

Comparing Univariate and Multivariate Time Series Models for Technical Debt Forecasting

Maria Mathioudaki, Dimitrios Tsoukalas(✉), Miltiadis Siavvas, and Dionysios Kehagias

Centre for Research and Technology Hellas, Thessaloniki, Greece
{mariamathi,tsoukj,siavvasm,diok}@iti.gr

Abstract. Technical debt (TD) is a successful and widely used metaphor that expresses the quality compromises that can yield short-term benefits but may negatively affect the overall quality of a software product in the long run. There is a vast variety of techniques and methodologies that have been proposed over the past years to enable the identification and estimation of TD during the software development cycle. However, it is only until recently that researchers have turned towards the investigation of methods that focus on forecasting its future evolution. Getting insights on the future evolution of TD can enable on-time decision-making and allow stakeholders to plan preventive strategies regarding TD repayment. In our previous studies, we have investigated time series analysis and Machine Learning techniques in order to produce reliable TD forecasts. In our current attempt, we aim to explore the capabilities of a statistical ARIMA model both in a univariate and a multivariate fashion. More specifically, the present paper investigates whether the adoption of an ARIMA model that takes into account, in addition to the TD value itself, various TD-related indicators may lead to more accurate TD predictions than its univariate alternative. For this purpose, dedicated models are constructed, evaluated, and compared on a dataset of five long-lived, open-source software applications.

Keywords: Software quality · Technical debt · Forecasting · Time series

1 Introduction

In 1992, Ward Cunningham coined the term Technical Debt (TD) [1], to characterize technical issues of a software application that are not immediately obvious to managers and end-users, but impede developers' productivity by requiring extra maintenance effort. Poor design and implementation decisions, as well as technical shortcuts to reduce time-to-market, are common causes of TD. TD, like financial debt, must be reduced early in the software development life cycle (SDLC) by applying code refactorings. If not done so, then it can accumulate

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13380, pp. 62–78, 2022.
https://doi.org/10.1007/978-3-031-10542-5_5

interest in the form of future software changes or improvements that will be more expensive or even impossible to implement. In that sense, it has been discovered that wasted effort due to TD can account for up to 23% of the total developers' time [2], and in extreme cases, it can lead to an unmaintainable software [3].

TD Management is a critical but difficult challenge. To this direction, researchers have examined a wide range of methodologies and tools for detecting, estimating, and repaying TD throughout the overall SDLC [4]. However, research towards the concept of *forecasting* the future evolution of a software's TD has only recently initiated [5]. In the software development market, systematic efforts towards this direction can be proven of great value, as foreseeing how TD will evolve in the future can help project managers and developers to react quickly to its accumulation and, thus, to keep software quality at a satisfactory level [5].

In previous studies [6–10], we have conducted research towards the TD identification and forecasting concept, by examining different methods ranging from simple statistical time series models to more sophisticated approaches, such as machine learning (ML) and deep learning (DL). Our findings revealed that ML and DL algorithms are suitable for making accurate TD predictions for both short- and long-term forecasting horizons. These complex models however, despite their apparent predictive capability, lack of interpretability (black box), are computationally expensive and are prone to overfitting.

To this end, in the context of this study we revisit the concept of employing time series analysis for TD forecasting. However, unlike our previous related study [7] that focused on *univariate* time series models, i.e., models that require only the past evolution of TD itself as input, this study examines whether the adoption of *multivariate* time series models, i.e., models with exogenous variables, may lead to more accurate TD predictions. In this endeavour, our multivariate models receive as input not only the TD value, but also various TD-related indicators and their combinations. Subsequently, their performances are compared with those of the univariate approach (acting as the baseline model), on a dataset of five long-lived, open-source software applications. Finally, the results are evaluated and discussed in order to reach safer conclusions.

2 Related Work

Gaining an understanding on the future evolution of a software system might be difficult, but it can also be highly beneficial for software businesses trying to deal with rising complexity and diminishing software quality in their products [11]. As a result, the concept of analyzing, gaining insights, and being able to forecast the evolution of different aspects of a software project, either directly or indirectly associated with TD, has been extensively examined by researchers in recent years (e.g. code smells [12], number of changes [13], software security [14], and defects prediction [15]). The vast majority of these efforts attempt to solve the problem by employing statistical time series or ML models on individual software attributes based on data analysis (e.g., historical data, trends, source code metrics, etc.).

One of the most widely used methodologies for understanding the evolution of software-related quality attributes and properties, is time series analysis. In a study by Yazdi et al. [13], the authors model software design evolution of numerous applications using ARMA time series and conclude that these models may accurately forecast the changes of future software updates. Similarly, Kenmei et al. [16] apply time series models to accurately estimate the evolution of future change requests using data from three large open source applications. Likewise, Goulo et al. [17] use data acquired from the Eclipse's tracking system for developing a time series model that forecasts the evolution of change requests. Moreover, they identify seasonal trends to validate that including seasonal information greatly improves the model's performance. In a different study, Raja et al. [15] apply time series techniques to forecast software defect-proneness, based on a dataset of defect reports from eight software projects. They point out that these models can facilitate budget and resource allocation planning. Finally, Antoniol et al. [18] build an ARIMA model to forecast the evolution of software metrics, such as size and complexity, based on related historical data from the Linux Kernel. Their findings suggest that forecasting the evolution of these software metrics can be utilized to improve the management of software artifacts or processes.

The significance of quality prediction in the software engineering domain is demonstrated by the numerous studies for predicting the evolution of software quality parameters, either directly or indirectly associated with TD. However, as a software system evolves, so does the amount of its accumulated TD. In a study by Tsoukalas et al. [5], the authors highlight the need for forecasting the evolution of TD and emphasize the importance of future research on this topic. The same authors contribute to the TD forecasting challenge by examining a variety of different forecasting techniques, ranging from simple statistical time series models to more complex ML and DL approaches [7–10]. Notably, in an initial endeavor to predict the TD evolution of long-lived open-source software projects, the authors present a single univariate ARIMA model [7] and conclude that it can yield reliable short-term TD forecasts for each of the five projects studied. They do observe, however, that the model's accuracy drops significantly for forecasting horizons greater than eight steps ahead. In an attempt to extend their previous work and introduce a more holistic TD forecasting methodology, the same authors conduct two follow-up studies [8,10], where they examine the ability of ML and DL methods to predict the TD evolution of 15 open-source projects. Their findings showcase that a nonlinear Random Forest regression can achieve sufficient accuracy for forecasts up to 40 horizons ahead [8], while a more complicated Multi-layer Perceptron model can produce reliable predictions for up to 150 steps ahead into the future [10].

3 Methodology and Experimental Setup

This section presents the overall methodology that was followed in the present paper. More specifically, it provides detailed information on the employed

dataset, the data pre-processing, the model construction and parameter tuning, and last but not least, the strategy that was followed for the experimental execution.

3.1 Dataset and Preparation

The first step of the proposed methodology is the construction of the dataset. For the purposes of TD forecasting, the dataset needs to consist of multiple software projects along with their history of TD measurements (TD and TD indicators' values), across their commit history. This data will serve as input to train and evaluate our univariate and multivariate models.

Hence, we decided to exploit the "Technical Debt dataset"[1], i.e., a dataset that was constructed and made publicly available to facilitate TD research [19]. This dataset is based on 33 real-world open-source Java projects obtained from the Apache Software Foundation and contains TD measurements for all available commits of the aforementioned projects. TD measurements were obtained using SonarQube[2], a widely-used open-source static code analysis platform that has been used as proof of concept in multiple research studies [4]. Each one of the 33 projects is provided as a csv file, where the columns contain the software-related metrics (including TD measurements), as extracted from SonarQube, and the rows correspond to the different commits through time.

In the context of the present work, we decided to select five long-lived Apache projects, namely Ambari, Commons-codec, Commons-io, Httpcomponents-client and Mina-sshd, for constructing and evaluating our TD forecasting models. We limited the number of examined projects to 5 out of 33, as the construction of individual time series models tailored to each dataset requires extensive manual effort with respect to ARIMA modeling, parameterization, and analysis. Eventually, the selection criteria were based on the projects' commit life (at least 3 years of evolution) and commit frequency (the higher the better). The selected projects are presented in Table 1, accompanied by additional information on the total number of analyzed commits and the analysis lifetime.

Pre-processing. In order to apply any time series model for forecasting purposes, fixed time intervals between the retrieved samples (i.e., commits) need to be ensured. This part is critical for the reliability of the models. However, the datasets used in this study are obtained from real-world applications. This means that, in some cases, multiple commits may have taken place during a particular period of time, whereas in other cases, a long period of inactivity (i.e. no commits at all) may have occurred. To alleviate this issue, we applied a filtering method in order to ensure that the final datasets will include time points corresponding to one commit per week, maintaining in this way fixed time intervals between the samples. Specifically, if multiple commits have taken place during a week, we isolate and keep the last commit of the week. If a period of more than one week

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

has passed without commits, we assign to that week the last available commit. By applying the aforementioned approach, we maintain the initial life span of a project, but reduce the frequency of commits to one per week. More details on the final number of samples (i.e., commits) per each dataset (i.e., project) are presented in Table 1.

Table 1. Selected projects of the TD dataset

Project name	Analyzed commits	Analysis lifetime	Filtered commits
Ambari	13397	09/11 – 06/15	192
Commons-codec	1726	04/03 – 05/18	532
Commons-io	2118	01/02 – 06/18	632
Httpcomponents-client	2867	12/05 – 06/18	576
Mina-sshd	1370	12/08 – 06/18	360

Feature Selection. As already mentioned, the purpose of the present study is to compare the performance of *univariate* time series models to that of time series models that follow a *multivariate* modeling approach. The rationale behind this, is that we are interested in investigating whether models that take into account various TD-related indicators and their combinations, may be able to discover underlying patterns on the TD evolution more effectively than models that consider only the TD value itself.

That being said, to apply any form of analysis following the multivariate approach, the independent variables need to be carefully selected. There is a total of 61 software-related metrics in the dataset that could potentially act as independent variables for the current problem. However, providing a model with such an input may result to what is called the "curse of dimensionality", which is the unlikely event of a significant drop in the model's predictive performance when a large dimension of variables is provided as input. Consequently, features that are not (or are slightly) associated with the target variable, i.e., total TD, should be filtered out before any model construction attempt.

To identify the most important TD indicators that could act as strong predictors for TD forecasting, in one of our previous studies [9] (where the same dataset was used for similar purposes), we performed an extensive feature selection analysis. More specifically, different feature selection algorithms were put in practice, including filter-based, wrapper-based and embedded methods. The results led to the selected TD indicators presented in Table 2. Therefore, the same TD indicators will be used as independent variables also in this study, during the next steps of the methodology.

That being said, similarly to our previous work [8], we also specify the *total technical debt* as our target variable, i.e., the variable that we will generate forecasts for. We define *total_technical_debt* as the total remediation effort (in minutes) that is required in order to fix all reliability, maintainability and security issues detected in the source code of a software project.

Table 2. Metrics 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_technical_debt	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 Forecasting with ARIMA and ARIMAX Models

A time series is a collection of consecutive observations made at equally spaced time intervals. Prior to any time series analysis, stationarity needs to be assured, so that the statistical properties of the time series data (e.g., mean, variance, and auto-correlation) become constant over time. In this respect, Box and Jenkins introduced the Autoregressive Integrated Moving Average (ARIMA) technique for modeling non-stationary time series data [20]. These models work with both stationary and non-stationary data and can be easily adjusted in cases where the underlying patterns alter significantly. Since then, ARIMA models have been widely employed for modeling software evolution [13,15–17].

ARIMA models usually require only the historical data of the target variable, i.e., the variable that we are interested in making forecasts for (univariate approach). However, they can also be extended to take into account "exogenous" variables, i.e., independent variables that may act as predictors for the target variable. This type of models are called ARIMAX and are essentially the multivariate version of the univariate ARIMA. The methodology that follows in this part is applied for both the ARIMA and the ARIMAX modeling approach.

ARIMA (and thus ARIMAX) models are fine-tuned by specifying three integers, namely p, d and q. Parameter p is the order of the autoregressive model

(AR), representing the number of values in the past that affect the current value of the time series. Parameter d represents the integrated (I) part of the model. It stands for the number of times the time series needs to be differenced so that it becomes stationary. Lastly, parameter q is the order of the moving-average (MA) model. It basically describes that the current error depends on previous errors with some lag equal to q. There are four key steps that form the Box and Jenkins ARIMA modelling strategy:

Identification: Before parameterizing the ARIMA model, stationarity of the time series needs to be ensured. Thus, during this step, if the observed data is found to be non-stationary, the effect can be removed by differencing the time series. The number of times that a series needs to be differenced in order to become stationary corresponds to the d parameter.

Estimation: During this step, the p and q parameters can be determined by inspecting the Auto-Correlation Function (ACF) and Partial Auto-Correlation Function (PACF) plots. The ACF correlogram reveals the correlation between the residuals of the data at specific lags. However, identifying the p and q parameters via the ACF and PACF plots is not always a trivial task. In such cases, the optimal p and q parameters are specified in the next step, by comparing the *goodness of fit* of various competing models.

Diagnostic Testing: During this step, appropriate tests are performed in order to define the best ARIMA model. Specifically, the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) are used and the goal is to minimize these scores through the testing of numerous ARIMA models with different p and q combinations.

Application: The final and most crucial step is to apply the selected model on unseen observations that were not used during the previous steps. Hence, to ensure the ability of the model to generalize well and, thus, to generate accurate predictions, it is of paramount importance to hold out some observations for evaluation purposes.

Performance Metrics. As in any forecasting task, predictions need to be as close to the real values as possible, while avoiding model overfitting. For evaluating the models' predictive power, similarly to our previous works [7–9] we used the *Mean Absolute Percentage Error* (MAPE). MAPE is a commonly used measure for the evaluation of forecasting models that uses absolute values to calculate the error in percentage terms.

To complement the evaluation of our results and at the same time remain inline with our previous studies [7–9], we also calculated the *Mean Absolute Error* (MAE) and the *Root Mean Squared Error* (RMSE), whose presence is usual when it comes to forecasting tasks. MAE calculates the magnitude of the errors coming from a group of forecasts by not taking under consideration their trend. RMSE represents the quantification of the quadratic mean of the differences between the observed and the predicted values.

4 Analysis, Results and Discussion

This section presents the results regarding the five different projects that comprise our dataset. Due to space limitation, we offer a detailed description of the ARIMA modelling approach for the Mina-sshd project. For the rest of the projects we provide only the final results for comparison purposes and discussion but we note that the exact same steps were followed.

4.1 Mina-sshd

Identification: As explained in Sect. 3.2, the first step of the time series analysis is to visualize the time series so that we can gain an initial insight on its nature. Figure 1 (a) illustrates the TD evolution of the Mina-sshd dataset over time.

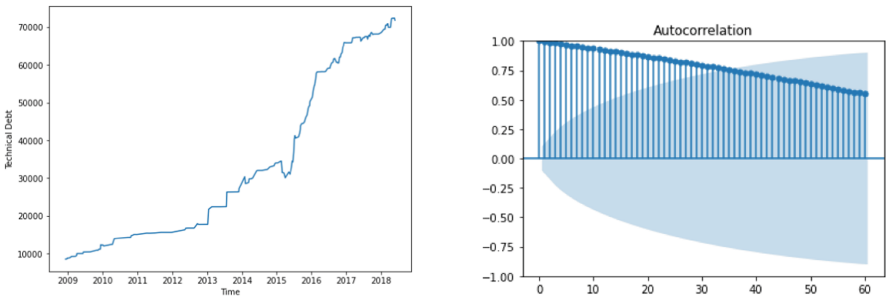

Fig. 1. TD evolution and ACF Correlogram of Mina-sshd original data

By inspecting Fig. 1 (a), it is clear that the data is non-stationary with an upward trend, i.e., the TD value of Mina-sshd increases over time. This can be further supported by the fact that the ACF chart depicted in Fig. 1 (b) is characterized by the slow linear decay in the spikes. Therefore, we need to make the time series stationary before proceeding to further analysis. A common way for making a series stationary is to transform it through *differencing*. Therefore, we took the first-order difference of the data to eliminate the overall trend from the series. If the original series is Y_t, then the differenced series is indicated by $Y_t = Y_t - Y_{t-1}$. The ACF and PACF plots of the first-order differenced time series are presented in Fig. 2.

To ensure that a first-order differencing made the time series stationary, we employed the Dickey-Fuller test [21], which tests the null hypothesis that a unit root is present in an autoregressive model. The results of the Dickey-Fuller test showed that the "Test Statistic" value is lower that the "Critical Value" (with p-value < 0.05), which means that the time series has become stationary. This is further supported by the illustrated first-order differenced

data on Fig. 2 (a) which demonstrates no apparent trend. As mentioned earlier, the number of transformations that take place until the time series becomes stationary corresponds to the d parameter of the *ARIMA(p,d,q)* model. Hence, setting the value of $d = 1$ can be safely supported by the above analysis.

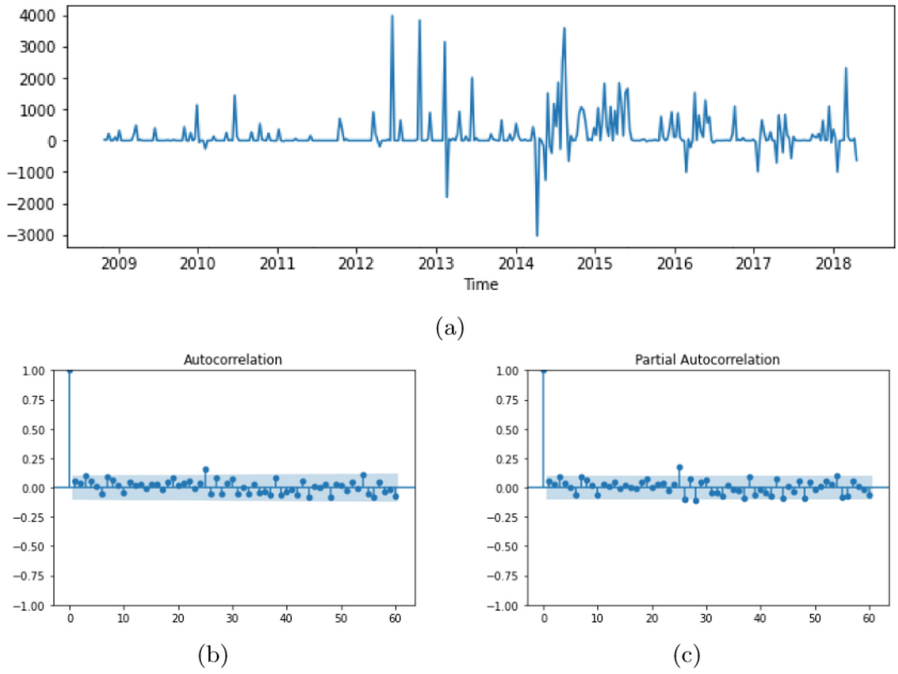

Fig. 2. Dickey-Fuller, ACF and PACF of the first-order differenced time series data

Estimation: During this step, we aim to identify the p (AR) and q (MA) parameters of the *ARIMA(p,d,q)* model, following the practical recommendations through visual inspection of the ACF and PACF correlograms [22]. For this purpose, we analyzed Fig. 2 (b) and (c) representing the ACF and PACF correlograms of the first-order differenced time series. In general, the x-axis of the ACF plot indicates the lag at which the autocorrelation is computed, while, the y-axis indicates the value of the correlation (between -1 and 1). In our case, the two plots indicate no clear cut-off as all correlation values are found to be within the intervals of 95% confidence, except for a few exceptions which occur for rather high lag values. In this case, p and q parameters will be determined during our next step, i.e., the *Diagnostic testing*.

Diagnostic testing: As mentioned in Sect. 3.2, when no clear cut-off is indicated in the ACF and PACF correlograms, we turn towards the AIC and BIC error minimization criteria. Therefore, we conducted experiments by assigning various combinations of values (between 0 and 5) to the p and q parameters, while

keeping the d parameter equal to 1. We excluded ARIMA (0,1,0) from our analysis as this is the "random walk" model and, thus, the simplest one. By inspecting the results, the models that provided the lowest AIC and BIC values were ARIMA (1,1,0), ARIMA (0,1,1) and ARIMA (1,1,1). The results of the three best performing models can be found in Table 3.

Table 3. ARIMA candidate model results for Mina-sshd

Model	AIC	BIC	Ljung-box	Prob	*Coef.* P > | z |
ARIMA (0,1,1)	5639.89	5647.65	1.16	0.28	***ma*** 0.001
ARIMA (1,1,0)	5638.97	5646.74	1.97	0.16	***ar*** 0.000
ARIMA (1,1,1)	5616.71	5628.36	0.14	0.70	***ar*** 0.000 ***ma*** 0.000

The coef P > | z | column of Table 3 indicates the significance of each feature weight. As can be seen, all of the *AR* and *MA* parameters of the examined models have p-values lower than 0.05. This means that it is both reasonable and safe to consider any of these candidate models for our experiments. In addition, Table 3 also presents the Ljung-Box [23] statistics for residual analysis. Ljung-Box Q test is considered one of the most common goodness of fit tests. As can be seen, the Ljung-Box test of the models provides values higher than 0.05, which indicates that the residuals are independent and all three models are suitable and well-adjusted to the time series.

Since the above metrics cannot facilitate our decision on the selection of best model, we focus our efforts on the AIC and BIC indicators. AIC and BIC measure the goodness of fit of statistical models for a given set of data by estimating the quality of each model, relative to each of the other models. Models that have a better fit receive a better (lower) AIC and BIC score than models that fit worse. By inspecting Table 3, we can see that while the AIC and BIC values of the three models are similar, for ARIMA(1,1,1) the values are slightly lower (better), i.e. 5616.71 and 5628.36 respectively.

To conclude, the three models seem identical and their scores are close. However, the fact that ARIMA(1,1,1) has a slightly better AIC, BIC and Ljung-Box Q test score indicates that it is a more appropriate candidate. Therefore, we chose ARIMA(1,1,1) as the most suitable model. The next step is to evaluate the performance of the selected model on the Mina-sshd dataset.

Application: The fourth and final step of the ARIMA modelling methodology is the Application. During this step, the model is trained on a subset of the data, and then it is tested on "unseen" observations to ensure the ability of the model to generalize well. In time series analysis tasks, *Walk-forward validation* [24] is a commonly used technique to assess the models accuracy, based on the concept that models are updated when new observations are made available (i.e., by respecting the temporal order in which the samples were collected). It involves the split of the timeseries at *n-1* equally distanced points. In this way, n folds are

Table 4. ARIMA(1,1,1) and ARIMAX(1,1,1) forecasting results for Mina-sshd

Model	Steps ahead	MAPE %	RMSE	MAE
ARIMA(1,1,1)	4	0.683	290.279	268.227
	8	1.197	485.614	432.432
	12	1.518	575.076	483.292
	Total average	**1.132**	**450.323**	**394.650**
ARIMAX(1,1,1)	4	0.240	122.872	102.095
	8	0.539	300.040	249.715
	12	0.538	296.198	232.803
	Total average	**0.438**	**239.703**	**194.871**

formed. Initially, the first fold is used to train a model whose accuracy is tested against the consecutive folds. Then, the first fold is merged with the second one and a new model is trained and tested against the consecutive remaining folds. The process is repeated until there is only one fold left to serve as a test set. Subsequently, *n-1* errors are obtained and the average of these errors is calculated to obtain one single value representing the overall performance of the model. In our case, taking under consideration that the filtered datasets are characterized by an average length, we chose the number of splits to be 3.

Therefore, we used the *Walk-forward validation* to conduct our experiments, by employing ARIMA(1,1,1) as the baseline model and ARIMAX(1,1,1) as the model that would prove that the inclusion of exogenous variables (i.e., TD-related indicators) in the ARIMA process may lead to more accurate TD predictions than its univariate alternative. We compared their results using MAPE, MAE and RMSE. In Table 4, we present the results for multiple (4, 8 and 12) time steps into the future. The results indicate that indeed the multivariate ARIMAX(1,1,1) model provides us with remarkably lower errors compared to its univariate alternative ARIMA(1,1,1). The average values of the errors for the different steps ahead are calculated to enable a smoother assessing of the results.

4.2 Ambari, Commons-codec, Commons-io, Httpcomponents-client

Identification: The steps that we followed during Sect. 4.1, were also repeated for the rest of the four datasets (i.e., software projects). Initially, by employing Dickey-Fuller tests for each one of the datasets we confirmed that a first-order difference is sufficient for the four time series to become stationary. Therefore, as explained in detail earlier, the d parameter of the ARIMA(p,d,q) model for all of the projects can be set to $d = 1$.

Estimation: During this phase of the analysis, we need to specify the (AR) p and (MA) q parameters for each of the datasets. We analyzed the corresponding ACF and PACF plots, but similarly to the Mina-sshd case, we were not able to draw safe conclusions as there were no clear cut-offs. Consequently, we turned

again towards the minimization criteria approach by testing competitive ARIMA models, while keeping d equal to 1 according to our previous step. The three best performing models for each one of the datasets are the following:

- *Ambari*: ARIMA(1,1,0), ARIMA(0,1,1) and ARIMA(1,1,1).
- *Commons-codec*: ARIMA(1,1,1), ARIMA(2,1,1) and ARIMA(2,1,2).
- *Commons-io*: ARIMA(1,1,1), ARIMA(2,1,1) and ARIMA(1,1,2).
- *Httpcomponents-client*: ARIMA(0,1,1), ARIMA(1,1,0) and ARIMA(2,1,2).

Diagnostic Testing: During this step, we compare the best performing models in terms of goodness of fit and residuals. Through this process we aim to conclude on which of the candidate ARIMA models is more suitable for each dataset. The summary of the fitted models is presented on Table 5.

Table 5. ARIMA candidate model results for the rest of the projects

Model	AIC	BIC	Ljung-box	Prob	*Coef.* P > | z |
Ambari					
ARIMA(0,1,1)	4418.97	4425.47	0.09	0.77	***ma*** 0.927
ARIMA(1,1,0)	4418.92	4425.42	0.10	0.75	***ar*** 0.917
ARIMA(1,1,1)	4418.51	4428.27	0.02	0.88	***ar*** 0.000 ***ma*** 0.014
Commons-codec					
ARIMA(1,1,1)	7158.41	7171.23	0.76	0.38	***ar*** 0.000 ***ma*** 0.000
ARIMA(2,1,1)	7152.83	7169.93	0.00	0.98	***ar*** 0.748 ***ma*** 0.000
ARIMA(2,1,2)	7152.35	7173.73	0.67	0.41	***ar*** 0.191 ***ma*** 0.931
Commons-io					
ARIMA(1,1,1)	9817.35	9830.69	0.08	0.77	***ar*** 0.000 ***ma*** 0.000
ARIMA(2,1,1)	9819.13	9836.92	0.00	0.98	***ar*** 0.921 ***ma*** 0.000
ARIMA(1,1,2)	9819.21	9836.99	0.00	0.97	***ar*** 0.000 ***ma*** 0.918
Httpcomponents-client					
ARIMA(0,1,1)	9894.14	9902.85	0.06	0.80	***ma*** 0.928
ARIMA(1,1,0)	9894.13	9902.84	0.06	0.81	***ar*** 0.928
ARIMA(2,1,2)	9895.41	9917.19	0.19	0.66	***ar*** 0.647 ***ma*** 0.626

After carefully inspecting the results and mainly the AIC values, we identified that for the Ambari dataset the optimal model is the ARIMA(1,1,1), for the Commons-io dataset the ARIMA(1,1,1) and for the Httpcomponents-client the ARIMA(1,1,0). In the case of the Commons-codec dataset, we observed that while the lowest AIC is obtained using the ARIMA(2,1,2) and ARIMA(2,1,1) models, their AR and MA parameters have p-values higher than 0.05 (see coef P > | z | column), which raises awareness on the goodness of fit of these models. For this reason, we decided to choose the third model, namely the ARIMA(1,1,1). We note that the coef P of the AR and MA parameters are higher than 0.05 also in the case of the Httpcompoents-client dataset. However, since this applies to all candidate models, we chose the model with the lowest AIC value.

Diagnostic Testing: During final step, similarly to the case of Mina-sshd, we compared the best performing ARIMA models for each of the datasets with their corresponding multivariate (ARIMAX) alternatives. Again, predictions were generated for 4, 8 and 12 versions ahead. The obtained errors are presented on Table 6. By inspecting Table 6, it is clear that the outcome that was obtained while inspecting the results for Mina-sshd is in line with the performance results obtained also for the rest of the datasets. More specifically, in all cases, the results

Table 6. ARIMA(1,1,1) and ARIMAX(1,1,1) forecasting results for the rest of the projects

Model	Steps ahead	MAPE %	RMSE	MAE
Ambari				
ARIMA(1,1,1)	4	3.698	4023.305	3744.349
	8	5.435	9457.667	7840.039
	12	6.759	11959.119	10389.987
	Average	**5.297**	**8480.030**	**7324.791**
ARIMAX(1,1,1)	4	1.237	1411.385	1296.111
	8	2.712	2663.183	2133.263
	12	3.440	2904.759	2456.603
	Average	**2.463**	**2326.442**	**1961.991**
Commons-codec				
ARIMA(1,1,1)	4	0.545	62.362	56.972
	8	0.911	106.136	95.587
	12	1.462	200.555	157.840
	Average	**0.972**	**123.017**	**103.466**
ARIMAX(1,1,1)	4	0.051	7.479	6.827
	8	0.308	43.414	32.988
	12	0.608	95.008	69.639
	Average	**0.322**	**48.633**	**36.484**
Commons-io				
ARIMA(1,1,1)	4	0.318	108.159	56.337
	8	3.515	994.881	717.263
	12	4.804	1173.283	965.836
	Average	**2.879**	**758.774**	**579.812**
ARIMAX(1,1,1)	4	0.197	24.022	21.461
	8	0.417	51.200	43.942
	12	0.696	99.657	78.653
	Average	**0.683**	**159.083**	**127.635**
Httpcomponents-client				
ARIMA(1,1,0)	4	3.032	3694.202	1948.633
	8	7.000	5691.715	4535.235
	12	8.343	6236.742	5420.759
	Average	**6.125**	**5207.553**	**3968.209**
ARIMAX(1,1,0)	4	1.120	915.976	745.284
	8	1.427	1046.078	922.350
	12	1.543	1077.187	977.793
	Average	**1.363**	**1013.080**	**881.809**

indicate that the multivariate ARIMAX models provide significantly lower errors when compared to their univariate alternative ARIMA models. Therefore, we can safely conclude that by bringing into the equation, in addition to the TD value itself, various TD-related indicators and their combinations, time series TD forecasting models can be significantly improved.

In order to visualize the above conclusions, Fig. 3 provides an example of forecasting the TD evolution of the Mina-sshd project for 35 commits ahead, using ARIMA(1,1,1) and ARIMAX(1,1,1). As can be seen, both models generated predictions that are close to the ground truth. However, the ARIMAX(1,1,1) model has managed to catch the fluctuations of the original time series much more accurately. Hence, as far as our dataset is concerned, the ARIMAX(1,1,1) models are able to capture the underlying TD evolution patterns in a much more effective way compared to the ARIMA(1,1,1).

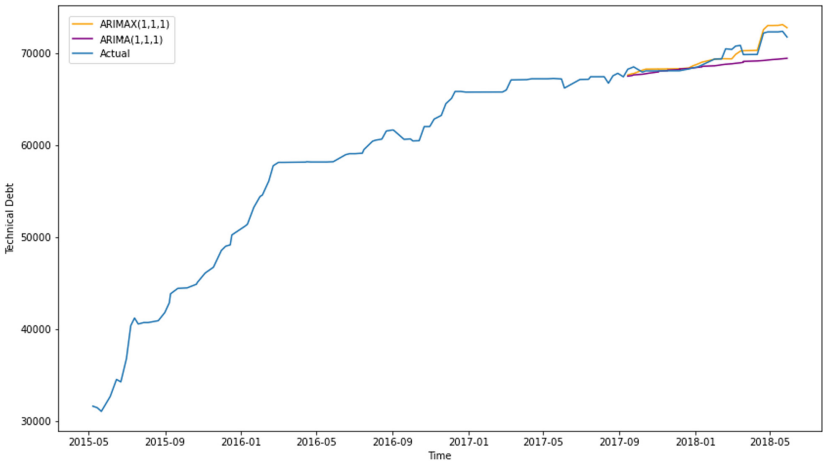

Fig. 3. Mina-sshd TD forecasting for 35 versions ahead using ARIMAX(1,1,1) and ARIMA(1,1,1)

4.3 Discussion

Across the five independent datasets, we have successfully applied the Box and Jenkins modelling strategy to identify the optimal ARIMA and ARIMAX models and compare their performances. Results showed that both models are able to fit and provide meaningful forecasts of TD evolution. However, when comparing their performance metrics, it was observed that a more complex model that relies on the historical data of both the target variable and the related independent variables outperforms a model that relies only on the target variable. Furthermore, this outcome was consistent for all forecasting horizons tested across all

five projects. Relying also on our previous work, we note that time series analysis consists a useful technique for modeling the TD evolution of long-lived software projects. Through the present study we conclude that when exogenous variables are considered during the model construction, accuracy is significantly improved.

5 Conclusions

The purpose of the current work, was to extend our knowledge on the predictive performance of time series models for the purposes of TD forecasting. More specifically, we strove towards getting an understanding on whether ARIMAX models that take under consideration independent variables acting as TD predictors can provide more accurate TD forecasts than common ARIMA models. For the purpose of our study we employed 5 datasets containing the evolution history of software-metrics, computed from static code analysis for 5 long-lived projects. Based on this data, we generated forecasts for different horizons using ARIMA and ARIMAX models. The obtained results lead to a single concrete conclusion. Across the 5 independent datasets, the accuracy of the ARIMAX models was significantly higher than the accuracy provided by the ARIMA models. Thus, we conclude that taking under consideration exogenous variables when employing ARIMA models for TD forecasting purposes, escalates the model's accuracy.

While our previous studies revealed that ML and DL approaches provide reliable forecasts for both short- and long-term TD predictions, the fact that they lack interpretability and are computationally"hungry" limits their practicality. Therefore, in the context of the current study, we turned our focus towards statistical time series models, focusing on rather short-term forecasting horizons, as our main goal was to compare the performances of the two ARIMA approaches (i.e., univariate and multivariate). However, the promising results obtained using ARIMAX models urge for further investigation. To this end, as a future work, we plan to generalize the outcomes of this research towards broader forecasting horizons, in order to directly compare ARIMAX to the ML and DL models of our previous studies, potentially contributing to new milestones that can and should be added in the TD forecasting field.

Acknowledgements. This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through SmartCLIDE project under Grant Agreement No. 871177.

References

1. Cunningham, W.: The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger **4**(2), 29–30 (1993)
2. Besker, T., Martini, A., Bosch, J.: Software developer productivity loss due to technical debt-A replication and extension study examining developers' development work. J. Syst. Softw. **156**, 41–61 (2019)
3. Suryanarayana, G., Samarthyam, G., Sharma, T.: Refactoring for Software Design Smells: Managing Technical Debt. Morgan Kaufmann (2014)

4. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. J. Syst. Soft. **101**, 193–220 (2015)
5. 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) (2018)
6. Tsoukalas, D., et al.: Machine learning for technical debt identification. IEEE Trans. Soft. Eng. (2021)
7. 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) (in press)
8. Tsoukalas, D., Kehagias, D., Siavvas, M., Chatzigeorgiou, A.: Technical Debt Forecasting: an empirical study on open-source repositories. J. Syst. Soft. **170**, 110777 (2020)
9. Tsoukalas, D., Mathioudaki, M., Siavvas, M., Kehagias, D., Chatzigeorgiou, A.: A clustering approach towards cross-project technical debt forecasting. SN Comput. Sci. **2**(1), 1–30 (2021)
10. Mathioudaki, M., Tsoukalas, D., Siavvas, M., Kehagias, D.: Technical debt forecasting based on deep learning techniques. In: Gervasi, O., et al. (eds.) ICCSA 2021. LNCS, vol. 12955, pp. 306–322. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-87007-2_22
11. Gall, H.C., Lanza, M.: Software evolution: analysis and visualization. In: 28th International Conference On Software Engineering, pp. 1055–1056. ACM (2006)
12. Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empir. Softw. Eng. **21**(3), 1143–1191 (2015). https://doi.org/10.1007/s10664-015-9378-4
13. Shariat Yazdi, H., Mirbolouki, M., Pietsch, P., Kehrer, T., Kelter, U.: Analysis and prediction of design model evolution using time series. In: Iliadis, L., Papazoglou, M., Pohl, K. (eds.) CAiSE 2014. LNBIP, vol. 178, pp. 1–15. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07869-4_1
14. 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 (2020)
15. Raja, U., Hale, D.P., Hale, J.E.: Modeling software evolution defects: a time series approach. J. Softw. Maint. Evol. Res. Pract. **21**(1), 49–71 (2009)
16. Kenmei, B., Antoniol, G., Di Penta, M.: Trend analysis and issue prediction in large-scale open source systems. In: 2008 12th European Conference on Software Maintenance and Reengineering, pp. 73–82. IEEE (2008)
17. Goulão, M., Fonte, N., Wermelinger, M., Abreu, F.B.: Software evolution prediction using seasonal time analysis: a comparative study. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 213–222. IEEE (2012)
18. Antoniol, G., Di Penta, M., Gradara, S.: Predicting Software Evolution: An Approach and a Case Study
19. Lenarduzzi, V., Saarimäki, N., Taibi, D.: The technical debt dataset. In: 15th International Conference on Predictive Models and Data Analytics, pp. 2–11 (2019)
20. Box, G.E.P., Jenkins, G.M.: Time Series Analysis: Forecasting and Control, pp. 161–215. Palgrave Macmillan, UK (2013)
21. Dickey, D.A., Fuller, W.A.: Distribution of the estimators for autoregressive time series with a unit root. J. Am. Stat. Assoc. **74**(366), 427–431 (1979)
22. McCleary, R., Hay, R.: Applied Time Series Analysis For The Social Sciences. Sage Publications, Beverly Hills (1980)

23. Ljung, G.M., Box, G.E.P.: On a measure of lack of fit in time series models. Biometrika **65**(2), 297–303 (1978)
24. Stone, M.: Cross-validatory choice and assessment of statistical predictions. J. Royal Stat. Soc. Ser. B (Methodol.) **36**(2), 111–147 (1974)