/18
Zookeeper	411	05/08 - 06/18

Within the context of this study, we did not exploit all 61 software-related metrics provided by SonarQube. Instead, we decided to narrow down the number of features by focusing only on those that act as the most prominent TD indicators. TD indicators are indicators usually related to software metrics [40, 1] that allow researchers and practitioners to point out TD-related quality issues residing in various attributes, features, or characteristics of a software artefact. Recent literature has identified a variety of software metrics that can act as potential TD indicators, while there is also a multitude of assessment tools [63] that support TD estimation based on these metrics. For instance, metrics such as cyclomatic complexity, code duplication or low test coverage (i.e. the absence of sufficient code tests) have been widely used to predict software maintainability decay and, in turn, the TD of a software application [25, 29, 43]. In addition, code issues that can negatively affect the quality of a soft-

ware, such as code violations or bugs, have also been studied for their impact on TD [65,69,31]. Finally, code smells, i.e., sub-optimal design solutions that violate at least one programming principle [21] are considered in the literature as one of the first indications of the presence of TD [1,46]. Code smells represent serious threats to the software maintenance process and impose the need for code refactoring [19]

In Table 2 we present the SonarQube metrics that were finally selected based on their ability to act as TD indicators. These metrics will be used as independent variables during the next steps of the methodology. In addition to the independent variables, the table also introduces the target variable, i.e., the variable we try to forecast, denoted as *total_principal* (in bold). In brief, *total_principal* is defined as the effort (in minutes) to fix all issues and is computed as the sum of code smell, bug, and vulnerability remediation effort.

Table 2 Brief description of the SonarQube metrics that were used as TD indicators

Metric	Description
Technical Debt Metrics	
sqale_index	Effort in minutes to fix all Code Smells.
reliability_remediation_effort	Effort in minutes to fix all bug issues.
security_remediation_effort	Effort in minutes to fix all vulnerability issues.
total_principal	**Effort in minutes to fix all issues. The sum of code smell, bug and vulnerability remediation effort**
Reliability Metrics	
bugs	Total number of bug issues of a project
Maintainability Metrics	
code_smells	Total number of code smell issues of a project.
Size Metrics	
comment_lines	Total number of lines that correspond to comments.
ncloc	Total number of lines that are not part of a comment.
Security Metrics	
vulnerabilities	Total number of vulnerability issues of a project
Complexity Metrics	
complexity	Total Cyclomatic Complexity calculated based on the number of paths through the code.
Coverage Metrics	
uncovered_lines	Total number of code lines that are not covered by unit tests.
Duplication Metrics	
duplicated_blocks	Total number of lines that belong to duplicated blocks.

3.2 Data Preparation

3.2.1 Feature Selection

After acquiring the data needed, in order to proceed with further analysis, the next step is the preparation of the dataset in order to transform it in a form

ready to be used for clustering and model training purposes. Therefore, the data preparation step first involves feature selection techniques and then the "sliding window" method, which is responsible for reframing the dataset in a form suitable for supervised ML problems.

The selection of independent (i.e. input) variables before designing or experimenting with an ML algorithm is a task that needs to be treated with special attention. A large number of input features, i.e. a high dimensional feature space, may lead to the "curse of dimensionality" [6]. According to this phenomenon, the increasing number of the model's inputs leads to a degradation in its predictive performance. Features that are not associated (or they are partially associated) with the target variable may negatively affect the model accuracy. Hence, after building our dataset, a clear understanding of the statistical significance of the TD indicators needs to be gained, and lastly the feature selection algorithms need to be employed in order to reduce the number of the model's inputs by keeping only the TD indicators (described in Table 2) that are statistically important to act as TD predictors.

In order to examine which of the TD indicators are statistically significant for predicting the TD, four different feature selection algorithms were employed. More specifically, two filter-based methods were used, namely Spearman Correlation and F-test, one wrapper-based method named Recursive Feature Elimination (RFE), and finally one embedded method named Tree-based Elimination (TBE). Generally, filter-based methods filter the features based on a few metrics. On the other hand wrapper-based methods consider the selection of a set of features as a search problem. Finally, Embedded methods use algorithms that have built-in feature selection methods (such as Lasso and Random Forest).

- The **Spearman correlation** is a non-parametric approach used to quantify the monotonicity of the relationship between the values of two given datasets. As it is in other correlation techniques such as Pearson, Spearman correlation coefficients take values between -1 and +1. However, as a non-parametric test, Spearman correlation does not assume any distribution for the studied data. A coefficient value of -1 or +1 suggests monotony, whereas a coefficient value of 0 suggests no correlation at all. In this work, we explore the absolute values of the Spearman correlation coefficients between the independent (TD Indicators) and dependent (TD Principal) variable in our dataset, and we rank the former based on these values. Then, we select the top N features based on the results of this test.
- The **F-test** technique is a univariate linear regression test that employs a linear method to test the relationship of each of various independent features. This method consists of two main steps. First, the correlation between each indicator and the target variable is calculated. Then, the result is transformed into an F-score and then into a p-value. A relatively low p-value indicates the existence of a relationship between the examined variables, whereas a relatively high p-value indicates that there is no relationship. The F-test method calculates the correlation between an

independent variable and the dependent variable and thus, it can be applied in order to determine and exclude the features that are likely to be insignificant for the evolution of the target variable and as a result, unsuitable to be used as TD indicators. In our occasion, the p-values between the independent variables (TD Indicators) and the dependent variable (TD Principal) are calculated and in this way we keep the top N features.
- The **Recursive Feature Elimination (RFE)** method, as its name implies, is a feature selection method that recursively eliminates features based on an estimator model trained on the initial set of features. The significance of each feature is implied either by a coefficient attribute (e.g. Logistic Regression) or by a feature-importance attribute (e.g. Random Forest) determined by the nature of the model. As this process evolves, the insignificant features are recursively excluded until we finally end up with only the required number of features. We select Logistic Regression as the estimator model. As a result, the RFE rates the significance of each of the features based on the coefficient given by the decision function of the Logistic Regression estimator and crops the initial dataset until it contains only the top N independent variables in terms of significance.
- **Tree-based Elimination (TBE)** is an embedded method. As the name implies, this method uses models that have built-in feature selection methods to reduce the initial number of features. In this case, oppositely to the RFE method, we use the Random Forest model to determine feature importance based on node uncleanness in each decision tree. Then, the top N features are selected based on the average of all feature importance values calculated by each decision tree.

All of the four aforementioned feature selection algorithms were put in practice using Python in conjunction with the scikit-learn[4] ML library. Each of the algorithms described above was employed independently on the feature set and retained the top $N = 5$ features that were selected by each method. We set $N = 5$ mainly due to the fact that both RFE and TBE methods stopped the feature elimination process when the feature subset reached the number of five features. As a result, selecting the top 5 features from each independent method allowed us to directly compare the selected features' subsets among the four methods. Then, we aggregated the results by ranking each feature based on the number of times it was selected to be in the top 5 of a particular method. On Table 3 we observe the features ranked by the number of times they were selected by each of the algorithms. A value of True in a specific column indicates that the feature of that specific row has been selected to be in the top 5 features of the algorithm of this column.

By inspecting Table 3, it is obvious that *code_smells* and *bugs* are in the top 5 features of every feature selection algorithm. Furthermore, *complexity*, *ncloc*, and *vulnerabilities* are also high in the list since they were selected by three out of the four algorithms. We finally decided to keep the features that were indicated by at least three out of the four feature selection algorithms

[4] https://scikit-learn.org/stable/

Table 3 Results of the four feature selection methods

Feature	Spearman	F-test	RFE	TBE	Total
code_smells	**True**	**True**	**True**	**True**	**4**
bugs	**True**	**True**	**True**	**True**	**4**
complexity	**True**	**True**	**False**	**True**	**3**
ncloc	**False**	**True**	**True**	**True**	**3**
vulnerabilities	**True**	**True**	**True**	**False**	**3**
duplicated_blocks	False	False	False	True	1
uncovered_lines	False	False	True	False	1
comment_lines	False	True	False	False	1

to be among the top five features. This basically means that features below *vulnerabilities* in the table will be excluded from the analysis that follows. To sum up, starting out with eight TD indicators that we examined, five of them turn out to be statistically significant for predicting TD, by three or four feature selection algorithms. At this point, it is worth mentioning that each of the aforementioned feature selection algorithms performs a statistical test in order to decide which of the features should be excluded and which should be included during the selection process. Thus, the statistically significant impact of the final selection of features with respect to the target variable (i.e., *total_principal*) are the result of the independent statistical tests applied by each of the feature selection algorithms. Consequently, the TD indicators that were found to be the most promising TD predictors according to our feature selection approach are *code_smells*, *bugs*, *complexity*, *ncloc*, and *vulnerabilities*. These metrics will be considered as input during the clustering process described in Section 3.3 and during the creation of the TD forecasting models presented in Section 3.4.

3.2.2 Sliding Window Method

As we brought up earlier, the project-specific datasets that will be utilized through this work contain information provided by different indicators, throughout different points in time (i.e., timestamps) of each project. Our goal is to examine the evolution of the dependent variable, i.e., TD, through time. Hence, every sample of the dataset contains information for a specific point in time that is very probable to depend on the past (few) samples. However, the majority of ML models are not able to capture the temporal relationships that may govern the observations through time. Therefore, time series data need to be transformed into an appropriate form for supervised learning before put into practice for ML tasks.

In order to gain a better insight into the need for transforming time series data before providing it as input for ML models, a sample of data gathered in a temporal sequence is shown in Figure 2. The rows represent data samples gathered at specific points in time. Columns 2 to 4 represent features $X1$ to $X3$ accordingly, whereas column $Y1$ represents the dependent variable, i.e., the variable we want to generate forecasts for. By inspecting the table, it be-

comes obvious that the current dataset structure is considered inappropriate for forecasting models training using supervised learning as there are two issues coming to light. First, if the dataset is utilized as it is for the training process, then the model will not be able to take under consideration any past information, as each row contains information only for a specific sample (i.e., point in time). Second, as the dependent variable of each sample holds only the value of that specific point in time, training a model using this format will result in a function that has learned how to make approximations only for that point in time (rather than generate future predictions).

One particular advantage of employing ML models instead of statistical models is their capability to support more than one input features. Taking advantage of this aspect of ML models, a method called "sliding window" was applied [15] on each project-specific dataset in order to bring it in a particular structure that integrates into one row information from multiple prior rows (steps) as inputs (X) in order to generate forecasts for future time steps as output (Y). Briefly, the sliding window expands each initial row of the dataset by including past and future information into one lag. More details on this approach are given below.

	X			*Y*
timestamp	*X1*	*X2*	*X3*	*Y1*
0	10	100	1000	10000
1	20	200	2000	20000
2	30	300	3000	30000
3	40	400	4000	40000
4	50	500	5000	50000
…	…	…	…	…

Fig. 2 Dataset collected in temporal order

The term that describes the number of past rows that we want to incorporate as input in the current lag is called the "window width". In the bibliography it may also be referred to as the "size of the lag". Firstly, we need to determine the sliding window width. On Figure 2, the red box represents the window width which corresponds to the present row (showed by the red arrow) plus a number of past rows. The current lag and the past lags containing past information will be merged into a new single row. For this specific case, the red box in Figure 2 contains the independent variables of the current lag (t) and also one step in the past lag ($t-1$) which will be merged into one new row. The forecasting horizon needs to also be determined. In our case, it is illustrated as the blue box in Figure 2. The blue box in our example indicates that forecasts will be generated for 1 step-ahead. This basically means that the Y value (i.e., the target variable) of $t+1$ lag will be chosen as the target value for the sample that corresponds to lag t. If we wanted to generate

forecasts for 2 steps-ahead, the Y value of $t + 2$ lag would be chosen as the target value, and so on. This process results in a new row, as shown in Figure 3. We repeat the process by "sliding" the two boxes at the same time over the rows, step by step, producing new lags until the window reaches the end of the dataset. When the whole process is over, a new re-framed dataset is produced, which now uses the current and one past step of each independent variable in order to generate forecasts for the target variable for one step ahead. The new dataset is shown in Figure 3.

	X						*Y*
index	***X1(t-1)***	***X2(t-1)***	***X3(t-1)***	***X1(t)***	***X2(t)***	***X3(t)***	***Y1(t+1)***
0	10	100	1000	20	200	2000	30000
1	20	200	2000	30	300	3000	40000
2	30	300	3000	40	400	4000	50000
3	40	400	4000	…	…	..	…

Fig. 3 The reframed dataset after applying the sliding window approach

There are no specific guidelines that can be followed in order to determine the size of the window (i.e., the window width). Most of the times, selecting the number of past steps that will be merged per row is a process that strongly correlates to the number of the independent features, the size of the forecasting horizon and the model that is used itself. Consequently, as the selection of the window width takes place, we need to strike a balance between the model complexity and the forecasting accuracy. As a result, a good plan is to examine different window widths by training a model and check what values lead to better results for various forecasting horizons. The goal throughout this process is to minimize the errors. For example, we discovered that the optimal number to use as the size of the lag is 2. This provided us with the minimum Mean Absolute Percentage Error (MAPE) when forecasting takes place for 5 steps ahead, for most of the application-specific datasets when different models were tested. If more lags were added, the complexity of the models increased, with no remarkable effect on the model performance. Accordingly, for larger forecasting horizons, a larger window width was noticed to be more appropriate and as a result we observed improved model accuracy.

We re-framed each dataset using the sliding window method, taking into account the forecasting length we wanted to test our models for, in order to enable the usage of supervised ML. As long as the time series dataset is restructured following this approach and the sequence of the lines is conserved, any of the usual linear and non-linear ML algorithms can be employed.

Another problem that arises from the approach of TD forecasting is the necessity to generate multi-step predictions (i.e., predictions for more than one time-step), as we are interested not only in a single predicted TD value for a particular point in the future, but also in gaining an understanding

of the overall evolution of the TD up to that point. There exist three main ML approaches that enable multi-step forecasts: i) the *Direct* approach, where different individual models are produced to generate forecasts for each forecast lead time, ii) the *Recursive* approach, where one model is produced to generate one-step ahead predictions, and the same model is employed repeatedly using as input the previous prediction to predict the following lead time, and iii) the *Multiple output* approach, where one model with multiple outputs is produced that predicts the whole forecast series simultaneously, i.e., in a one-shot way.

The ML models that we decided to consider in this work do not support more than one output at once thus, we reject the *Multiple* output approach. In addition, and as long as we are interested in multivariate forecasting, i.e., besides the dependent variable we consider also independent variables, we cannot put in practice the *Recursive* approach as this would demand predictions of the independent variables to forecast more than $t + 1$ steps into the future. Thus, we decided to proceed with the *Direct* approach, which means that various independent models will be trained for each of the forecasting horizons. In practice, if we want to generate forecasts for N steps-ahead, the dataset needs to be transformed N times by applying the sliding window method exactly as we presented it before, where every time the target variable Y will lead to a point from $t + 1$ to $t + N$ steps ahead respectively. Thus, N independent models will be developed, each one aiming to predict one future value starting from $t + 1$ up to $t + N$. Lastly, the outputs of the models will be put together into a single array which eventually will hold the future evolution of the TD up to N steps ahead.

3.3 Project Clustering

In the previous section, we first gave a description on the data collection procedure we followed in order to prepare our initial application-specific datasets and then described the process under which the dataset is reframed in order to be ready to be used for clustering and model training purposes. This step of our approach involves the execution of a clustering algorithm that takes as input the refined dataset and creates groups based on project "similarity" measures.

As already mentioned, although TD forecasting models perform well when applied to software projects on which they have been trained (i.e., within-project forecasting), they fail to provide sufficient results in cross-project forecasting. This can probably be explained by the fact that each project exhibits intrinsic characteristics with respect to their type, size, complexity, etc., which may affect the produced models. However, it is reasonable to expect that the application of a TD forecasting model on a project that is highly similar (with respect to the aforementioned intrinsic characteristics) to the project that was used for building the forecasting model, may demonstrate sufficient predictive performance. Clustering is a viable solution for grouping software projects

based on their similarity, and therefore for evaluating the correctness of the aforementioned hypothesis.

Clustering belongs to unsupervised machine learning. Hence, there are no labels we are trying to predict. To date, there exist numerous algorithms dedicated to perform clustering tasks, such as K-means, Agglomerative clustering, Mean Shift and Spectral Clustering, etc., each having its advantages and disadvantages. However, K-means is considered one of the simplest and most commonly-used clustering algorithms. Therefore, we decided to exploit its capabilities for the purposes of this work. K-Means is a centroid-based clustering algorithm. A centroid is basically a point in the center of a cluster. Each cluster is represented by a central vector or a centroid. The centroid point does not necessarily belong to the dataset that is used for clustering. The number of centroids is the number of clusters that we choose. Afterwards, all data points of the dataset we use for clustering are assigned to the closest centroid.

In our case, to proceed with applying K-means for the clustering task, we created a sub-set of our initial dataset in which each of the initial 27 projects is represented by one sample, so as to ensure equal representativeness for the clustering process. To do this, we selected the latest version of each one of the 27 projects in order for each software application in the new dataset to be represented by only one commit. Subsequently, to optimally group the software projects based on their similarity, we used the popular "Elbow" method, a heuristic commonly used in determining the number of clusters in a dataset. After applying the Elbow method, our 27 projects have been assigned to six different groups. We remind the reader at this point that the features we used for the clustering process are the following: *code_smells*, *bugs*, *complexity*, *ncloc*, *vulnerabilities*, and *total_principal*. In Figure 4, we offer a visualization of the six different clusters as they have been formed after applying K-means. For visualization purposes, we used only two features in order to draw the clusters into two dimensions. More specifically, in this particular illustration, the y-axis represents the TD (i.e., *total_principal*), while the x-axis represents a randomly-chosen feature among the ones selected as optimal TD predictors. In this case, this feature is *ncloc*.

In Table 4, the resulting clusters as produced by the K-means algorithm are presented. More specifically, the reader may observe the datasets (i.e., projects) that have been allocated to each cluster.

3.4 Forecasting Model Construction

During the previous step of our approach, a carefully selected clustering algorithm receives as input the refined dataset of analyzed projects and after finding the optimal number of clusters, it creates groups based on TD-related measures. Therefore, as can be seen in Figure 1, the next step involves the construction of representative TD forecasting models (i.e., kernel models) for each cluster that will actually be used to provide TD forecasts on any new project that is assigned to a specific cluster in a later stage. Towards constructing

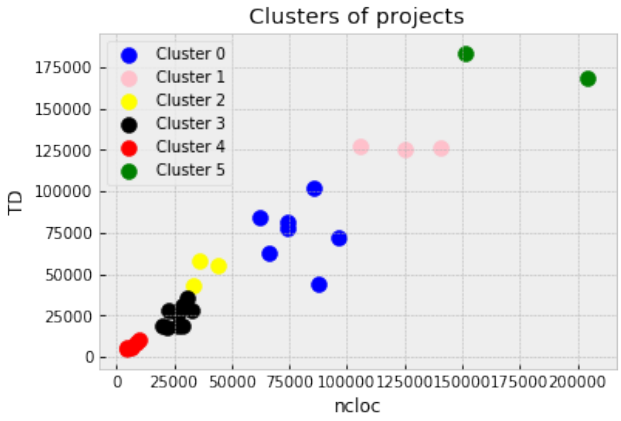

Fig. 4 Clusters of the 27 datasets as they have been formed after K-means clustering with respect to TD and ncloc

Table 4 Visualization of the 6 clusters and the datasets they consist of as generated by k-means clustering algorithm

Cluster 0	**Cluster 1**	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
Felix	*Santuario*	*Commons-beanutils*	*Commons-validator*	*Commons-cli*	*Batik*
Zookeeper	*Aurora*	*Commons-io*	*Commons-vfs*	*Commons-daemon*	*Atlas*
Httpcomponents-client	*Beam*	*Commons-bcel*	*Commons-dbcp*	*Commons-exec*	
Commons-configuration			*Commons-digester*	*Commons-fileupload*	
Mina-sshd			*Commons-jexl*		
Commons-collections			*Commons-jxpath*		
Httpcomponents-core			*Commons-net*		
			Commons-ognl		

kernel models for each cluster, we investigate the performance of a number of ML models trained on a specific project to forecast the TD Principal evolution of the other projects assigned within the same cluster and then, choose the best-performing model, based on its cross-project forecasting performance. The above process can be characterized as cross-project within-cluster forecasting, in a sense that a TD forecasting model trained on a specific project is tested on projects of the same cluster.

To further validate the idea that a kernel model of a specific cluster should provide better cross-project TD forecasting results when used on projects within the same cluster, we will also perform cross-project / cross-cluster predictions, in a sense that a TD forecasting model trained on a specific project

is tested on projects of a different cluster. This will allow us to investigate whether the prediction accuracy of a kernel model is higher when forecasting for projects of the same cluster, compared to forecasting for projects of a different cluster. More details on this experiment, as well as the accompanying result are provided in Section 4.1.

As stated above, within this step of the methodology we investigate the performance of a number of ML models trained on a specific project to forecast the TD Principal evolution of another project. ML techniques are commonly used to make predictions for the future of a variable that interests the researchers. The goal is to maximize model accuracy, which means that the predictions need to be as close to the real values as possible, avoiding the risk of overfitting or underfitting the training data. In this work, we used the Mean Absolute Percentage Error (MAPE) as measure of prediction accuracy of a forecasting method. MAPE is a commonly used measure in order to validate the forecasting performance that takes into account absolute values to quantify the significance of the error expressed as a percentage. MAPE is a reliable measure and has two main advantages. The first one is that the absolute values lead to taking under consideration both the positive and negative errors preventing them from cancelling out each other which allows for no information to be missed. The second one is that due to the fact that respective errors do not correlate to the scale of the target variable, MAPE allows the comparison of forecast accuracy between data that are differently scaled (e.g. different software projects). The equation that describes the MAPE is presented below:

$$MAPE = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right| \tag{1}$$

where A_t is the actual value and F_t is the forecasted value. The difference between the actual and the forecasted value is divided by the actual value. The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points. Multiplying the equation by 100% makes it a percentage error.

Predicting the TD Principal is a regression problem, as this is a continuous variable. Thus, we need to use regression algorithms. In this paper, five different algorithms have been examined for TD forecasting. Three of them are considered linear and the other two non-linear. More specifically, the linear algorithms that were investigated within the context of this work are Linear, Lasso, and Ridge Regression, whereas the nonlinear are Support Vector Regression (SVR) using the Gaussian (rbf) kernel, as well as Random Forest. The majority of these algorithms have been investigated in the literature for their capability to generate predictions for different software quality aspects, such as TD [61] and Maintainability [13,41], or lower-level software attributes, such as code smells [20] and defects [11]. Nevertheless, the task of determining the best-performing ML model per occasion is usually the outcome of a trial-and-error process, as the forecasting performance of any algorithm strongly relies

on the distinctive characteristics of the data (such as size and structure). As a result, we decided to examine various ML algorithms to account for highly diverging data relationships that might rule the different software applications and thus, to overcome the restrictions of different techniques.

Again, for the utilization of the experiments, Python programming language was used and more specifically the scikit-learn ML library. For every algorithm under investigation, a grid search is initially performed in order to determine the optimal values of their parameters for the specific datasets that we have. The models that we chose to employ are described below.

3.4.1 Linear, Lasso and Ridge Regression

Linear regression is a method which explores the existence of linearity between a dependent variable and one or more independent variables. More specifically, given data with a number of variables and one single target variable, the purpose is to find a function that will return the best fit. Supposing that there is a linear relationship between the dimensional variables and the target variable, the model can be described as follows:

$$f(w1, \cdots, wn, b) = y = w \cdot x + b + \varepsilon \tag{2}$$

where w represents coefficients and b is an intercept. The best fit can be found by minimizing the sum of the squared errors in the following way:

$$min \sum_{i=1}^{m} (y_i - \hat{y_i})^2 = \sum_{i=1}^{m} (y_i - (w \cdot x + b))^2 \tag{3}$$

The problem that occurs when a model learns exactly the details in the training data but fails to generate accurate predictions on new data is called "overfitting". In order to avoid this problem a regularization term can be introduced and this is exactly what Lasso and Ridge Regression do. Lasso Regression adds absolute values of the coefficients and thus, coefficients of insignificant variables will become zero. On the other hand, Ridge Regression, instead of adding absolute values, adds squares of the coefficients. Hence, the bigger the coefficient of a feature, the bigger the loss will be. Here we present Lasso (top) and Ridge (bottom) Regression formulas.

$$\sum_{i=1}^{m} (y_i - (w \cdot x + b))^2 + C \sum_{i=1}^{m} |w_j| \tag{4}$$

$$\sum_{i=1}^{m} (y_i - (w \cdot x + b))^2 + C \sum_{i=1}^{m} |w_j^2| \tag{5}$$

However, the reader should keep in mind that Lasso and Ridge Regression do not necessarily provide us with better results than Linear Regression. This strongly depends on the characteristics of the different datasets.

3.4.2 Support Vector Regression – Radial Basis Function

Support Vector regression is a type of Support vector machine that supports linear and non-linear regression. In our case we choose non-linear as we have another three linear algorithms. More specifically we will use RBF (Radial Basis Function). RBF is a function with respect to the origin or a certain point c, i.e., $\phi(x) = f(||x - c||)$ where the norm is usually the Euclidean norm but can be other type of measure.

The RBF learning model assumes that the dataset $D = (x_n, y_n), n = 1 \cdots N$ influences the hypothesis set $h(x)$, for a new observation x in the following way:

$$h(x) = \sum_{i=1}^{N} w_x \times exp(-\gamma||x - x_n||^2) \tag{6}$$

which means that each x_i of the dataset influences the observation in a gaussian shape. Of course, if a data point is far away from the observation its influence is residual (the exponential decay of the tails of the gaussian makes it so). It is an example of a localized function ($x \to \infty \Rightarrow \phi(x) \to 0$).

3.4.3 Random Forest

A Regression Tree constitutes an alternative of decision trees that is constructed during a repeated procedure of separating the dataset into the decision branches. This procedure is called "binary recursive partitioning". An improved type of the Regression Tree is Random Forest (RF) paradigm. RF is a group of RTs which are trained by using the "bagging" approach. Bagging chooses random values over and over again by restoring the dataset and fits trees to these values. When the training process is completed, predictions for unknown data are produced by calculating the midpoint of the forecasts, or by taking the general contribution of all the independent RTs. Two are the main key benefits of RF. First, its accountability and second, its ability to learn complicated, highly non-linear relationships. Nonetheless, RF models require relatively long time to train and need more memory compared to the rest of the models that we consider. Last but not least, RF models are vulnerable to extensive overfitting because of the nature of decision trees.

3.5 Cross-project Forecasting

At this point it is considered valuable to provide some clarifications with respect to main terms that are used in this paper, in order to avoid confusion in the rest of the text. With the term "within-project forecasting" we refer to the process of building and investigating the performance of models dedicated to predicting the evolution of the TD Principal in the software applications on which they have been trained. This means that the investigated models are

trained on the data of a specific project, and evaluated based exclusively on data retrieved from the corresponding project. On the other hand, in "cross-project forecasting", whose applicability we aim to assess within the context of this work, emphasis is given on the ability of a given pre-trained TD forecasting model to accurately predict the future of the TD Principal in previously unknown software projects (i.e., software projects that were not used for the model training).

In the same manner, the term "within-cluster forecasting" refers to the procedure of constructing TD forecasting models on software projects assigned to a specific cluster to predict the future of the TD Principal in projects that are assigned to the same cluster. On the contrary, "cross-cluster forecasting" refers the process of using TD forecasting models trained on software projects of a specific cluster to forecast the TD evolution of projects assigned to different clusters. In order to ensure the ability of the proposed clustering approach to lead to satisfactory cross-project TD forecasts, the performance of the produced ML models is evaluated both in a within-cluster and in a cross-cluster manner in Section 4. In brief, in order for the proposed approach to be meaningful, the produced ML models should demonstrate lower errors in within-cluster evaluation. Hence, based on the above description, both the within-cluster and cross-cluster cases deal with cross-project TD forecasting, which is the main focus of the present work.

4 Results and Discussion

4.1 Within-cluster and Cross-cluster Evaluation

Having assigned all 27 projects to 6 clusters, in this section we aim to identify whether or not datasets that belong to the same cluster provide lower errors than datasets that belong to different clusters when used for cross-project TD forecasting. In order to investigate this assumption, experiments were employed for all combinations of the datasets. This means that each dataset was used for training a forecasting model and then, this model was used to generate predictions for all the rest of the 26 datasets. The reader may assume that for these 27 projects, when a specific algorithm is used, 27x27 combinations (excluding the cases of the pairs referring to the same projects) are formed, thus 702 performance measurement values (MAPE values) are obtained. As stated in Section 3.4, five algorithms were employed for TD forecasting, namely Linear Regression, Lasso Regression, Ridge Regression, SVR and Random Forest. Hence, in total, we obtained 702*5=3510 MAPE values for all of the algorithms for five steps (commits) ahead. For reasons of brevity, we will not present the results for all possible combinations here. In Table 5, we present a fragment of the MAPE values for the case of Lasso Regression used to generate forecasts for five steps ahead so that the reader may have a quick look on a sample of the results. Detailed results for every algorithm can be found at the online Appendix [62]. More specifically, the MAPE values for SVR, Random For-

est, Linear Regression, Lasso Regression and Ridge Regression can be found in Table 24, Table 25, Table 26, Table 27, and Table 28 (online Appendix) respectively.

To complement the forecasting performance evaluation, besides forecasting for 5 steps (commits) ahead, cross-project experiments were also extended for 10 steps ahead. However, since we believe that using the results of 5 steps-ahead forecasts is enough to answer the within-cluster and cross-cluster assumption that we aim to evaluate in this section, we decided to limit the 10 steps-ahead experiments only for those combinations of datasets that belong to the same cluster (within-cluster experiments). Thus, for these results we do not provide five tables as the case of five steps ahead results. Instead, we provide one table per each cluster, including within-cluster results for each algorithm. As we have six clusters and five possible algorithms there are 30 such tables. The reader may find these tables at the online Appendix [62] (Table 29 - Table 58).

Within the context of cross-project TD forecasting, training was performed on the entire dataset of the project that was chosen each time, while testing was performed also on the entire dataset of each project acting as testing dataset. This means that forecasts were generated at every sample (commit) of the testing dataset, throughout its whole evolution. In addition, since we aim at generating predictions for 5 and 10 steps (commits) ahead by following the *direct* approach described in Section 3.2.2, predictions are generated first for 1 step ahead, then for 2 steps ahead, and so on. Subsequently, the predictions are aggregated into a common vector. This way, 5 or 10 different values (depending on the chosen horizon) are generated which represent the forecasts for 5 and 10 steps ahead respectively at a specific point of the testing dataset. These 5 or 10 values are then compared to the real values using the MAPE and a mean value of the MAPEs is calculated. This way, N such MAPEs are obtained for each testing dataset, where N is the number of points in the test dataset (i.e., number of project commits). The mean value of these N errors is calculated so that we have one single value representing the performance of the model when tested on each testing dataset.

To facilitate the readability of the tables, we define three categories of MAPE values to make it easier for the reader to examine and compare the results. MAPE values between 0-20% are marked with three asterisks (***), MAPE values between 21-50% are marked with two asterisks (**) and finally MAPE values between 51-80% are marked with one asterisk (*). MAPE values higher than 81% are marked with no asterisk at all. Finally, the parenthesis next to each project denotes the corresponding cluster number that this project belongs to for an even easier examination. The same notation holds for all tables in the rest of this paper.

After carefully examining Table 5, one may easily notice that the MAPE values tend to be lower in within-cluster forecasting and higher in cross-cluster forecasting. In simple words, a project-specific TD forecasting model tends to provide better forecasts when it is applied to projects that belong to the same cluster, than when applied to projects that belong to different clusters. This

Table 5 Fragment of cross-project MAPE values obtained for all combinations of the 27 projects (within-cluster and cross-cluster) using Lasso Regression (the rest of the table can be found in Table 27 - online Appendix [62])

Train on: **Test on:**	*Zookeeper (0)*	*Httpcomponent-client (0)*	*Commons-beanutils (2)*	*Commons-vfs(3)*
Batik (5)	49%**	1%***	2%***	1%***
Felix (0)	42%**	3%***	5%***	3%***
Zookeeper (0)		0.6%***	13%***	1%***
Httpcomponents-client (0)	28%**		12%***	2%***
Commons-configuration (0)	57%*	3%***	5%***	1%***
Santuario (1)	70%*	6%***	6%***	3%***
Commons-beanutils (2)	33%**	1%***		1%***
Commons-validator (3)	84%	3%***	9%***	3%***
Commons-vfs(3)	39%**	2%***	6%***	
Commons-io (2)	75%*	4%***	7%***	2%***
Mina-sshd (0)	32%**	1%***	14%***	2%***
Aurora (1)	65%*	3%***	7%***	1%***
Beam (1)	8%***	4%***	16%***	5%***
Commons-collections (0)	16%***	2%***	2%***	1%***
Httpcomponents-core (0)	42%**	2%***	11%***	2%***
Commons-bcel (2)	37%**	1%***	4%***	1%***
Commons-cli (4)	157%	18%***	23%**	16%***
Commons-daemon (4)	336%	12%***	45%**	9%***
Commons-dbcp (3)	31%**	7%***	3%***	2%***
Commons-digester (3)	201%	48%**	34%**	8%***
Commons-exec (4)	309%	9%***	34%**	7%***
Commons-fileupload (4)	278%	11%***	39%**	10%***
Commons-jexl (3)	107%	10%***	12%***	7%***
Commons-jxpath (3)	9%***	3%***	5%***	1%***
Commons-net (3)	16%***	2%***	12%***	2%***
Commons-ognl (3)	13%***	2%***	7%***	1%***
Atlas (5)	128%	59%*	64%*	57%*

where (*): MAPE between 51-80%, (**): MAPE between 21-50%, (***): MAPE between 0-20%

supports our initial statement that the similarity of the different projects may play an important role on the predictive performance of the TD forecasting models in cross-project TD forecasting. The same observations can be made for the other regression algorithms, as can be seen in Table 24 - Table 28 (online Appendix [62]). Hence, the similarity of the projects seems to affect the performance of TD forecasting models in cross-project forecasting, regardless of the selected regression algorithm.

As mentioned above, in all the studied cases the errors tend to be lower in within-cluster TD forecasting, than in cross-cluster TD forecasting. In order to reach safer conclusions, hypothesis testing is required. For this purpose, we defined the following null hypothesis (along with its alternative hypothesis) and tested it at the 95% confidence interval:

H_0: *No statistical significant difference is observed between the errors of within-cluster and cross-cluster TD forecasting experiments*
H_1: *A statistical significant difference is observed between the errors of within-cluster and cross-cluster TD forecasting experiments*

The hypothesis was tested for each one of the five regression algorithms that we examined. In order to conduct the statistical tests, the errors reported

in Table Table 24 - Table 28 (online Appendix [62]) were properly grouped into two groups based on whether they correspond to within- or cross-cluster TD forecasting experiment. A fragment of this grouping is presented in Table 6.

Table 6 Fragment of the within-cluster and cross-cluster MAPE values, for each one of the five machine learning algorithms that were tested

SVR RBF		Decision Trees		Linear Regression		Lasso Regression		Ridge Regression	
Within	Cross	Within	Cross	Within	Cross	Within	Cross	Within	Cross
24%	100%	29%	124%	8%	81%	13%	81%	32%	101%
29%	127%	28%	126%	25%	125%	26%	125%	28%	134%
32%	151%	40%	162%	16%	110%	19%	117%	28%	155%
72%	242%	83%	263%	75%	217%	73%	216%	81%	250%
40%	59%	57%	76%	40%	46%	40%	45%	39%	77%
40%	308%	42%	363%	40%	270%	20%	346%	38%	317%
37%	220%	72%	220%	15%	215%	18%	215%	49%	216%
67%	870%	68%	983%	19%	60%	6%	805%	67%	889%
68%	809%	70%	821%	6%	883%	41%	899%	74%	741%
145%	569%	160%	580%	40%	552%	13%	558%	162%	566%
120%	478%	129%	602%	12%	305%	12%	308%	129%	497%
59%	193%	65%	222%	12%	153%	7%	152%	70%	177%
156%	46%	164%	69%	8%	12%	13%	12%	162%	31%
14%	13%	17%	34%	13%	26%	7%	21%	18%	14%
24%	100%	29%	124%	8%	81%	13%	81%	32%	101%

In order to test the null hypothesis for each one of the five cases, the Wilcoxon's Rank Sum test was employed, which is a non-parametric test that does not assume any distribution for the analyzed data. More specifically, the Wilcoxon Rank Sum Test was employed five times, one for each studied regression algorithm, with the purpose to compare the errors of the within- and cross-cluster experiments. The results of the tests are presented in Table 7.

Table 7 The results of the Wilcoxon Rank Sum Test for each one of the five machine learning algorithms that were tested

Algorithm Name	p-value
SVR RBF	<2.2e-16
Decision Trees	<2.2e-16
Linear Regression	<2.2e-16
Lasso Regression	<2.2e-16
Ridge Regression	<2.2e-16

As can be seen in Table 7, in all the studied cases the p-value was found to be lower than the threshold of 0.05. As a result, in all the studied cases the null hypothesis is rejected, which leads to the acceptance of the alternative hypothesis. This suggests that a statistical significant difference is observed between the errors of within-cluster and cross-cluster TD forecasting. Thus, this provides further support to our observation that the similarity of software projects may play an important role in the predictive performance of TD forecasting models in cross-project forecasting. In other words, when a TD forecasting model is applied on a software project that is highly similar to the

project (or projects) that was used for its training, it is highly likely to provide satisfactory TD forecasts.

4.2 Selection of the Kernel Model per Cluster

In the previous section, we have shown that training and testing cross-project TD forecasting models within a specific cluster offers lower errors compared to cross-cluster predictions. The next important step of the overall approach is the selection of the best performing TD forecasting model for each one of the constructed clusters, in terms of both the forecasting algorithm and the dataset that it is trained on. The best model of each cluster will actually be used as the representative (or kernel) model of this cluster, i.e., it will be applied for providing TD forecasts on any new project that is assigned to this cluster afterwards.

For the purpose of identifying the best performing TD forecasting model for each cluster, we will exploit the results obtained through the exhaustive experiments performed within Section 4.1 for all combinations of the 27 projects. However, we will focus exclusively on within-cluster forecasting, since we are willing to find the best model for each cluster. Similarly to the cross-project MAPE values reported in Section 4.1, in this section, each time a specific project is used for training, we calculate the mean value of all the MAPE values generated when testing is performed on the rest of the projects of the same cluster. This way, we have a single value representing the performance of every project when used for the training process. Afterwards, we extend this approach by calculating also the mean value of these mean values, in order to have one single value representing the cross-project performance of each algorithm on each cluster.

As an example, Table 8 through Table 12 report the within-cluster MAPE values obtained by using the five selected forecasting algorithms for five steps (commits) ahead for Cluster 0. For reasons of brevity, we do not provide similar tables for the rest of the clusters. However, this information can be easily retrieved from the detailed tables found in the online Appendix [62] (Table 24 to Table 58), by focusing only on the cross-project results referring to a specific cluster. By inspecting Table 8 through Table 12, we can see that the last row consists of the mean of the MAPE values when a specific project is used for training and the remaining projects of the cluster are used for testing. The bottom right cell represents the mean value of all the aforementioned mean values, which as mentioned earlier represents the within-cluster cross-project performance of every project when used for the training process. It should be noted that the diagonal is empty, since at this point we do not care about within-project prediction. What we are looking for is the model that performs best in cross-project TD forecasting, within a given cluster.

As mentioned earlier, in this section we aim to identify which is the best forecasting model for each cluster, so it can actually be used as the representative (or kernel) model of this cluster in order to provide TD forecasts on

Table 8 Within-cluster cross-project MAPE values obtained for Cluster 0 using SVR for 5 steps ahead

Train on: Test on:	*Felix*	*Zookeeper*	*Httpcomponents-client*	*Commons-configuration*	*Mina-sshd*	*Commons-collections*	*Httpcomponents-core*	
Felix		74%*	58%*	44%**	47%**	50%**	44%**	
Zookeeper	16%***		19%***	57%*	44%**	32%**	55%*	
Httpcomponents-client	149%	183%		89%	104%	122%	91%	
Commons-configuration	161%	213%	151%		72%*	111%	52%*	
Mina-sshd	117%	155%	111%	55%*		88%	57%*	
Commons-collections	78%*	113%	72%*	56%*	54%*		55%*	
Httpcomponents-core	205%	257%	196%	88%	118%	157%		
Mean Value:	**121.5%**	**166.3%**	**101.5%**	**65.2%**	**73.7%**	**93.9%**	**59.6%**	**97.4%**

Table 9 Within-cluster cross-project MAPE values obtained for Cluster 0 using Random Forest for 5 steps ahead

Train on: Test on:	*Felix*	*Zookeeper*	*Httpcomponents-client*	*Commons-configuration*	*Mina-sshd*	*Commons-collections*	*Httpcomponents-core*	
Felix		70%*	14%***	37%**	27%**	18%***	17%***	
Zookeeper	11%***		3%***	46%**	23%**	19%***	7%***	
Httpcomponents-client	50%**	163%		40%**	34%**	27%**	23%**	
Commons-configuration	81%	197%	55%*		36%**	32%**	103%	
Mina-sshd	53%*	141%	19%***	22%**		20%**	29%**	
Commons-collections	33%**	102%	14%***	24%**	19%***		38%**	
Httpcomponents-core	78%*	232%	29%**	25%**	27%**	24%**		
Mean Value:	**51.6%**	**151.4%**	**22.8%**	**32.9%**	**28.1%**	**23.9%**	**36.8%**	**49.6%**

any new project that is assigned to this cluster afterwards. In what follows, we explore the performance of each model per cluster. To do that, we aggregate the values representing the cross-project performance of each algorithm trained on each project of each cluster (last row in Table 8 through Table 12) on graphs so that we can visualize the results that finally will lead to a general conclusion. There are two graphs for each cluster for five and 10 steps (commits) ahead respectively, thus, there are 12 graphs in total.

In Figure 5 and Figure 6, the results for Cluster 0 for 5 and 10 steps ahead respectively are illustrated for all five algorithms. X-axis represents the project on which the within-cluster cross-project model is trained, while y-axis represents the mean value of MAPE values for that model.

By inspecting Figure 5 and Figure 6, we can see that the algorithm that gave the highest MAPE value is SVR for all different training datasets. Random Forest provided us with the lowest errors when the model was trained on *Felix* both for 5 and 10 steps ahead. For 5 steps ahead, Linear Regression gave the lowest MAPE value when the model was trained on *Zookeeper*, whereas both Linear and Ridge Regression offered the lowest MAPE values when the model was trained on *Httpcomponents-client*, *Commons-configuration*, *Mina-*

Table 10 Within-cluster cross-project MAPE values obtained for Cluster 0 using Linear Regression for 5 steps ahead

Train on: Test on:	*Felix*	*Zookeeper*	*Httpcomponents-client*	*Commons-configuration*	*Mina-sshd*	*Commons-collections*	*Httpcomponents-core*	
Felix		45%**	3%***	5%***	5%***	3%***	22%**	
Zookeeper	69%*		0.6%***	1%***	0.4%***	5%***	5%***	
Httpcomponents-client	78%*	28%**		1%***	2%***	6%***	2%***	
Commons-configuration	136%	52%*	2%***		7%***	6%***	3%***	
Mina-sshd	88%	31%**	1%***	2%***		3%***	5%***	
Commons-collections	67%*	16%***	2%***	3%***	7%***		8%***	
Httpcomponents-core	79%*	42%**	2%***	2%***	3%***	7%***		
Mean Value:	**86.6%**	**36.1%**	**2.3%**	**3.0%**	**4.3%**	**5.5%**	**7.9%**	**20.8%**

Table 11 Within-cluster cross-project MAPE values obtained for Cluster 0 using Lasso Regression for 5 steps ahead

Train on: Test on:	*Felix*	*Zookeeper*	*Httpcomponents-client*	*Commons-configuration*	*Mina-sshd*	*Commons-collections*	*Httpcomponents-core*	
Felix		42%**	3%***	5%***	13%***	3%***	29%**	
Zookeeper	46%**		0.6%***	1%***	0.6%***	5%***	7%***	
Httpcomponents-client	59%*	28%**		1%***	4%***	6%***	2%***	
Commons-configuration	91%	57%*	3%***		20%***	6%***	7%***	
Mina-sshd	62%*	32%**	1%***	2%***		3%***	7%***	
Commons-collections	46%**	16%***	2%***	3%***	20%***		9%***	
Httpcomponents-core	68%*	42%**	2%***	2%***	7%***	7%***		
Mean Value:	**62.5%**	**36.6%**	**2.5%**	**2.9%**	**11.2%**	**5.5%**	**10.5%**	**18.8%**

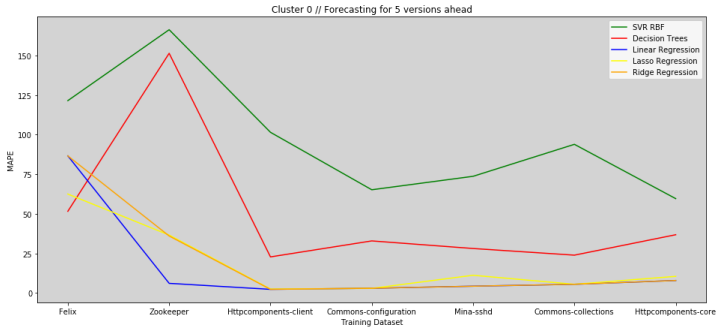

Fig. 5 Within-cluster mean values of MAPE values obtained for Cluster 0 for 5 steps ahead

sshd, *Commons-collections* and *Httpcomponents-core*. For 10 steps ahead, the performance of Linear and Ridge Regression is exactly the same providing us with the lowest errors except for the case when training was performed on *Felix*. It is also obvious from the graphs that the datasets that are the most suitable to use for training are *Httpcomponents-client* and *Commons-configuration*.

Table 12 Within-cluster cross-project MAPE values obtained for Cluster 0 using Ridge Regression for 5 steps ahead

Train on: Test on:	*Felix*	*Zookeeper*	*Httpcomponents-client*	*Commons-configuration*	*Mina-sshd*	*Commons-collections*	*Httpcomponents-core*	
Felix		45%**	3%***	5%***	5%***	3%***	22%**	
Zookeeper	69%*		0.6%***	1%***	0.4%***	5%***	5%***	
Httpcomponents-client	78%*	28%**		1%***	2%***	6%***	2%***	
Commons-configuration	136%	52%*	2%***		7%***	6%***	3%***	
Mina-sshd	88%	31%**	1%***	2%***		3%***	5%***	
Commons-collections	67%*	16%***	2%***	3%***	7%***		8%***	
Httpcomponents-core	79%*	42%**	2%***	2%***	3%***	7%***		
Mean Value:	**86.6%**	**36.0%**	**2.3%**	**3.0%**	**4.3%**	**5.5%**	**7.9%**	**20.8%**

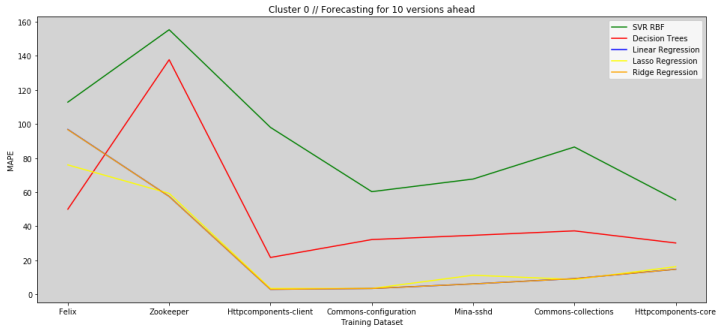

Fig. 6 Within-cluster mean values of MAPE values obtained for Cluster 0 for 10 steps ahead

On Figure 7 and Figure 8 the reader can see the results for Cluster 1 for 5 and 10 steps ahead respectively for all 5 algorithms.

There are a few differences spotted between the two graphs. When the model is trained on *Beam*, SVR produces the highest errors followed by Linear, Lasso and Ridge Regression for 5 steps ahead whereas, for 10 steps ahead, Linear and Ridge Regression offer slightly higher errors than Lasso Regression followed by SVR. For both 5 and 10 steps ahead Random Forest gave us the lowest errors. In addition, when the model is trained on *Aurora* and *Santuario*, for 5 and 10 steps ahead, all algorithms have almost identical performance. For these cases Lasso, Ridge and Linear Regression produce the lowest errors and SVR the highest.

Overall, SVR produced the highest MAPE values except for the case of training on *Beam* for 10 steps ahead. On the other hand, Ridge and Lasso Regression generated the lowest MAPE values except when we train the model on *Beam* where the lowest errors are generated when Random Forest is used. As the reader can see, *Aurora* is the dataset that is the best out of the three of Cluster 1 to use for training.

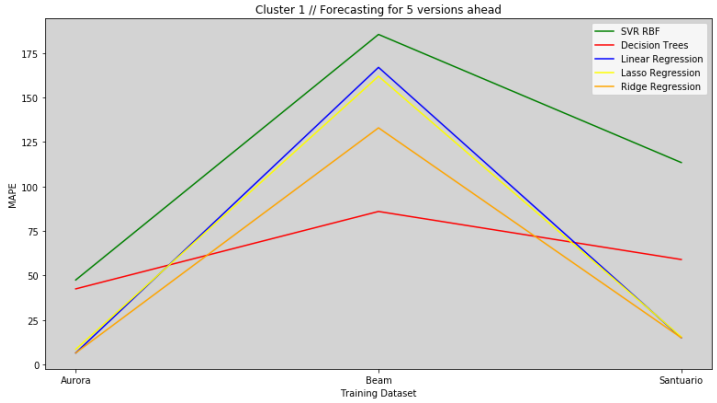

Fig. 7 Within-cluster mean values of MAPE values obtained for Cluster 1 for 5 steps ahead

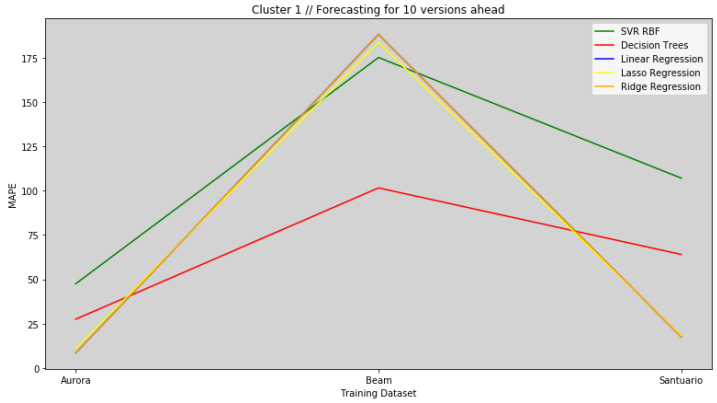

Fig. 8 Within-cluster mean values of MAPE values obtained for Cluster 1 for 10 steps ahead

On Figure 9 and Figure 10 the reader can see the results for Cluster 2 for 5 and 10 steps ahead respectively for all algorithms.

For Cluster 2, for 5 and 10 steps ahead, the two graphs are almost identical. When the model is trained on *Commons-beanutils*, *Commons-io* and *Commons-bcel* the lowest MAPE values were produced when using Linear, Ridge and Lasso Regression, while the highest MAPE values were produced when using SVR.

Overall, Linear and Ridge Regression provide us with the best results and the dataset that performs best as training dataset is *Commons-io*.

On Figure 11 and Figure 12 the reader can see the results for Cluster 3 for 5 and 10 steps ahead respectively for all algorithms.

Comparing the two graphs we can verify that 5 and 10 steps ahead results are very much alike with a few differences. An exception is the case of training on *Commons-digester* where, for 10 steps ahead, errors obtained when using

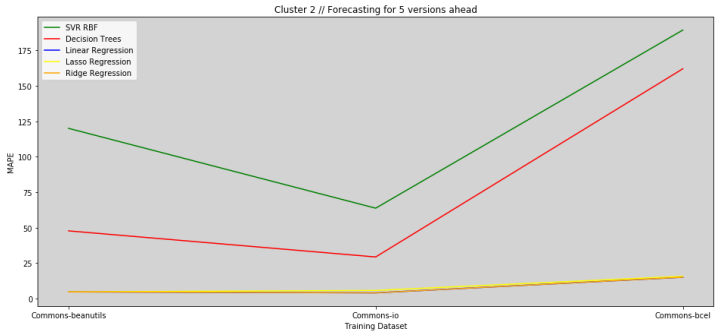

Fig. 9 Within-cluster mean values of MAPE values obtained for Cluster 2 for 5 steps ahead

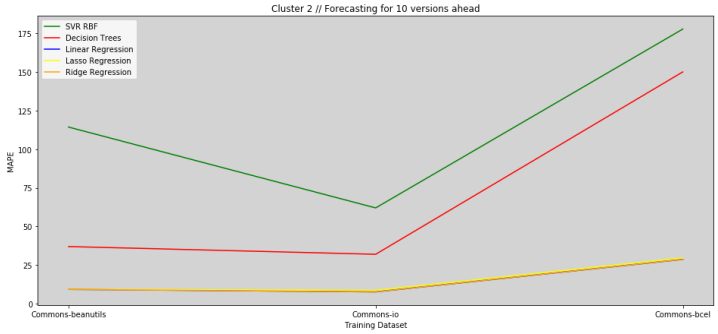

Fig. 10 Within-cluster mean values of MAPE values obtained for Cluster 2 for 10 steps ahead

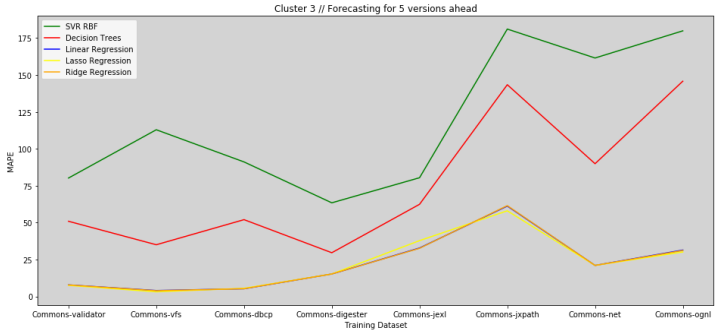

Fig. 11 Within-cluster mean values of MAPE values obtained for Cluster 3 for 5 steps ahead

Random Forest are slightly lower than the errors obtained using the linear algorithms, whereas for 5 steps ahead the exact opposite is observed. Also, another exception is in the case of *Commons-jexl* where the performance of Random Forest, Linear, Lasso and Ridge Regression for 10 steps ahead is very

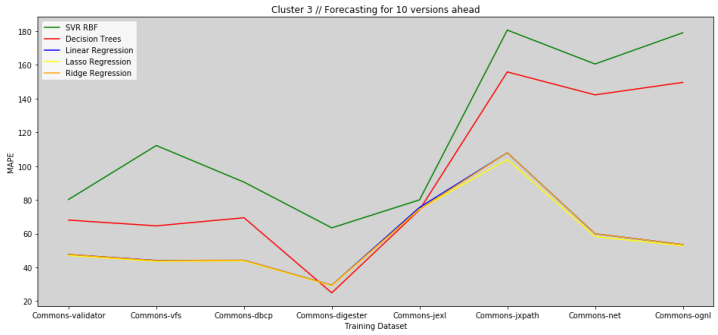

Fig. 12 Within-cluster mean values of MAPE values obtained for Cluster 3 for 10 steps ahead

close whereas for 5 steps ahead Linear, Lasso and Ridge Regression produced way lower errors than Random Forest.

Overall, SVR provided us with the highest errors. As we can easily detect from above, there is not a clear conclusion as to which algorithm or which training dataset might offer us the lowest errors both for 5 and 10 steps ahead. However, for 5 steps ahead, it is clear that Linear, Lasso and Ridge Regression display the best performance and the most optimal datasets to use as training datasets are *Commons-vfs*, *Commons-dbcp* and *Commons-validator*.

On Figure 13 and Figure 14 the reader can see the results for Cluster 4 for 5 and 10 steps ahead respectively for all algorithms.

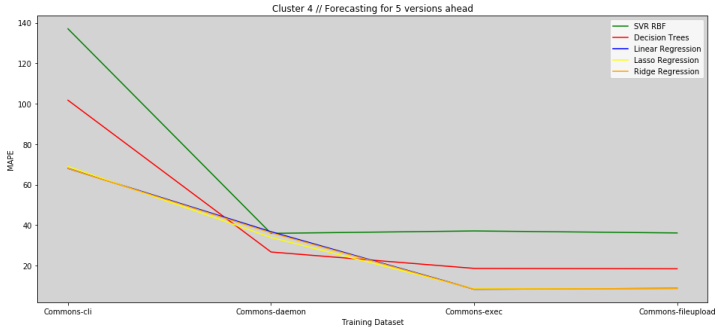

Fig. 13 Within-cluster mean values of MAPE values obtained for Cluster 4 for 5 steps ahead

Both graphs look alike except for the cases where we use *Commons-cli* and *Commons-daemon* as training datasets. In the first case, Linear, Lasso and Ridge Regression produce better results than Random Forest for 5 steps ahead whereas for 10 steps ahead MAPE values produced by Linear, Lasso and Ridge Regression exceed by far those produced by Random Forest. In the case where training was performed on *Commons-daemon*, for 5 steps ahead,

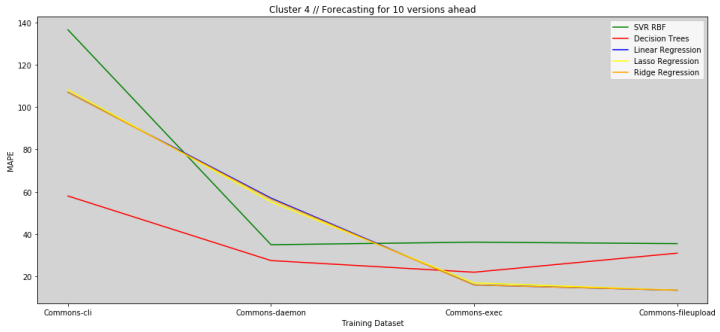

Fig. 14 Within-cluster mean values of MAPE values obtained for Cluster 4 for 10 steps ahead

Linear, Lasso, Ridge Regression and SVR present the same performance which is worse than Random Forest performance, whereas for 10 steps ahead, again the lowest errors are provided by Random Forest, but this time, SVR follows with Linear, Lasso and Ridge Regression producing the highest errors.

For 5 and 10 steps ahead, there is not a general conclusion that can apply to both cases. For 5 steps ahead, for *Commons-cli*, *Commons-exec* and *Commons-fileupload*, the linear algorithms provide us with the best results whereas for *Commons-daemon*, Random Forest provides us with the best results. The datasets that generate the lowest errors when used for training are *Commons-exec* and *Commons-fileupload*.

On Figure 15 and Figure 16 the reader can see the results for Cluster 5 for 5 and 10 steps ahead respectively for all algorithms.

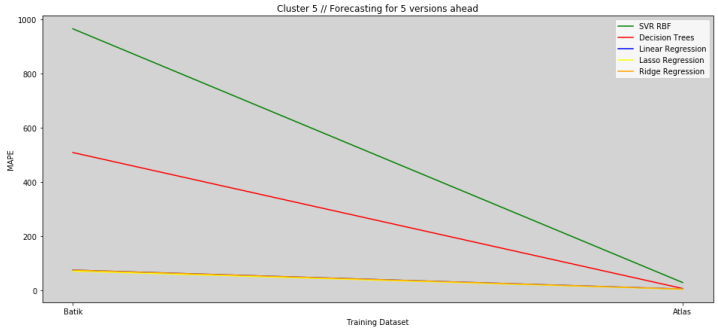

Fig. 15 Within-cluster mean values of MAPE values obtained for Cluster 5 for 5 steps ahead

Cluster 5 only consists of two datasets. Both graphs representing 5 and 10 steps ahead are once more indistinguishable. When training was performed on *Batik*, SVR produce the highest MAPE values, followed by Random Forest followed by the linear algorithms producing by far the lowest errors. When we

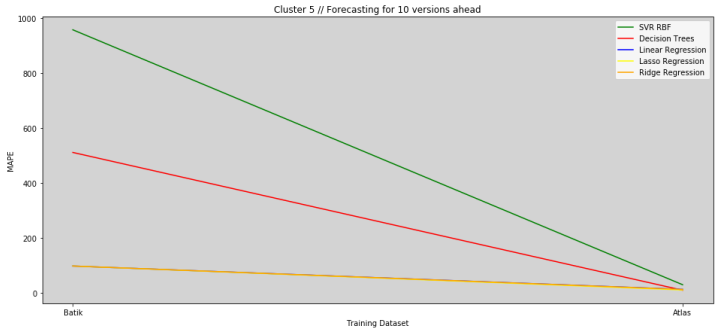

Fig. 16 Within-cluster mean values of MAPE values obtained for Cluster 5 for 10 steps ahead

train the model on *Atlas* the lowest MAPE value is given by Lasso Regression and the highest MAPE value is given by SVR. To sum up, *Atlas* is the most suitable dataset for training using Lasso Regression.

After providing detailed results on the performance obtained by each algorithm when executing experiments by training and testing on every dataset of each cluster (i.e., within-cluster cross-project predictions), in the part that follows, we aim to aggregate these results in order to conclude on which algorithm is the optimal choice for each cluster. To do so, we obtain the mean of the mean values of the errors produced when training on each dataset of each cluster. In order to make it easier for the reader to understand, we take a look again on the bottom right cell of Table 8 through Table 12. The reader may assume that since we have 6 clusters and 5 different algorithms, we obtain 30 such values which are presented on Table 13. Through this process we can observe which of the algorithms offers the lowest mean value for each cluster, and therefore to determine the best-performing model (or kernel model) for each cluster.

Table 13 Performance of each algorithm for each cluster. The lowest values for each cluster have been marked with an asterisk

Cluster: Algorithm:	*Cluster 0*	*Cluster 1*	*Cluster 2*	*Cluster 3*	*Cluster 4*	*Cluster 5*
SVR	97.4%	115.5%	140.3%	112.5%	69.0%	497.5%
Random Forest	49.6%	62.5%	85.3%	74.8%	46.9%	258.5%
Linear Regression	20.8%	62.8%	**9.3%***	**21.5%***	31.0%	41.0%
Lasso Regression	**18.8%***	62.0%	10.2%	21.7%	**30.4%**	**38.6%***
Ridge Regression	20.8%	**51.5%***	**9.3%***	**21.5%***	30.8%	41.0%

As a reminder, in this section we try to identify the best performing TD forecasting model for each one of the constructed clusters, in terms of both the

forecasting algorithm and the dataset that it is trained on. At this point we summarize the results which construct the answer to the question we initially set in this section. In order to make a statement on which of the projects of each cluster are the optimal to use for training, we carefully examine the tables of the results of the corresponding algorithm that we just stated as optimal for each cluster. This means that we inspect Figure 5 to Figure 16, seeking for the curve that represents the performance of the optimal algorithm for each cluster. Then, we choose the projects which provide the lowest errors when used for training with the optimal algorithm. On Table 14 the reader may find the best training projects and the optimal algorithm for each cluster. Out of the total number of projects lying within each cluster, we chose to present the top 30% of them that exhibit the best performance. Hence, for Cluster 0, 1, 2, 3, 4 and 5 we choose two, one, one, three, two, and one projects respectively.

Table 14 Optimal training projects for each cluster and the corresponding selected algorithm that provided the lowest errors

Cluster 0	***Cluster 1***	***Cluster 2***	***Cluster 3***	***Cluster 4***	***Cluster 5***
Optimal Datasets for Training					
Httpcomponents-client *Commons-configuration*	*Aurora*	*Commons-io*	*Commons-vfs* *Commons-dbcp* *Commons-validator*	*Commons-exec* *Commons-fileupload*	*Atlas*
Selected Algorithm					
Lasso Regression	Ridge Regression	Linear Regression	Linear Regression	Lasso Regression	Lasso Regression

By inspecting Table 14, we are now able to clearly state that for Cluster 0 the optimal algorithm is LassoRegression, for Cluster 1 Ridge Regression, for Cluster 2 and 3 Linear Regression and finally, for Cluster 4 and 5 Lasso Regression. Lastly, we are also able to state that the optimal projects to be used for training for Cluster 0 are *Httpcomponents-client* and *Commons-configuration*, for Cluster 1 *Aurora*, for Cluster 2 *Commons-io*, for Cluster 3 *Commons-vfs*, *Commons-dbcp* and *Commons-validator*, for Cluster 4 *Commons-exec* and *Commons-fileupload* and finally, for Cluster 5 *Atlas*.

4.3 Merging Datasets

In the previous section, we tried to identify the best performing TD forecasting model for each one of the constructed clusters, in terms of both the forecasting algorithm and the dataset that it is trained on. While the choice of the optimal forecasting algorithm per cluster was clear, we noticed that there were cases where for some specific clusters more than one projects provided good cross-project results. In a real scenario where a new project arrives and we need to perform forecasts for, it would be difficult to find the optimal project and "borrow" its pre-trained model. This would require a lot of ex-

periments based on trial and error. However, a merged dataset possibly allows for more generalizability. Thus, as our next step, we aim to examine whether merging project datasets within a cluster would improve the cross-project TD forecasting model performance. For this purpose, we chose to merge the best-performing projects (as reported in Table 14 of Section 4.2) which correspond to the proportion of 30% of the total number of datasets within each cluster, train cross-project models on the merged datasets and then test the performance of these models on the remaining datasets of the same cluster. For those clusters that 1 dataset corresponds to the proportion of 30%, no merging took place. We remind the reader that for Cluster 0, 1, 2, 3, 4 and 5 we chose two, one, one, three, two, and one projects respectively.

We ran the experiments for all the clusters for 5 steps ahead using the algorithms that were found to be optimal for each cluster as reported in Table 14. For the cases where a merged dataset is formed, we also present the results that were obtained before merging, so that we can gain a deeper understanding on whether the merged datasets demonstrate better cross-project forecasting performance or not. In Table 15 through Table 20, the errors obtained for each cluster are presented.

Table 15 Within-cluster cross-project MAPE values obtained for Cluster 0 using a merged dataset. Results corresponding to training on the datasets that the merged dataset consists of are also presented for comparison reasons

Train on: **Test on:**	*Merged (0)*	*Httpcomponents-client (0)*	*Commons-configuraton (0)*
Felix (0)	4%***	3%***	5%***
Zookeeper (0)	1%***	0.7%***	1%***
Mina-sshd (0)	1%***	1%***	2%***
Commons-collections (0)	3%***	2%***	3%***
Httpcomponents-core (0)	4%***	2%***	2%***
Mean Value:	**2.6%**	**1.6%**	**2.6%**

Table 16 For Cluster 1, 30% of the cluster corresponds to 1 dataset thus we present again the results when Aurora was used as a training dataset

Train on: **Test on:**	*Aurora (1)*
Beam	7%***
Santuario	6%***
Mean Value:	**6.5%**

By inspecting Table 15 through Table 20, we can see that for all 6 clusters the interpretation of the results is somewhat similar. More specifically, when training is performed on the merged datasets, the obtained cross-project performance is equal or in some cases higher than the performance obtained when

Table 17 For Cluster 2, 30% of the cluster corresponds to 1 dataset thus we present again the results when Commons-io was used as a training dataset

Train on: **Test on:**	*Commons-io (2)*
Commons-beanutils (2)	8%***
Commons-bcel (2)	3%***
Mean Value:	**5.5%**

Table 18 Within-cluster cross-project MAPE values obtained for Cluster 3 using a merged dataset. Results corresponding to training on the datasets that the merged dataset consists of are also presented for comparison reasons

Train on: **Test on:**	*Merged (3)*	*Commons-vfs (3)*	*Commons-dbcp (3)*	*Commons-validator (3)*
Commons-digester (3)	16%***	15%***	11%***	18%***
Commons-jexl (3)	7%***	7%***	9%***	14%***
Commons-jxpath (3)	1%***	1%***	2%***	8%***
Commons-net (3)	2%***	2%***	3%***	2%***
Commons-ognl (3)	1%***	1%***	4%***	4%***
Mean Value:	**5.4%**	**5.2%**	**5.8%**	**9.2%**

Table 19 Within-cluster cross-project MAPE values obtained for Cluster 4 using a merged dataset. Results corresponding to training on the datasets that the merged dataset consists of are also presented for comparison reasons

Train on: **Test on:**	*Merged (4)*	*Commons-exec (4)*	*Commons-fileupload (4)*
Commons-cli (4)	20%***	15%***	19%***
Commons-daemon (4)	2%***	8%***	6%***
Mean Value:	**11.0%**	**11.5%**	**12.5%**

Table 20 For Cluster 5, 30% of the cluster corresponds to 1 dataset thus we present again the results when Atlas was used as a training dataset

Train on: **Test on:**	*Atlas (5)*
Batik (5)	5%***
Mean Value:	**5.0%**

training is performed on the individual projects of which the merged datasets consist.

4.4 Case Study

The last step of our approach is the actual execution of the pre-trained models on previously unknown real-world software projects (i.e., software projects that

have not been used for the previous analysis). According to our approach, when a new project arrives, it is initially assigned to one of the six clusters, based on its similarity, and, subsequently, the representative (i.e., kernel) TD forecasting model of this cluster is applied to the project. In order for our approach to be valid, the TD forecasting model of the cluster to which the new project is assigned should provide better forecasts (i.e., lower errors) compared to the TD forecasting models of the other clusters.

To this end, in the present section, we use three real-world software projects that were not used neither for the construction of the clusters, nor for the construction of the representative TD forecasting models of the clusters. The new testing projects are *Commons-jelly*, *Commons-codec* and *Commons-dbutils* and were also retrieved from the TD dataset provided by Lenarduzzi et al. [39], as presented in Section 3.1. After employing the K-means algorithm presented in Section 3.3, these projects have been assigned to Cluster 2, 3 and 4 respectively. Therefore, kernel models of the three dedicated clusters (i.e., Cluster 2, 3 and 4) will be used to make cross-project predictions. Our expectation is that these pre-trained kernel models will offer lower errors compared to the pre-trained models of other clusters.

Having assigned the new datasets to our clusters, we test the pre-trained models by referring to Table 14 that consists of our results. The testing process is similar to the testing process we followed in the previous section. This time however, we will not test all possible combinations of the datasets for 5 different algorithms, but instead we will test the kernel pre-trained models on the entire new testing dataset of each project assigned to the corresponding cluster. For Cluster 0, the pre-trained model is trained on the merged dataset that consists of *Httpcomponents-client* and *Commons-configuration* using Lasso Regression. For Cluster 1, the pre-trained model is trained on *Aurora* using Ridge Regression. For Cluster 2, the pre-trained model is trained on *Commons-io* using Linear Regression. For Cluster 3, the pre-trained model is trained on the merged dataset that consists of *Commons-vfs*, *Commons-dbcp* and *Commons-validator* using Linear Regression. For Cluster 4, the pre-trained model is trained on the merged dataset that consists of *Commons-exec* and *Commons-fileupload* using Lasso Regression. Finally, for Cluster 5 the pre-trained model is trained on *Atlas* using Lasso Regression. The results that were obtained when testing was performed on the new projects are presented on Table 21. Each row represents the performance of the pre-trained models on the new testing datasets. The errors that have been obtained when training and testing took place within the same cluster have been marked with an asterisk.

The results have met the expectations. Indeed, as can be seen by inspecting Table 21, the pre-trained kernel models of each of Clusters 2, 3 and 4 produced errors lower or equal for each of the new testing projects compared to the pre-trained models of other clusters.

Table 21 MAPEs obtained for the use cases of 4 new projects using all of the 6 pre-trained models

Train on: Test on:	*Merged (0)*	*Aurora (1)*	*Commons-io (2)*	*Merged (3)*	*Merged (4)*	*Atlas (5)*
Commons-jelly (2)	2%	2%	**1%***	1%	7%	2%
Commons-codec (3)	4%	5%	2%	**1%***	7%	14%
Commons-dbutil (4)	2%	7%	5%	4%	**2%***	9%

5 Conclusions and Future Work

TD refers to deliberate or inadvertent non-optimal design decisions made during the software development lifecycle that lead to poorly designed systems and therefore additional maintenance effort. Due to its interdisciplinary nature, TD has attracted the attention of both academia and industry over the last years, resulting in a considerable increase in the number of methods and accompanying tools that support TD management activities, such as TD identification, quantification or repayment. Besides standard TD management activities, predicting the accumulated TD during the evolution of a software application is considered crucial, since such an action would allow project managers and developers perform long-term effective software maintenance. However, current TD forecasting approaches build on the assumption that reliable historic data are available in order for the models to be applied to a specific software project and provide accurate forecasts. Under those circumstances, a method that enables the provision of TD forecasts for software projects that do not exhibit a long commit history could provide practical decision-making mechanisms from the early stages of software development. Cross-project TD forecasting, that is, building a forecasting model based on data retrieved from one project and using it to get reliable forecasts for a new, previously unknown software project, would allow project managers and developers leverage the benefits of TD forecasting in cases where a long history of commits is not available, e.g., from the very early stages of the development.

The purpose of this paper is to examine whether the adoption of clustering is a promising solution for enhancing the accuracy of cross-project TD forecasting. In other words, we investigate whether the consideration of TD-related similarities between software projects could be the key for more accurate cross-project forecasting. For this purpose, a large dataset was utilized, comprising 27 real-world open-source Java applications. These applications were then fed into a clustering algorithm and divided into clusters of similar projects with respect to their TD aspects. Subsequently, cluster-representative TD forecasting models were constructed through several experiments, using five regression algorithms for forecasting horizons between 1 and 10 steps ahead. We observed that the cross-project prediction accuracy of the cluster-representative models was higher when forecasting for projects assigned to the same cluster (i.e.,

within-cluster forecasting), compared to forecasting for projects assigned to a different cluster (i.e., cross-cluster forecasting), while in most of the cases, the future TD value was captured with a sufficient level of accuracy. Furthermore, we investigated whether merging projects would further improve the cross-project forecasting performance of the cluster-representative models. Finally, to evaluate the usefulness of the proposed approach in practice, various previously unknown real-world applications were assigned to the defined clusters and the TD forecasting models associated to these clusters were applied to predict their future TD evolution. The forecasting error was observed to be smaller when the kernel model of the cluster to which the previously unknown project was assigned was used, compared to the error of the other kernel models. This suggests that the similarity of the software projects may be a key factor for enhancing cross-project TD forecasting, and, in turn, that clustering may be a viable solution for achieving better results in cross-project forecasting. In brief, the results of the analysis were encouraging and suggest that the proposed approach is a promising solution for more accurate cross-project TD forecasting.

Several directions for future work can be identified. First of all, the present study was based solely on open-source Java applications retrieved from the Apache Software Foundation. In order to investigate the generalizability of the produced results, we plan to replicate our study on a broader spectrum of real-world software applications that are written in other programming languages and that belong to different domains. Secondly, in this study, the consideration of TD-related similarities between software projects, as well as the selection of features to be used as predictors for the construction of forecasting models was based only on TD indicators retrieved from SonarQube. We consider to extend this study by considering also TD indicators retrieved from other sources, such as different ASA tools and OO metric suits. We also plan to investigate the extension of cross-project TD forecasting to lower levels of granularity of a software project, such as package, class or function level. Finally, future work includes improving the proposed approach by considering more sophisticated clustering methods, as well as the possibility that the clustering process dynamically repeats the training process once a new project arrives, in order to update the clusters and retrain the dedicated forecasting models taking into account the newly acquired data.

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

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Alves, N.S.R., Mendes, T.S., Mendonça, M.G.d., Spínola, R.O., Shull, F., Seaman, C.: Identification and management of technical debt: A systematic mapping study. Information and Software Technology **70**, 100 – 121 (2016). DOI http://dx.doi.org/10.1016/j.infsof.2015.10.008. URL http://www.sciencedirect.com/science/article/pii/S0950584915001743
2. Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., Chatzigeorgiou, A.: Establishing a framework for managing interest in technical debt. In: 5th International Symposium on Business Modeling and Software Design (BMSD). Citeseer (2015). DOI 10.5220/0005885700750085
3. Ampatzoglou, A., Michailidis, A., Sarikyriakidis, C., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: A Framework for Managing Interest in Technical Debt: An Industrial Validation. In: Proceedings of the 2018 International Conference on Technical Debt (TechDebt) (2018). DOI http://dx.doi.org/10.1145/3194164.3194175
4. Arisholm, E., Briand, L.C.: Predicting fault-prone components in a java legacy system. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering and measurement (ESEM), pp. 8–17. ACM (2006). DOI http://dx.doi.org/10.1145/1159733.1159738
5. Arvanitou, E.M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Stamelos, I.: Monitoring technical debt in an industrial setting. In: Proceedings of the Evaluation and Assessment on Software Engineering, pp. 123–132. Association for Computing Machinery (2019)
6. Bellman, R.: Dynamic Programming. Dover Books on Computer Science Series. Dover Publications (2003). URL https://books.google.gr/books?id=fyVtp3EMxasC
7. Boehm, B.W., others: Software engineering economics. IEEE Transactions on Software Engineering **SE-10**(1), 4 – 21 (1984). DOI 10.1109/TSE.1984.5010193
8. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., others: Managing technical debt in software-reliant systems. In: Proceedings of the workshop on Future of software engineering research (FSE/SDP), pp. 47–52. ACM (2010). DOI http://dx.doi.org/10.1145/1882362.1882373
9. Canfora, G., Lucia, A.D., Penta, M.D., Oliveto, R., Panichella, A., Panichella, S.: Multi-objective Cross-Project Defect Prediction. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 252–261 (2013)
10. Chaikalis, T., Chatzigeorgiou, A.: Forecasting java software evolution trends employing network models. IEEE Transactions on Software Engineering **41**(6), 582–602 (2015). DOI http://dx.doi.org/10.1109/TSE.2014.2381249
11. Challagulla, V.U.B., Bastani, F.B., Paul, a.R.A.: Empirical assessment of machine learning based software defect prediction techniques. In: 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), pp. 263–270 (2005). DOI http://dx.doi.org/10.1109/WORDS.2005.32
12. Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A., Amanatidis, T.: Estimating the breaking point for technical debt. In: IEEE 7th international workshop on Managing technical debt (MTD), pp. 53–56. IEEE (2015). DOI http://dx.doi.org/10.1109/MTD.2015.7332625
13. Chug, A., Malhotra, R.: Benchmarking framework for maintainability prediction of open source software using object oriented metrics. International Journal of Innovative Computing, Information and Control **12**(2), 615–634 (2016)
14. Cunningham, W.: The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger **4**(2), 29–30 (1993). DOI http://dx.doi.org/10.1145/157710.157715
15. Dietterich, T.G.: Machine learning for sequential data: A review. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), pp. 15–30. Springer (2002). DOI http://dx.doi.org/10.1007/3-540-70659-3_2
16. Digkas, G., Lungu, M., Avgeriou, P., Chatzigeorgiou, A., Ampatzoglou, A.: How do developers fix issues and pay back technical debt in the apache ecosystem. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 153–163. IEEE (2018). DOI 10.1109/SANER.2018.8330205

17. Digkas, G., Lungu, M., Chatzigeorgiou, A., Avgeriou, P.: The evolution of technical debt in the apache ecosystem. In: European Conference on Software Architecture (ECSA), pp. 51–66. Springer (2017). DOI http://dx.doi.org/10.1007/978-3-319-65831-5_4
18. Elish, M.O., Elish, K.O.: Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study. In: 2009 13th European Conference on Software Maintenance and Reengineering (CSMR), pp. 69–78 (2009). DOI 10.1109/CSMR.2009.57. ISSN: 1534-5351
19. Fontana, F.A., Ferme, V., Spinelli, S.: Investigating the impact of code smells debt on quality code evaluation. In: Proceedings of the Third International Workshop on Managing Technical Debt (MTD), pp. 15–22. IEEE Press (2012). DOI http://dx.doi.org/10.1109/MTD.2012.6225993
20. Fontana, F.A., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering **21**(3), 1143–1191 (2016). DOI https://doi.org/10.1007/s10664-015-9378-4
21. Fowler, M.: Refactoring: improving the design of existing code. Addison-Wesley Professional (1999)
22. Gall, H.C., Lanza, M.: Software evolution: analysis and visualization. In: Proceedings of the 28th international conference on Software engineering (ICSE), pp. 1055–1056. ACM (2006). DOI http://dx.doi.org/10.1145/1134285.1134502
23. Gelenbe, E., Zhang, Y.: Performance optimization with energy packets. IEEE Systems Journal **13**(4), 3770–3780 (2019)
24. Gelenbe, E., Zhang, Y.: Sharing energy for optimal edge performance. In: SOFSEM 2020: Theory and Practice of Computer Science, pp. 24–36. Springer International Publishing, Cham (2020)
25. Giger, E., Pinzger, M., Gall, H.C.: Can we predict types of code changes? An empirical analysis. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 217–226 (2012). DOI 10.1109/MSR.2012.6224284. ISSN: 2160-1852
26. Godfrey, M.W., German, D.M.: The past, present, and future of software evolution. In: Frontiers of Software Maintenance (FoSM), pp. 129–138. IEEE (2008). DOI http://dx.doi.org/10.1109/FOSM.2008.4659256
27. Gondra, I.: Applying machine learning to software fault-proneness prediction. Journal of Systems and Software **81**(2), 186–195 (2008). DOI http://dx.doi.org/10.1016/j.jss.2007.05.035
28. Goulão, M., Fonte, N., Wermelinger, M., e Abreu, F.B.: Software evolution prediction using seasonal time analysis: a comparative study. In: 16th European Conference on Software Maintenance and Reengineering (CSMR), pp. 213–222. IEEE (2012). DOI http://dx.doi.org/10.1109/CSMR.2012.30
29. Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., Williams, B.: The correspondence between software quality models and technical debt estimation approaches. In: Sixth International Workshop on Managing Technical Debt (MTD), pp. 19–26. IEEE (2014). DOI 10.1109/MTD.2014.13
30. He, Z., Shu, F., Yang, Y., Li, M., Wang, Q.: An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering **19**(2), 167–199 (2012). Publisher: Springer
31. Izurieta, C., Vetrò, A., Zazworka, N., Cai, Y., Seaman, C., Shull, F.: Organizing the Technical Debt Landscape. In: Proceedings of the Third International Workshop on Managing Technical Debt (MTD), MTD '12, pp. 23–26. IEEE Press (2012). DOI 10.5555/2666036.2666040. Event-place: Zurich, Switzerland
32. Kadioglu, Y.M., Gelenbe, E.: Product-form solution for cascade networks with intermittent energy. IEEE Systems Journal **13**(1), 918–927 (2018)
33. Kalouptsoglou, I., Siavvas, M., Tsoukalas, D., Kehagias, D.: Cross-project vulnerability prediction based on software metrics and deep learning. In: Computational Science and Its Applications – ICCSA 2020, pp. 877–893. Springer International Publishing, Cham (2020)
34. Kenmei, B., Antoniol, G., Di Penta, M.: Trend analysis and issue prediction in large-scale open source systems. In: 12th European Conference on Software Maintenance and Reengineering (CSMR), pp. 73–82. IEEE (2008). DOI http://dx.doi.org/10.1109/CSMR.2008.4493302

35. Khoshgoftaar, T.M., Allen, E.B., Deng, J.: Using regression trees to classify fault-prone software modules. IEEE Transactions on Reliability **51**(4), 455–462 (2002). DOI http://dx.doi.org/10.1109/TR.2002.804488
36. Kitchenham, B.A., Mendes, E., Travassos, G.H.: Cross versus Within-Company Cost Estimation Studies: A Systematic Review. IEEE Transactions on Software Engineering **33**(5), 316–329 (2007)
37. Kouros, P., Chaikalis, T., Arvanitou, E.M., Chatzigeorgiou, A., Ampatzoglou, A., Amanatidis, T.: Jcaliper: search-based technical debt management. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 1721–1730 (2019)
38. Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE **68**(9), 1060–1076 (1980). DOI http://dx.doi.org/10.1109/PROC.1980.11805
39. Lenarduzzi, V., Saarimäki, N., Taibi, D.: The technical debt dataset. In: Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 2–11 (2019)
40. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software pp. 193–220 (2015). DOI http://dx.doi.org/10.1016/j.jss.2014.12.027
41. Malhotra, R., Lata, K.: On the Application of Cross-Project Validation for Predicting Maintainability of Open Source Software using Machine Learning Techniques. In: 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), pp. 175–181. IEEE (2018). DOI 10.1109/ICRITO.2018.8748749
42. Maragos, K., Lentaris, G., Soudris, D.: In-the-field mitigation of process variability for improved fpga performance. IEEE transactions on Computers **68**(7), 1049–1063 (2019)
43. Marinescu, R.: Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development **56**(5), 9–1 (2012). DOI 10.1147/JRD.2012.2204512
44. Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., Cok, D.: Local vs. global models for effort estimation and defect prediction. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 343–351 (2011)
45. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th international conference on Software engineering (ICSE), pp. 452–461. ACM (2006). DOI http://dx.doi.org/10.1145/1134285.1134349
46. Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A.: On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. Empirical Software Engineering **23**(3), 1188–1221 (2018). DOI http://dx.doi.org/10.1007/s10664-017-9535-z
47. Papadopoulos, L., Marantos, C., Digkas, G., Ampatzoglou, A., Chatzigeorgiou, A., Soudris, D.: Interrelations between software quality metrics, performance and energy consumption in embedded applications. In: Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, pp. 62–65 (2018)
48. Raja, U., Hale, D.P., Hale, J.E.: Modeling software evolution defects: a time series approach. Journal of Software Maintenance and Evolution: Research and Practice **21**(1), 49–71 (2009). DOI http://dx.doi.org/10.1002/smr.398
49. Roumani, Y., Nwankpa, J.K., Roumani, Y.F.: Time series modeling of vulnerabilities. Computers & Security **51**, 32 – 40 (2015). DOI http://dx.doi.org/10.1016/j.cose.2015.03.003. URL http://www.sciencedirect.com/science/article/pii/S0167404815000358
50. Sas, D., Avgeriou, P., Fontana, F.A.: Investigating instability architectural smells evolution: an exploratory case study. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 557–567. IEEE (2019)
51. Siavvas, M., Gelenbe, E.: Optimum Checkpointing for Long-running Programs. In: 15th China-Europe International Symposium on Software Engineering Education (2019)
52. Siavvas, M., Gelenbe, E.: Optimum checkpoints for programs with loops. Simulation Modelling Practice and Theory **97**, 101951 (2019)
53. Siavvas, M., Gelenbe, E.: Optimum interval for application-level checkpoints. In: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 145–150. IEEE (2019)

54. Siavvas, M., Gelenbe, E., Kehagias, D., Tzovaras, D.: Static analysis-based approaches for secure software development. In: Security in Computer and Information Sciences, pp. 142–157. Springer International Publishing, Cham (2018)
55. Siavvas, M., Marantos, C., Papadopoulos, L., Kehagias, D., Soudris, D., Tzovaras, D.: On the Relationship between Software Security and Energy Consumption. In: 15th China-Europe International Symposium on Software Engineering Education (2019)
56. Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D., Anicic, N., Gelenbe, E.: An empirical evaluation of the relationship between technical debt and software security. In: 9th International Conference on Information Society and Technology (ICIST), vol. 2019 (2019)
57. Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Tzovaras, D.: Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises. Enterprise Information Systems **0**(0), 1–43 (2020). DOI 10.1080/17517575.2020.1824017
58. Skourletopoulos, G., Mavromoustakis, C.X., Bahsoon, R., Mastorakis, G., Pallis, E.: Predicting and quantifying the technical debt in cloud software engineering. In: 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pp. 36–40. IEEE (2014). DOI http://dx.doi.org/10.1109/CAMAD.2014.7033201
59. Tan, J., Lungu, M., Avgeriou, P.: Towards Studying the Evolution of Technical Debt in the Python Projects from the Apache Software Ecosystem. In: 17th Belgium-Netherlands Software Evolution Workshop (BENEVOL), pp. 43–45 (2018)
60. Tsoukalas, D., Jankovic, M., Siavvas, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D.: On the Applicability of Time Series Models for Technical Debt Forecasting. In: 15th China-Europe International Symposium on Software Engineering Education (CEISEE 2019) (2019). DOI 10.13140/RG.2.2.33152.79367. (in press)
61. Tsoukalas, D., Kehagias, D., Siavvas, M., Chatzigeorgiou, A.: Technical Debt Forecasting: An empirical study on open-source repositories. In: Journal of Systems and Software, vol. 170, p. 110777 (2020). DOI https://doi.org/10.1016/j.jss.2020.110777. URL http://www.sciencedirect.com/science/article/pii/S0164121220301904
62. Tsoukalas, D., Mathioudaki, M., Siavvas, M., Kehagias, D., Chatzigeorgiou, A.: A Clustering Approach towards Cross-project Technical Debt Forecasting - Supporting Material (2020 (accessed October 1, 2020)). URL https://sites.google.com/view/clustering-td-forecasting/appendix
63. Tsoukalas, D., Siavvas, M., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D.: Methods and Tools for TD Estimation and Forecasting: A State-of-the-art Survey. In: International Conference on Intelligent Systems (IS 2018), pp. 698–705. IEEE (2018). DOI http://dx.doi.org/10.1109/IS.2018.8710521
64. Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering **14**(5), 540–578 (2009). Publisher: Springer
65. Vetro', A.: Using Automatic Static Analysis to Identify Technical Debt. In: Proceedings of the 34th International Conference on Software Engineering (ICSE), ICSE '12, pp. 1613–1615. IEEE Press (2012). DOI 10.5555/2337223.2337499. Event-place: Zurich, Switzerland
66. Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a Fault Prediction Model to Allow Inter Language reuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08, pp. 19–24. Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1370788.1370794. URL https://doi.org/10.1145/1370788.1370794. Event-place: Leipzig, Germany
67. Xygkis, A., Soudris, D., Papadopoulos, L., Yous, S., Moloney, D.: Efficient winograd-based convolution kernel implementation on edge devices. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2018)
68. Yazdi, H.S., Mirbolouki, M., Pietsch, P., Kehrer, T., Kelter, U.: Analysis and prediction of design model evolution using time series. In: International Conference on Advanced Information Systems Engineering (CAiSE), pp. 1–15. Springer (2014). DOI 10.1007/978-3-319-07869-4_1

69. Zazworka, N., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F., others: Comparing four approaches for technical debt identification. Software Quality Journal **22**(3), 403–426 (2014). DOI https://doi.org/10.1007/s11219-013-9200-8