On the Applicability of Time Series Models for Technical Debt Forecasting

Dimitrios Tsoukalas*†, Marija Jankovic*, Miltiadis Siavvas*, Dionysios Kehagias*,
Alexander Chatzigeorgiou†, Dimitrios Tzovaras*
* *Centre for Research and Technology Hellas, Thessaloniki, Greece*
† *Department of Applied Informatics, University of Macedonia, Thessaloniki 54643, Greece*
tsoukj@iti.gr, jankovicm@iti.gr, siavvasm@iti.gr, diok@iti.gr, achat@uom.gr, dimitrios.tzovaras@iti.gr

***Abstract*—Technical debt (TD) is commonly used to indicate additional costs caused by quality compromises that can yield short-term benefits in the software development process, but may negatively affect the long-term quality of software products. Predicting the future value of TD could facilitate decision-making tasks regarding software maintenance and assist developers and project managers in taking proactive actions regarding TD repayment. However, no notable contributions exist in the field of TD forecasting, indicating that it is a scarcely investigated field. This study constitutes an initial attempt towards this direction. To this end, in the present study, we empirically evaluate the ability of time series analysis to model and predict TD evolution. To create our dataset, we obtain weekly snapshots of five open source software projects over three years and compute their TD values. We find that the autoregressive integrated moving average model ARIMA(0,1,1) can provide accurate predictions over a fairly long time period for all sampled projects. The model can be used to facilitate planning for software evolution budget and time allocation. The approach presented in this paper provides a basis for predictive TD analysis, suitable for projects with a relatively long history.**

***Keywords*—technical debt, technical debt forecasting, time series analysis, ARIMA**

I. Introduction

The Technical Debt (TD) metaphor, a term inspired by the financial debt of economic theory, was introduced in 1992 by Ward Cunningham [1] to describe the problem of making quality compromises that can yield short-term benefits in the software development process, but may negatively affect the long-term quality of software products. The TD metaphor was initially related to software implementation (i.e. at the code level) but was gradually extended to other phases of the software development lifecycle (SDLC), i.e. software architecture, design, documentation, requirements, and testing [2]. In the same manner like financial debt, TD incurs interest payments in the form of increased future software costs usually caused by poor design and code quality. To effectively manage the identification, quantification, and repayment of TD during the SDLC, researchers and practitioners have developed and adopted a multitude of theories, methods and tools [3].

However, prediction of accumulated TD during the software evolution is an open and challenging research issue, as both software system and its TD emerge in parallel [4]. A possibility to predict TD is primarily crucial for software maintainability, which is recognized as one of the most-effort intense activities in the SDLC [5]. System engineers and project managers need the right tools and appropriate training support to be able to perform long-term effective software maintenance [5]. Hence, forecasting the evolution of TD could be valuable for estimating the point in which the software product could become unmaintainable.

Although various aspects, which are relevant for the TD concept, such as code smells [6], fault-proneness [7] and evolution trends [8], have attracted the attention of both academia and industry, to the best of our knowledge no studies are focusing on TD itself [9]. Hence, a method or tool that would provide practical decision-making support by predicting future TD of a software system in uncertainty is of eminent importance. Consequently, software architects and project managers would be able to gain a better understanding of future TD issues and plan appropriate refactoring activities.

The SDK4ED European project aims to address this challenging issue by developing a TD Forecasting toolbox for various design-time and run-time software qualities, as a part of integrated Energy Optimization and TD Elimination framework. As a first step towards TD Forecasting toolbox realization, we have developed and applied specific time series models for TD forecasting. Time series models have been widely used in previous studies for predicting software evolution trends, future change requests or software defects [8], [10]–[13]. To this end, in the present paper we attempt to empirically evaluate the ability of time series to adequately forecast future TD trends. The problem that the present work attempts to solve can be summarized in the following research question:

> ***RQ***: *Is the usage of time series models a valid and accurate approach to forecasting Technical Debt in a long-lived, open-source software?*

A positive answer to this question will suggest that time series models can potentially be used as the basis for the construction of the TD forecasting toolbox. We will also investigate the extent to which these models can properly capture the evolution of TD values.

In order to provide answers to the research question mentioned above, we conducted an empirical study. In particular, we initially constructed a relatively large code repository comprising five real-world open-source Java applications retrieved

from the GitHub[1] online repository. For each application, we collected 150 snapshots (commits) in weekly intervals, spanning up to 3 years of each system's evolution. This approach led to a dataset containing up to 750 snapshots in total. For each snapshot, we calculated the TD value using SonarQube[2], a popular static code analysis tool. Subsequently, for each application, we employed the Box-Jenkins modelling method to construct and identify the parameters of our models. Finally, we compared the accuracy of these models with a random walk model for various steps ahead into the future in order to reach safer conclusions regarding the significance of the observed results. To the best of our knowledge, this is the first study in the field of TD that examines the applicability of time series models for TD forecasting.

The rest of the paper is structured as follows: Section II discusses the related work in the field of TD forecasting and background on time series forecasting. Section III describes the experiment setup, while Section IV presents the analysis and a discussion on the results of the experiment. Finally, Section V concludes the paper and discusses ideas for future work.

II. Background

A. Software Evolution and TD Forecasting Trends

Various researchers have addressed the topic of forecasting the evolution of various aspects of a software project (e.g. code smells [6], fault-proneness [7], [14]–[16]) which are directly or indirectly related to the TD concept. Gaining a higher level of information about the evolution of large software systems is a critical challenge in dealing with increasing complexity and decreasing software quality [17]. For this reason, the various attempts to analyze, understand and predict the evolution of a software system, have increased considerably in the last years [8], [10]–[13].

In a recent study, Chaikalis and Chatzigeorgiou [8] employ Network Models to forecast software evolution trends of Java systems and evaluate the predictive power of the proposed model against the actual evolution of 10 open-source projects. They conclude that the achieved accuracy in the prediction of several network and software properties appears to be promising. A commonly used technique to analyze the evolution of software systems is time series. In their study, Yazdi et al. [10] model the evolution of the design of software systems by applying ARMA time series to several typical projects successfully. Based on the empirical results the authors point out that time series models can predict the future changes of the next revisions of the systems with good accuracies. In another study, Raja et al. [11] use the time series approach to predict defects in software evolution. They use defect reports for eight open source projects and build time series models to predict software defects which leads to the conclusion that the model may be used to facilitate planning for software evolution budget and time allocation. Likewise, Goulão et al. [12] build a time series model to forecast the change requests evolution based on data collected from Eclipse's change request tracking system. Additionally, they include the identification of seasonal patterns and tendencies, which is important to validate that usage of seasonal information significantly improves the estimation ability of this model, when compared to other ARIMA models. Finally, in [13] Kenmei et al. use time series models to forecast future change requests evolution and to identify trends based on data collected from three large open source applications. They highlight that time series are capable to model change requests and act as a support tool for project staffing and planning.

In addition to time series analysis, a multitude of studies address the problem of forecasting the evolution of various aspects of a software project by employing machine learning techniques. In their study Fontana et al. [6] compare 16 different supervised machine learning techniques for code smell detection using over 74 software systems. Regarding fault-proneness prediction, in a study conducted by Arisholm et al. [7], the authors propose a multivariate model for predicting fault-prone components of object-oriented, legacy systems by using history change and fault data from previous releases. Moreover, in [14], Gondra et al. propose the use of machine learning to predict software fault-proneness. Their approach first employs sensitivity analysis to select software metrics that are more likely to indicate the existence of errors, and afterward, trains an Artificial Neural Network (ANN) to predict future fault-proneness. In a relative study, Nagappan et al. [15] use principal component analysis on code metrics to build regression models that accurately predict the likelihood of post-release defects. Finally, Khoshgoftaar et al. [16] use regression and classification trees to identify fault and not-fault prone modules on multiple releases of a large scale legacy telecommunications system, concluding that these algorithms result in predictions with satisfactory accuracy and robustness.

The multitude of models that are available in the literature for predicting the evolution of specific quality attributes and quality properties reveal the importance of quality prediction and forecasting in the software engineering community. However, no concrete approaches have been proposed so far regarding the forecasting of TD evolution. The need for forecasting the evolution of TD has been highlighted by a recent study, in which Tsoukalas et al. [9] raise the awareness of the gap in the field of TD. They claim that an interesting topic would be to investigate different efficient ways to produce TD forecasting models for accurate prediction of TD principal and interest evolution. In addition, they stress that it would be useful to examine if TD forecasting could foster the development of high-quality software products.

B. Forecasting with ARIMA models

A time series is a collection of consecutive observations made at equally spaced time intervals. A fundamental assumption in time series analysis is stationarity, as statistical properties such as mean, variance, and autocorrelation are constant over time. Box and Jenkins introduced the Autoregressive

[1]https://github.com

[2]https://www.sonarqube.org/

Integrated Moving Average (ARIMA) technique to deal with the modeling of non-stationary time series [18]. ARIMA models have been successfully used in software evolution modelling [10]–[13] for forecasting based on historical data. These models are suited for stationary and non-stationary series and can be adjusted easily if significant changes take place in the trends.

The ARIMA model is parameterized by adjusting three distinct integers: p, d and q. Parameter $\boldsymbol{p}$ represents the auto-regressive (AR) part of the model, i.e., regression of the time series onto itself. Basic assumption is that current series values depend on previous values with some lags, where p is the maximum lag in the model. Parameter $\boldsymbol{d}$ stands for the integrated (I) part of the model and incorporates the amount of differencing (i.e. the number of past time points to subtract from the current value) to apply to the time series. Parameter $\boldsymbol{q}$ is the moving average (MA) part of the model. Basic assumption is that current error depends on the previous with some lag, which is referred to as q. This allows to set the error of the model as a linear combination of the error values observed at previous time points in the past. The Box and Jenkins ARIMA modelling strategy involves four steps:

Identification: In order to successfully apply an ARIMA model for prediction, the observed time series has to be analyzed for stationarity. If non-stationarity is identified during this process, the effect can be removed by differencing or seasonal differencing the series. The number of differences performed before a time series becomes stationary corresponds to the d parameter of the model.

Estimation: Once stationarity is ensured, the next step is to plot the Auto-Correlation Function (ACF) and Partial Auto-Correlation Function (PACF) of the time series to determine the appropriate values of p and q parameters. The ACF correlogram reveals the correlation between the residuals of the data at specific lags. Then, the actual time series has to be modelled using the ARIMA(p,d,q) parameters previously defined.

Diagnostic testing: Once a series has been estimated, the next step is to test the model against competing models. Appropriate tests, such as fit statistics and goodness of fit are used to select the best model, by analyzing the residuals of the series for any possible correlations.

Application: During a training phase, the model is optimized for the data it is built from. As a final step, it is critical to test the model on observations that have not been used to train it. Hence, to ensure the ability of the model to generalize well, it is important to hold out some observations that can be used to evaluate the predictive power of the model on unseen data.

III. Experimental Setup and Methodology

A. Data Collection

The data used in this study were obtained from five popular open source projects from GitHub repository, namely Apache Kafka, Apache SystemML, Square OkHttp, Square Retrofit and Jenkins. The selection criteria were based on the popularity, activity level, data availability, and the Java programming language. For each system, we collected 150 snapshots (commits) in weekly intervals, spanning up to 3 years of each system's evolution.

B. Variable Specification

In the present work, we decided to use the Software Quality Assessment based on Lifecycle Expectations (SQALE) method [19] for quantifying the TD of the selected software products. Specifically, we use the SQALE Index plugin that is provided by SonarQube, a popular open source platform for continuous inspection of code quality. The SQALE Index quantifies the effort that is required to fix all the code violations (i.e. code smells, bugs and vulnerabilities) that reside in the analyzed software product. SonarQube expresses the effort in minutes. Finally, in order to avoid being biased by the size of the software products, similarly to [4], we decided to express the SQALE Index as a ratio i.e., SQALE Index divided by size (lines of code) of the software products. We term the normalized SQALE Index as TD Density and we model it via time series.

C. ARIMA Technique

Although there exist several forecasting methods that have proven to yield high predictive power, like Artificial Neural Networks (ANNs), support vector regression (SVR), or Regression Trees (RT), the application of these methods usually requires extensive parameter tuning and computational power. Moreover, these methods depend on the availability and reliability of data on independent variables over the forecasting period, which requires further efforts in data collection and estimation. Thus, as an initial approach towards TD forecasting we decided to employ time series models. The reason is that time series models, compared to machine learning models, are more straightforward, less computationally expensive and less data hungry, in a sense that they provide another modelling approach, which only requires the historical data of the variable of interest to forecast its future evolution behavior.

IV. Analysis, Results and Discussion

This section reports the empirical study results for the five analyzed projects. Due to space limitation, we describe the ARIMA approach in detail only for the Apache Kafka project, while for the rest of the projects we present only the results of the selected models.

A. Apache Kafka

1) Identification: As described in Section III, the first step is the time series plot, which provides an understanding of the nature of the series. When visualizing the weekly Apache Kafka TD evolution over the last three years, we can observe an increasing variation of TD density values. To eliminate this variation and linearize the series, we performed a natural $log(ln)$ transformation. As a result, the fluctuations in the transformed series were far less than the original series.

After the logarithmic transformation, we removed the mean and analyzed the transformed series for stationarity, i.e. seasonality and trends. The decomposition of time series is a

statistical task that deconstructs a time series into several components, each representing one of the underlying categories of patterns, i.e. trend, seasonality, and residual components of the data. By applying seasonal decomposition to the series, the existence of trend and seasonality becomes even more clear. This can be depicted in Fig. 1.

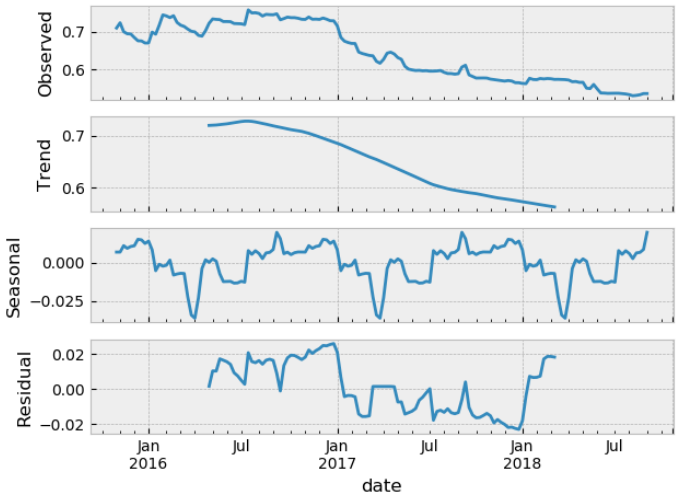

Fig. 1. Seasonal decomposition of the Technical Debt evolution of the Apache Kafka project

By looking at Fig. 1, we observe that the seasonal component and the decreasing trend of the data are nicely separated, leading to the conclusion that the series is not stationary in nature and needs to be adjusted in order to successfully apply an ARIMA model for prediction. The most common practice for making a series stationary is to transform the series through differencing. The process below describes all the required steps to make the series stationary and involves ACF and PACF correlograms analysis, as well as Dickey–Fuller tests [20], which test the null hypothesis that a unit root is present in an autoregressive model. Detailed results of the Dickey-Fuller test on original data are presented in TABLE I.

TABLE I
DICKEY-FULLER TEST ON ORIGINAL DATA

Test Statistic	-0.572788
p-value	0.877013
Critical Value (1%)	-3.475325
Critical Value (5%)	-2.881275
Critical Value (10%)	-2.577293

For a time series to pass the stationarity test, the "Test Statistic" value should be lower than the "Critical Value". For the original time series, clearly this is not the case. This can be further supported by the fact that the ACF chart is characterized by the slow linear decay in the spikes. As a next step, we performed the test on the ln-transformed time series rather than the original. Detailed results of the Dickey-Fuller test on ln-transformed data are presented in TABLE II.

TABLE II
DICKEY-FULLER TEST ON LN-TRANSFORMED DATA

Test Statistic	-0.453466
p-value	0.900782
Critical Value (1%)	-3.475325
Critical Value (5%)	-2.881275
Critical Value (10%)	-2.577293

While the ln-transformation helped to improve the stationarity of the data, it did not completely eliminate the fluctuations and alternating factors. Again, the ACF chart is characterized by a slow linear decay in the spikes. The next step is to take a 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 first-order differenced time series are presented in Fig. 2, while Dickey-Fuller test results are presented in TABLE III.

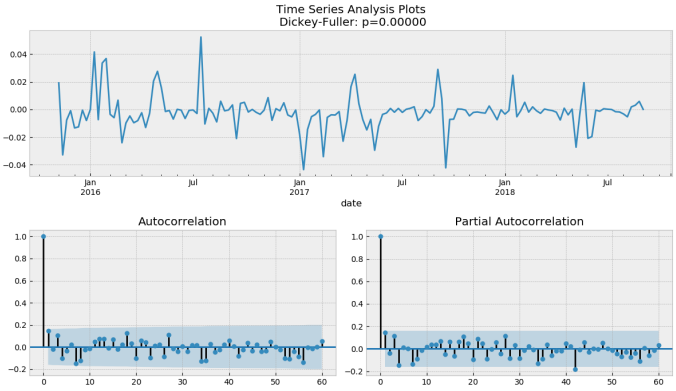

Fig. 2. Dickey-Fuller, ACF and PACF of the first-order differenced time series data

TABLE III
DICKEY-FULLER TEST ON FIRST-ORDER DIFFERENCED DATA

Test Statistic	-10.53053
p-value	9.177006e-19
Critical Value (1%)	-3.475325
Critical Value (5%)	-2.881275
Critical Value (10%)	-2.577293

The table above indicates that the time series is now stationary, as the Test Statistic value is lower that the Critical Value. The significance of p–*value* (p<0.05) also confirms that taking the first-order difference has made the data stationary. This is also illustrated on the plot itself, as there is no visible trend. The number of required transformations until the series becomes stationary corresponds to the d parameter of the ARIMA(p,d,q) model, thus setting the value of d = 1 can be safely supported by the analysis performed in this step.

2) Estimation: During the previous analysis, we applied first-order differencing on the original data to make it stationary and then identified the d parameter of the ARIMA model.

During this phase of the analysis, we have to identify the auto-regressive (AR) p and moving average (MA) q parameters. The selection process of these parameters is not trivial task, but we followed the practical recommendations for selection of parameters p and q through visual inspection of the ACF and PACF correlograms [21]. In short, if the ACF of the series disappear gradually, and the PACF of the series disappear abruptly, it indicates an AR component. An opposite behavior, i.e., ACF disappear abruptly and PACF disappear gradually, indicates an MA component. If both ACF and PACF disappear gradually or disappear abruptly, various models can be tested to identify the p and q parameters accurately.

We analyzed the ACF and PACF plots of the first-order differenced values illustrated in Fig. 2 in accordance with the ARIMA guidelines. In general, the x-axis of the ACF plot indicates the lag at which the autocorrelation is computed. The y-axis indicates the value of the correlation (between -1 and 1). For the dataset, both ACF and PACF plots indicate a cut off at lag 1. In this case, three competing models, ARIMA(1,1,0), ARIMA(0,1,1) and ARIMA(1,1,1), need to be compared using goodness of fit tests and residual analysis to accurately identify the p and q parameters. We excluded ARIMA(0,1,0) from further analysis as we used it as a "random walk" for the purpose of comparing it with the selected model during the application step. The next step is to compare the three models for goodness of fit by applying fit statistics and select the most suitable one based on the optimal results.

3) Diagnostic testing: During this phase of the ARIMA analysis, we analyzed goodness of fit of the selected models as well as the residuals (i.e., the difference between the predicted and the actual values) for any possible correlations to verify adequacy of these models. A summary of the fitted models ARIMA(1,1,0), ARIMA(0,1,1) and ARIMA(1,1,1) is presented in TABLE IV.

TABLE IV
ARIMA CANDIDATE MODEL RESULTS

Model	**Model Results**				
	AIC	***Ljung-Box***	***Prob***	***coef*** $P > \|z\|$	
ARIMA(0,1,1)	-987.52	39.62	0.49	***ma***	0.001
ARIMA(0,1,1)	-985.69	40.66	0.44	***ar***	0.016
ARIMA(1,1,0)	-985.366	41.19	0.42	***ma***	0.727
				ar	0.880

The coef $P > |z|$ column indicates the significance of each feature weight. MA parameter of the first model has a *p–value* below 0.05, so it is reasonable to retain it in our model. The same applies to AR parameter of the second model. However, the AR and MA parameters of the third model have the *p–value* above 0.05, which indicates that there is room for adjustments, and that retaining both AR and MA parameters may decrease the predictive performance of the model.

One of the most common goodness of fit tests is the Ljung–Box [22] statistics for residual analysis. The Ljung–Box Q test indicates, through its high significance value (>0.05), that the residuals are independent and all three models are suitable and well-adjusted to the time series. Another indicator that can facilitate our selection process is the Akaike information criterion (AIC) [23], which measures the goodness of fit of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Hence, models that have a better fit will receive a better (lower) AIC score than similar models that fit worse. Both ARIMA(1,1,0) and ARIMA(1,1,1) models have similar AIC scores, with values -985.687 and -985.366 respectively. For ARIMA(0,1,1), the value is slightly lower (better), i.e. -987.523. In general, the three models seem identical and their scores are close. However, the fact that the first model has a slightly better AIC and Ljung–Box Q test score indicates that it is a more appropriate candidate. Therefore, we chose ARIMA(0,1,1) as the most suitable model.

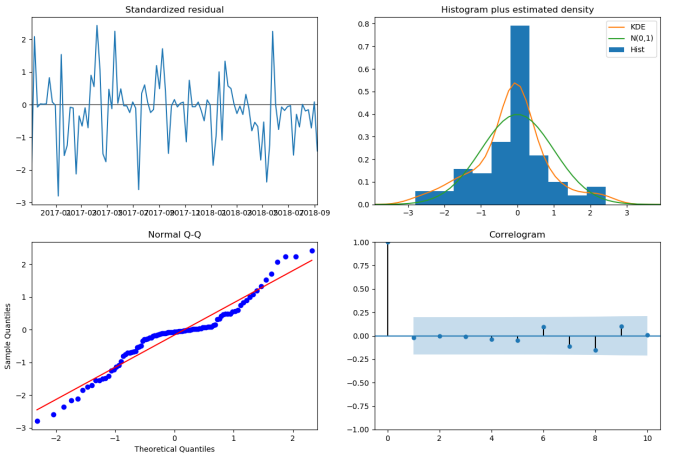

Fig. 3. Residual analysis results of the ARIMA(0,1,1) time series forecasting model

A residual analysis also evaluates the model's goodness of fit and helps to investigate for any unusual behavior. Fig. 3 presents the residual analysis and diagnostics of the ARIMA(0,1,1) model. The primary concern is to ensure that the residuals of the model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved. In our case, the model diagnostics suggests that the model residuals are normally distributed based on the following:

- In the top right plot, the red kernel density estimation (KDE) line follows the N(0,1) line. This is a good indication that the residuals are normally distributed.
- The QQ-plot on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). This is also an indication that the residuals are normally distributed.
- The residuals over time (top left plot) do not display any obvious seasonality. This can be further confirmed by the ACF on the bottom right, which shows that the time series

residuals have low correlation with lagged versions of itself.

Those observations led us to conclude that the model produces a satisfactory fit that could help forecast future values. The next step is to evaluate the selected model's performance on the dataset.

4) Application: The fourth and final ARIMA modelling step is Application. During this step, the selected model is optimized for the data it is built from and then tested on observations that have not been used during training to ensure the ability of the model to generalize well. For this purpose, it is important to hold out some observations that can be used to evaluate the predictive power of the model on unseen data. Validation methods extensively used in machine learning, such as k-fold cross-validation, cannot be directly used with time series data due to the temporal order in which values were observed. Hence, observations cannot be randomly split into groups without respecting the temporal order.

To assess prediction accuracy and compare different models we adopted walk-forward validation [24], a strategy inspired by k-fold cross-validation. Walk-forward validation is a commonly used way to evaluate time series models' performance, based on the notion that models are updated when new observations are made available. In brief, during walk-forward validation a subset of n consecutive points extracted from the original time series is used to train an initial model. Then, accuracy of the model is tested against future time steps and prediction is evaluated against the known value to compute prediction errors. Finally, the time window is moved one-step forward to include the known value into the training set and the process is repeated.

We choose the n = 52 (one year) as our sliding training window. Then, we applied three independent walk-forward validation processes, where predictions were made for the next n+4 (1 month), n+8 (2 months), and n+12 (3 months) future steps respectively. In time series analysis, it is common to include a random walk model for the purpose of comparing it with the selected model. The random walk model excludes the auto-regressive (AR) and moving average (MA) parameters. Since the proposed model is ARIMA(0,1,1), the random walk model to be used is ARIMA(0,1,0).

We evaluated forecasts for both, the proposed model as well as the random walk model using Root Mean Squared Error (RMSE). The benefit of RMSE is that it penalizes large errors and the scores are in the same units as the forecast values (TD density per week). We also computed Mean Absolute Percentage Error (MAPE) as well as Mean Absolute Error (MAE). In TABLE V, we report a comparison of prediction errors of both, our selected model as well as the random walk model for multiple (4, 8 and 12) time steps into the future.

Results indicate that the model is stable over the holdout sample for all steps ahead and that the proposed model outperforms the random walk model for 4 and 8 steps ahead. Of course, the predictive power of this modelling approach decreases as we forecast longer into the future. For 12 steps ahead, RMSE and MAE errors are equal for both models, while the MAPE error of our model is bigger that the MAPE of the random walk.

TABLE V
ARIMA(0,1,1) AND RANDOM WALK MODEL COMPARISON

Model	Steps ahead	Model Fit statistics		
		RMSE	***MAPE***	***MAE***
ARIMA(0,1,1)	4	0.010	1.349	0.008
	8	0.017	2.430	0.014
	12	0.025	3.623	0.021
ARIMA(0,1,0)	4	0.011	1.372	0.010
	8	0.018	2.455	0.015
	12	0.025	3.612	0.021

B. Apache SystemML, Square OkHttp, Square Retrofit and Jenkins

1) Identification: For the rest of the applications, similar to the Apache Kafka case, we performed a $log(ln)$ and first-order differencing to eliminate the non-stationarity of the series. The results of applying a Dickey-Fuller test on the initial and first-order differenced time series are presented in TABLE VI.

TABLE VI
DICKEY-FULLER TESTS ON FIRST-ORDER DIFFERENCED DATA

Series	Test Statistic (Critical Value)
Apache SystemML	
Initial	-1.043719 (3.475018)
1st difference	-11.31556 (3.475018)
Square OkHttp	
Initial	-1.800630 (-3.456355)
1st difference	-10.03932 (-3.456355)
Square Retrofit	
Initial	-1.426334 (-3.498198)
1st difference	-10.24958 (-3.498198)
Jenkins	
Initial	-1.796879 (-3.475637)
1st difference	-9.991324 (-3.475637)

TABLE VI indicates that for every project, taking the first-order difference of the values has made the time series stationary. The Dickey-Fuller test statistic is lower than the critical value so we reject the null hypothesis of unit root. As stated above, the number of required transformations until the series becomes stationary corresponds to the d parameter of the ARIMA(p,d,q) model, thus setting the value of d = 1 can be safely supported by the above analysis.

2) Estimation: During this phase of the analysis, the auto-regressive (AR) p and moving average (MA) q parameters have to be defined. We analyzed the ACF and PACF plots of the four series in accordance with the ARIMA guidelines in Section III to accurately identify the p and q parameters. Based on this analysis, competitive models for each project were identified. These models are presented below:

- *Apache SystemML*: ARIMA(1,1,0), ARIMA(0,1,1) and ARIMA(1,1,1).

- *Square OkHttp*: ARIMA(0,1,2), ARIMA(1,1,2), ARIMA(0,1,1), ARIMA(1,1,0) and ARIMA(1,1,1).
- *Square Retrofit*: ARIMA(1,1,0), ARIMA(0,1,2), ARIMA(0,1,1) and ARIMA(1,1,1).
- *Jenkins*: ARIMA(1,1,0), ARIMA(0,1,2), ARIMA(0,1,1) and ARIMA(1,1,1).

3) Diagnostic testing: During this phase of the ARIMA analysis, goodness of fit of the selected models as well as the residuals (i.e., the difference between the predicted and the actual values) have to be analyzed for any possible correlations to verify adequacy of these models. A summary of the fitted models for each project is presented in TABLE VII.

TABLE VII
ARIMA CANDIDATE MODEL RESULTS

Model	Model Results				
	AIC	***Ljung-Box***	***Prob***	***coef***	$P > \|z\|$
Apache SystemML					
ARIMA(0,1,1)	**-1265.54**	**25.99**	**0.96**	***ma***	**0.562**
ARIMA(1,1,0)	-1265.49	25.99	0.96	***ar***	0.581
ARIMA(1,1,1)	-1264.89	24.90	0.97	***ma***	0.124
				ar	0.056
Square OkHttp					
ARIMA(0,1,1)	**-1069.41**	**6.89**	**1.00**	***ma***	**0.040**
ARIMA(1,1,0)	-1068.50	9.14	1.00	***ar***	0.626
ARIMA(0,1,2)	-1052.98	8.06	1.00	***ma***	0.504
ARIMA(1,1,2)	-1067.47	6.96	1.00	***ar***	0.899
				ma	0.342
ARIMA(1,1,1)	-1067.81	7.96	1.00	***ar***	0.752
				ma	0.709
Square Retrofit					
ARIMA(0,1,1)	**-632.89**	**27.70**	**0.93**	***ma***	**0.035**
ARIMA(1,1,0)	-632.87	27.90	0.93	***ar***	0.926
ARIMA(0,1,2)	-631.93	26.92	0.94	***ma***	0.494
ARIMA(1,1,1)	-631.32	27.10	0.94	***ar***	0.156
				ma	0.099
Jenkins					
ARIMA(0,1,1)	**-1047.50**	**34.84**	**0.70**	***ma***	**0.042**
ARIMA(1,1,0)	-1047.22	35.57	0.67	***ar***	0.103
ARIMA(0,1,2)	-1046.86	29.55	0.89	***ma***	0.116
ARIMA(1,1,1)	-1045.99	33.71	0.75	***ar***	0.854
				ma	0.793

Models with the lowest AIC value are presented in bold. Based on the AIC test score we identified ARIMA(0,1,1) as the best candidate for all projects under analysis. A residual analysis also evaluated that the residuals of the selected model are uncorrelated and normally distributed with zero-mean. Those observations lead us to conclude that ARIMA(0,1,1) produced a satisfactory fit that could help forecast future values for the four projects under analysis.

4) Application: The fourth and final ARIMA modelling step is Application. Similarly, to the Kafka project, we compared our models to the random walk. We choose the $n = 52$ (one year) as our sliding training window. Then, we applied three independent walk-forward validation processes, where predictions were made for the next n+4 (1 month), n+8 (2 months), and n+12 (3 months) future steps respectively. In TABLE VIII, a comparison of prediction errors is reported for multiple (4, 8 and 12) time steps into the future for each project.

TABLE VIII
ARIMA(0,1,1) AND RANDOM WALK MODEL COMPARISON

Model	Steps ahead	Model Fit statistics		
		RMSE	***MAPE***	***MAE***
Apache SystemML				
ARIMA(0,1,1)	4	0.005	0.476	0.003
	8	0.006	0.740	0.005
	12	0.008	1.024	0.006
ARIMA(0,1,0)	4	0.005	0.482	0.004
	8	0.006	0.741	0.006
	12	0.008	1.016	0.006
Square OkHttp				
ARIMA(0,1,1)	4	0.006	0.998	0.004
	8	0.010	1.502	0.006
	12	0.012	2.074	0.009
ARIMA(0,1,0)	4	0.008	1.348	0.006
	8	0.010	1.784	0.007
	12	0.011	1.921	0.008
Square Retrofit				
ARIMA(0,1,1)	4	0.011	1.215	0.007
	8	0.015	2.082	0.013
	12	0.020	3.000	0.018
ARIMA(0,1,0)	4	0.011	1.223	0.008
	8	0.016	2.096	0.014
	12	0.020	2.948	0.018
Jenkins				
ARIMA(0,1,1)	4	0.009	0.962	0.005
	8	0.012	1.602	0.009
	12	0.016	2.271	0.013
ARIMA(0,1,0)	4	0.011	0.987	0.007
	8	0.012	1.599	0.009
	12	0.016	2.283	0.013

As in the Apache Kafka case, results indicate that the ARIMA(0,1,1) model is stable over the holdout sample for all steps ahead and outperforms the random walk model for 4 and 8 steps ahead. Of course, also in this case it is reasonable to expect that the predictive power decreases as we forecast longer into the future. In the following section, we discuss our results in more details.

C. Discussion

Across the five independently developed, maintained, and managed open source projects, a single first-order moving average model with one order of non-seasonal differencing and constant term ARIMA(0,1,1) was shown to fit and forecast the pattern of weekly TD density evolution. The comparison of ARIMA(0,1,1) with the random walk indicates that a more complex model performs better when the forecast horizon is from 4 to 8 weeks. ARIMA model has shown better fitness statistics and higher predictive power compared to the random walk, which leads to the conclusion that the TD patterns can be modeled adequately by time series techniques.

However, when trying to forecast for 12 weeks ahead, the random walk model performs equally or even better in almost all of the cases. This is a reasonable finding as trying to forecast longer into the future, increases the uncertainty of the predictions. The only exception is the Jenkins case, where our model performs better. This finding deserves more investigation and will be the subject of future work.

Furthermore, the fact that the same ARIMA(0,1,1) model fits across the five time series could possibly point out that the pattern of TD is persistent across the five studied projects. This is a very interesting finding indicating that we can sacrifice the benefits of fine-tuning the prediction models to each of the products in favor of the simplicity of always applying the same model. However, the validity of this model assumes that no dramatic changes are made into the evolution process of the projects. In any case, the applicability of this model needs to be investigated on more projects in order to further support this conclusion.

More generally, we conclude that time series analysis is a useful technique for analyzing the evolution of the TD density over a relatively long period. However, these models build on the assumption that reliable historic data are available and that the data is collected in constant time intervals (e.g. daily, weekly, or monthly frequency). Generally, this is hardly the case as sufficient historic data are usually hard to acquire. Moreover, although time series models can yield satisfactorily accuracy, they demand a relatively long history of past data, require frequent re-training on new data, and are difficult to tune properly.

V. Conclusions and Future Work

This research work examines the usage of time series models as a valid and accurate approach to forecasting TD in long-lived, open-source, software projects. To the best of our knowledge, this is the first time that time series forecasting, and more specifically ARIMA methodology has been employed for this purpose.

For the purpose of our study we obtained 3 years of TD evolution history, computed from the static code analysis of five independently developed, maintained, and managed open-source software systems. Based on this data, we observed that a single time series model, ARIMA(0,1,1), can accurately predict the pattern of software evolution TD for each of five projects. The results shown indicate that ARIMA time series models outperform purely random processes and random walks up to 8 weeks into the future.

However, we have observed that predictive power decreases considerably for longer forecasting horizons. In addition, the fact that sufficient historic data are rarely available and hard to acquire, leads us to the conclusion that we should also consider other forecasting techniques. Machine learning algorithms, compared to classical methods for time series forecasting, include the ability to handle irrelevant features, as well as to support complex relationships and tolerance to noise between variables. As a future work, the suitability of popular methods such as Causal or Associative models as well as Machine Learning models, such as Support Vector Regression, Regression Trees, or ANNs for forecasting the evolution of TD especially in the case of longer forecasting horizons will be examined.

Acknowledgment

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through EXA2PRO project under Grant Agreement No. 801015.

References

[1] W. Cunningham, "The WyCash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, and others, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010.

[3] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," 2015.

[4] G. Digkas, M. Lungu, A. Chatzigeorgiou, and P. Avgeriou, "The evolution of technical debt in the apache ecosystem," in *European Conference on Software Architecture*. Springer, 2017, pp. 51–66.

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

[6] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, 2016.

[7] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17.

[8] T. Chaikalis and A. Chatzigeorgiou, "Forecasting java software evolution trends employing network models," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 582–602, 2015.

[9] D. Tsoukalas, M. Siavvas, M. Jankovic, D. Kehagias, A. Chatzigeorgiou, and D. Tzovaras, "Methods and Tools for TD Estimation and Forecasting: A State-of-the-art Survey," in *International Conference on Intelligent Systems (IS 2018)*. IEEE, 2018.

[10] H. S. Yazdi, M. Mirbolouki, P. Pietsch, T. Kehrer, and U. Kelter, "Analysis and prediction of design model evolution using time series," in *International Conference on Advanced Information Systems Eng.*, 2014.

[11] U. Raja, D. P. Hale, and J. E. Hale, "Modeling software evolution defects: a time series approach," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 1, pp. 49–71, 2009.

[12] M. Goulão, N. Fonte, M. Wermelinger, and F. B. e Abreu, "Software evolution prediction using seasonal time analysis: a comparative study," in *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 2012, pp. 213–222.

[13] B. Kenmei, G. Antoniol, and M. Di Penta, "Trend analysis and issue prediction in large-scale open source systems," in *2008 12th European Conference on Software Maintenance and Reengineering*, 2008.

[14] I. Gondra, "Applying machine learning to software fault-proneness prediction," *Journal of Systems and Software*, 2008.

[15] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 452–461.

[16] T. M. Khoshgoftaar, E. B. Allen, and J. Deng, "Using regression trees to classify fault-prone software modules," *IEEE Trans. on Reliab.*, 2002.

[17] H. C. Gall and M. Lanza, "Software evolution: analysis and visualization," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 1055–1056.

[18] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[19] J.-L. Letouzey and M. Ilkiewicz, "Managing technical debt with the sqale method," *IEEE software*, vol. 29, no. 6, pp. 44–51, 2012.

[20] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[21] R. McCleary and R. Hay, *Applied time series analysis for the social sciences*. Sage Publications, 1980.

[22] G. M. Ljung and G. E. Box, "On a measure of lack of fit in time series models," *Biometrika*, vol. 65, no. 2, pp. 297–303, 1978.

[23] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected papers of hirotugu akaike*, 1998.

[24] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.