Evaluating Time-Dependent Methods and Seasonal Effects in Code Technical Debt Prediction

Mikel Robredo[a], Nyyti Saarimäki[b], Matteo Esposito[a], Davide Taibi[a], Rafael Peñaloza[c], Valentina Lenarduzzi[a]

[a] *University of Oulu, Finland*
[b] *University of Luxembourg, Luxembourg*
[c] *University of Milano-Bicocca, Italy*

Abstract

Background. Code Technical Debt (Code TD) prediction has gained significant attention in recent software engineering research. However, no standardized approach to Code TD prediction fully captures the factors influencing its evolution.
Objective. Our study aims to assess the impact of time-dependent models and seasonal effects on Code TD prediction. It evaluates such models against widely used Machine Learning models also considering the influence of seasonality on prediction performance.
Methods. We trained 11 prediction models with 31 Java open-source projects. To assess their performance, we predicted future observations of the SQALE index. To evaluate the practical usability of our TD forecasting model and their impact on practitioners, we surveyed 23 software engineering professionals.
Results. Our study confirms the benefits of time-dependent techniques, with the ARIMAX model outperforming the others. Seasonal effects improved predictive performance, though the impact remained modest. ARIMAX/SARIMAX models demonstrated to provide well-balanced long-term forecasts. The survey highlighted strong industry interest in short- to medium-term TD forecasts.
Conclusions. Our findings support using techniques that capture time dependence in historical software metric data, particularly for Code TD. Effectively addressing this evidence requires adopting methods that account for temporal patterns.

Keywords: Technical Debt, Software Quality Mining Software Repositories, Empirical Software Engineering, Time Series Analysis

1. Introduction

In Software Engineering (SE), and specially in Mining Software Repository (MSR) studies, researchers often investigate the relationships among different variables collected from the history of software projects. As an example, researchers have investigated the correlations between two variables, such as code smells [1, 2, 3], their trend over time [4], and the impact of different qualities of software [5, 6, 7]. However, many of these studies have omitted the hidden potential impact on the temporal dependency of the variables analyzed and the possible threats related to statistical techniques not designed for analyzing temporally dependent data [8]. As an example, the introduction of a technical issue in a commit heavily depends on the code that was present in the repository *before* said commit. However, most studies have not considered this aspect, mainly due to a lack of clear guidelines on the statistical techniques used in this context. Saarimäki et al. [8] highlighted three main issues in previous studies: 1) discarding the temporal nature of the commits, 2) assuming independence of data, and 3) mixing projects of different sizes, where big projects overwhelm small ones. To overcome these issues, they proposed to analyze the dependent data in the SE field, considering the dependency through the time effect [8]. Within the provided study context, we use the code technical debt (code TD) prediction analysis to evaluate the impact of the time dependence factor. Code TD is an essential metric in software projects as it measures professionals' efforts to clean the code. Therefore, code TD denotes the time dependence on past mistakes. Consequently, data analysis techniques that assume this dependence should be considered.

In this direction, the research community traditionally uses machine learning (ML) models [9, 10]. Similarly, we identified two relevant works that applied Time Series Analysis (TSA) to predict code TD [11, 12] comparing univariate and multivariate TSA models. The results obtained were promising in terms of prediction performance. Hence, our objective is to analyze the robustness of TSA techniques predicting time-dependent variables such as code TD by comparing them with traditional ML methods. Therefore, we are striving to identify existing potential variables that can help explain the behavior of code TD.

Email addresses: mikel.robredomanero@oulu.fi (Mikel Robredo), nyyti.saarimaki@uni.lu (Nyyti Saarimäki), matteo.esposito@oulu.fi (Matteo Esposito), davide.taibi@oulu.fi (Davide Taibi), rafael.penaloza@unimib.it (Rafael Peñaloza), valentina.lenarduzzi@oulu.fi (Valentina Lenarduzzi)

We designed and conducted an empirical study to evaluate the code TD prediction performance of different TSA models and a set of time-agnostic ML models commonly used in the literature, which do not consider the data's temporal factor. Furthermore, we assessed the impact of the seasonality factor on the observed data. For that, we adopted the seasonally adjusted version of the TSA models already used in the SE literature and measured their performance to quantify the impact of addressing the seasonality factor. As the potential dependent variable adopted for the description of the code TD, we considered the SQALE index metric, computed by SonarQube (SQ) to measure code TD. Specifically, we adopted two different multivariate applications of the *Autoregressive Integrated Moving Average* (ARIMA) model [13] as well as their modified counterparts, which control the seasonality component of the data, and seven ML models to perform the confronted evaluation (four linear and three nonlinear). Furthermore, we implemented a backward variable selection approach [14, 15] to provide models with the best combination of variables to help improve their efficiency. The best resulting multivariate TSA models were afterwards used to explore their accuracy when performing long-term time-series forecasting, a facet of interest in software maintenance activities [10].

Finally, to evaluate the practical usability of our TD forecasting model and their impact on practitioners, we surveyed 23 software engineering professionals. The survey was used to determine their willingness to use the model, the usability of the model in real-world TD management, and the predicted window of choice for decision making. The survey asked questions about the professional experience of the respondents, their knowledge of TD and whether they manage TD issues in real life. In addition, the participants provided information on the best prediction horizons, trading off precision for predictions with practicability. The study supports the notion that practitioners desire high-precision predictions in short to medium-term horizons, which is consistent with agile software development cycles and allows for the use of TD predictions within industry pipelines.

Our results revealed a robust superiority obtained by the multivariate TSA models and demonstrated a slight improvement in predictive performance when adjusting the models to capture the seasonality effect. Basically, *Seasonally adjusted Autoregressive Integrated Moving Average* with extra regressors (SARIMAX) [16] and its non-adjusted counterpart ARIMAX [17] were the best prediction models among the methods considered in this study. These both methods implement a multivariate variant of the commonly known ARIMA model. However, the ARIMAX model outperforms the other ML algorithms in predictive performance [17]. Long-term forecast performance showed increased MAPE errors over time but remained balanced across projects. The MAPE error of the SARIMAX model increased from 1.08% at 6 months to 4.19% at 36 months, while ARIMAX remained at 3.15% after 12 biweekly periods and 5.97% after 72 periods. The variance remained low, with stable trends in descriptive statistics. Finally, our survey reveals good industry confidence in our tool and its effectiveness for short- to medium-term forecasting windows, which aligns closely with industry needs.

The main contributions of this paper are:

- An improvement in current knowledge of the prediction of the code TD with an emphasis on investigating the importance of including time-dependent factors.
- An additional contribution to the scientific community emphasizing the potential benefits of considering a monitored seasonal code TD measurement for subsequent improvement on the control of code TD in software projects.
- A large-scale comparison among the time-dependent and ML prediction models already adopted in the literature.
- A demonstration of the promising capabilities of classic, yet efficient multivariate TSA forecasting models on long-term code TD forecasting, aiming to enable system engineers and project managers to perform long-term effective software maintenance.
- An industrial survey with 23 experts confirming strong industry interest in short- to medium-term TD forecasts, aligning with agile workflows.

Paper structure. Section 2 describes the background on which our paper is based. In Section 3, we present the empirical study design. Section 4 presents the results obtained, and Section 5 shows the derived discussion based on them. Section 6 describes the identified threats to the validity of our study. Section 7 presents related work and Section 8 divulges the conclusions of this study and outlines the potential future work.

2. Background

SQ is one of the most commonly adopted static analysis tools both in academia [18, 19] and in industry [20]. We aim to help developers create more clean, secure, maintainable, readable, and modular code. SQ evaluates the code by testing it against a predefined set of rules. The rules are considered *coding standards*, and breaking them is deemed undesirable. Therefore, each time a code violates a rule, the SQ presents an issue to the developers. Practitioners can use SQ on its official "as a Service" flavor hosted on the sonarcloud.io website or the on-demand version to run on a private server. The tool supports 29 programming languages: Java, Python, C++, and JavaScript.[1] It performs various calculations, including measuring metrics like lines of code and code complexity. It also establishes specific thresholds known as "quality gates" for each metric and rule.

In this study, the selected dataset (Section 3.2) analyzed projects with SQ version 7.5, which includes three rule categories:

[1] https://docs.SQ.org/9.6/analyzing-source-code/languages/overview/

- *Reliability rules* named as bugs that create an issue that "represents something wrong in the code" and that "will soon be reflected in a bug in the code."
- *Maintainability rules* named as code smells that decrease code readability and modifiability. It is important to note that the term "code smells" adopted in SQ contains only a subset of the well-known code smells defined by Fowler et al. [21].
- *Security rules* named as Vulnerability, creates a problem that impacts the application's security.

To quantify and therefore measure the existing technical debt in software projects, SQ leverages a series of metrics, mostly quantitative translations of code analysis metrics. In the version considered for this study, SQ computed three types of TD, *reliability remediation effort* and *security remediation effort*, which consisted on the time to fix all the open issues classified as bugs and vulnerabilities, and the *technical debt ratio*, which SQ named as *SQALE index* resembling the naming of SQALE methodology [22] used in older versions, and consisting on the time to fix all open issues classified as *code smell*.

3. The Empirical Study Design

We now describe our study, reporting the goal and research questions, the context, the collection and analysis of data. We designed our experimental study based on the guidelines defined by Wohlin et al. [23].

3.1. Goal and Research Questions

We formalized the goal of this study according to the GQM approach [24] as follows:

Analyze time-dependence and seasonality effect factors,
for the purpose of comparing code TD prediction,
with respect to performance,
from the point of view of developers,
in the context of open source software.

Based on this goal, our formulated Research Questions (**RQs**) are presented next.

> **RQ_1.** *Which multivariate time-dependent method predicts code TD with the highest performance?*

$\mathbf{RQ_1}$ compares two different multivariate TSA approaches already implemented in the SE literature to determine which approach can predict code TD better. Evaluating the prediction performance of the latest models that control the time dependence factor within the data enables us to determine which of these models should be considered by practitioners when deliberating which prediction techniques to use when predicting code TD. To address RQ_1, we collected as code TD the SQALE index metric computed by SQ (as explained in Section 2), and Code Smell rule violations generated by SQ as potential explanations of the SQALE index's behavior.

The first approach, known as ARIMA + LM, combines multiple ARIMA models and linear regression models (LM) [12]. This approach utilizes a univariate ARIMA model for the prediction of each independent variable. Using predicted values within a regression model, it predicts future values of the dependent variable. The second approach, known as ARIMAX, consists of building a multivariate ARIMA model to predict the values of the dependent variable [11]. In the ARIMAX model, the history of the dependent variable and the past values of the considered independent variables are used as features to train the model and provide future predictions of the dependent variable.

Our prior assumption is that the ARIMAX model performs better than the ARIMA + LM models in explaining the future evolution of the dependent variable. We assume that combining the independent variables with the past values of the dependent variable has more information than separately predicting the values of the independent variables to predict the dependent variable through a regression model.

The purpose of this study is to establish the applicability of the most efficient time-dependent techniques for code TD prediction and evaluate how the adopted techniques perform against other prediction techniques commonly used in the SE literature. These techniques are mostly ML-based algorithms that do not consider the temporal factor when providing future predictions. Therefore, we formulate our $\mathbf{RQ_2}$.

> **RQ_2.** *How accurate is the prediction performance of a multivariate time-dependent approach for code TD prediction compared to that of an ML algorithm?*

To assess the applicability of a time-dependent approach for code TD prediction ($\mathbf{RQ_2}$), in our case multivariate TSA, we need to compare its performance with other approaches that do not consider the data's time dependency. In this study, the comparison is made with several ML models.

Similarly, as for $\mathbf{RQ_1}$, our prior assumption is that the prediction performance of multivariate TSA methods for code TD prediction overcomes that of ML models. Moreover, TSA methods are based on temporally ordered serialized data, helping the models capture potential temporal factors, for example, the seasonality of the observation [25]. Models can be adjusted to these factors to observe their impact on the predicted results. Therefore, only comparing already used TSA methods is not sufficient, and we formulate a third research question.

> **RQ_3.** *Does addressing seasonality improve the prediction performance of multivariate time-dependent approaches for code TD prediction?*

As explained, $\mathbf{RQ_3}$ expands $\mathbf{RQ_1}$. For that, given the modification of the models to handle seasonality patterns, $\mathbf{RQ_3}$ compares the new models with the former in terms of the prediction of the code TD and therefore evaluates the impact of seasonality in the prediction performance.

Our prior assumption is that SARIMAX, SARIMA + LM approaches and seasonally adjusted models of the previously defined TSA model counterparts perform better than their former model format, which did not handle seasonality patterns. In addition, we take advantage of the comparison proposed in RQ_1. Thus we assume that using the multivariate SARIMAX model is more informative than the SARIMA+LM model, given the extension of both TSA models to capture the impact of the seasonality effects.

Similarly, evaluating the seasonality adjustment in prediction performance compared to the selected ML algorithms becomes paramount to contributing to the SE community through this study. Therefore, we ask a fourth question.

RQ_4. *How accurate is the prediction performance of a seasonally adjusted multivariate time-dependent approach for code TD prediction compared to that of an ML algorithm?*

Our prior assumption is that the additional modification implemented in the multivariate TSA models regarding seasonality control enables their code TD prediction performance to be better than that of ML models.

For all RQs described so far, we evaluated the models using three key **performance metrics**: Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) (see Section 3.6.5 for a detailed description of these metrics).

RQ_5. *How accurate are multivariate time-dependent methods on long-term forecasting?*

A challenge that emerges from the concept of code TD prediction is the need to make multi-step forecasts, that is, forecasts for more than one time-step into the future. Providing reasonable long-term forecasts would allow system engineers and project managers to perform long-term effective software maintenance. While single-step predictions can be useful for short-term planning, multi-step forecasting offers more practical insights for software maintenance and allows project managers to make strategic decisions for the future. Therefore, through the defined $\mathbf{RQ}_5$ we will investigate the performance of the resulting best TSA models when performing multi-step forecasting to enable practitioners observe the long-term impact of their current project maintenance level within the context of code TD.

RQ_6. *How valuable do software practitioners perceive a Technical Debt forecasting to be?*

- **$RQ_{6.1}$.** *What forecasting window combinations do practitioners consider acceptable and useful for Technical Debt prediction?*
- **$RQ_{6.2}$.** *What is the preferred scope of Technical Debt prediction among practitioners?*

While many techniques exist to measure and manage code TD retrospectively, forecasting its evolution could allow practitioners to take proactive measures. Understanding how practitioners perceive code TD forecasting is crucial to determining whether such predictive models can influence decision-making processes in software projects or are just academic interests ($\mathbf{RQ}_6$). If practitioners do not see substantial value, efforts in this direction might need re-evaluation or refinement to align with industry needs.

To further explore this aspect, we investigate the specific forecasting configurations that practitioners deem useful. To effectively forecast the evolution of code TD ($\mathbf{RQ}_{6.1}$), it is crucial to determine the appropriate time horizon for predictions. Practitioners may prefer short-term forecasts over weeks or months to aid in immediate refactoring efforts. In contrast, long-term forecasts, spanning years, could be more useful for strategic planning. Knowing which time frames practitioners consider valid and useful ensures that code TD prediction models meet actual industry needs, thus facilitating their practical application. Moreover, selecting suitable forecast windows is just one part of the equation. Defining the preferred scope of code TD predictions to customize these forecasts properly is equally important.

Code TD can appear at different levels, from specific code components to software projects. Some practitioners focus on predicting TD at the module or class level to aid in detailed refactoring efforts ($\mathbf{RQ}_{6.2}$). In contrast, others prefer predictions at the system-wide level to help with architectural decisions. Understanding the preferred scope ensures that code TD forecasting tools provide valuable insights at the appropriate level of abstraction, making them more useful in practitioners' daily workflows. Our survey offers a comprehensive view on creating Technical Debt forecasting methods that are practical and actionable for software professionals.

For $\mathbf{RQ}_5$ and $\mathbf{RQ}_6$, we evaluated the models using only MAPE since it is one of the most intuitive metrics for practitioners to discuss [26, 27].

3.2. Context

In the context of our study, we consider the Technical Debt Dataset [28] version 2.0 adopted by Saarimäki et al. [8]. The data set contains 31 Java projects from the Apache Software Foundation (ASF) repository.[2] The projects in the data set were selected based on "criterion sampling" [29], requiring all of the following criteria: developed in Java, older than 3 years, more than 500 commits and 100 classes, and the usage of an issue tracking system with at least 100 issues reported. The projects were selected, maximizing their diversity and representation by considering a comparable number of projects on the age, size, and domain of the project. The projects can be considered mature due to the strict review and inclusion process required by the ASF. In addition, the included projects regularly review their code and follow a

[2] http://apache.org

strict quality process.[3] Full details of the data can be found in the online repository of the data set.[4] We adopted this data set since all its projects were already analyzed with SQ and it contains the data related to the SQALE index.

3.3. Variables

In this section, we describe the dependent and independent variables used in the models adopted for the prediction of code TD.

- **Dependent variable**: We consider the metric SQALE index computed by SQ. *The SQALE index* denotes the efforts to fix all the Code Smell rules that are violated in the code and is measured in minutes. SQ denotes TD using the total value of the SQALE index. The metric is continuous and, therefore, is suitable for the TSA models adopted in this study. In addition, it provides a clear definition of code TD to answer the objective of our study.
- **Independent variables:** We consider the Code Smell rule violations detected by SQ. *Code smell rule violations* describe the number of violations for each SQ rule of code smell type. The TD dataset has issues from 205 different rules of type code smell, and each rule is included as a separate independent variable in the study. The nature of the collected issues is discrete as it is based on the count of issues detected in each SQ analysis for each code smell type.

3.4. Data Collection

In this study, we were interested in studying the impact of temporal factors on the selected dependent variable, the SQALE index. As described in Section 2, code smell rule violations are directly related to the SQALE index, which is the remediation cost. Hence, we decided to collect the 205 code smell rule violations existing in the adopted data set as the initial set of independent variables for our study.

3.4.1. Data preprocessing: Feature selection

The data collected contains 205 potential independent variables. Such a large number could have a negative impact on the models due to the so-called "curse of dimensionality:" including a large number of independent variables may drastically reduce the predictive power of a model. To identify a set of the most explicative independent variables for our dependent variable, we undertook a feature selection process based on well-known feature selection techniques. The aim was to categorize the potential independent variables according to their importance. These techniques are briefly described next.

- **Variance Thresholding**: Method that utilizes the variance or spread of a variable in a data set to select model-independent variables. In summary, this method calculates the variance of all potential independent variables in the data sets. Removes those with low variance as they are less informative and do not have much predictive power [30].
- **Zero percentage method**: This method eliminates predominantly empty independent variables, i.e. the percentage of emptiness is higher than a predefined threshold and is therefore poorly informative. It is used with count data [31].
- **Feature importance**: This method determines the importance of each variable in a data set using a Random Forest (RF) algorithm. RF is an ensemble learning technique consisting of multiple decision trees, in this case, used for prediction. The measure of importance is based on the reduction in node impurities due to the splitting of the variable. Averaged across all trees, the measure of the impurity of the node of all independent variables is calculated by the residual sum of squares. Thus, this method excludes variables with low feature importance [32].
- **Correlation analysis**: The method evaluates the degree of linear association between the independent and dependent variables. Given the high skewness illustrated in Figure 1, and therefore lack of normality in our variables [33], we measure the correlational association through the non-parametric Spearman's ρ rank correlation coefficient [34] for each independent variable. Thus, variables with a poor informative contribution for predictive modeling are discarded [35].

One could expect that each of the selected feature selection techniques would provide different results on the initial collected issue-type variables. Therefore, we ranked the independent variables in terms of their resulting importance score for each of the importance techniques of the characteristics, resulting in four different rankings. We used the *Interquartile Range* (IQR) method to select the upper quartile in the ranking distribution based on the importance scores. In a nutshell, given a set of observations, the IQR method builds the underlying distribution from the observations. Select the subset of the different quartiles in the distribution based on the objective of the analysis [36]. In our scenario, we were interested in selecting the most informative independent variables of our set of variables; therefore, we selected the quartile with the variables denoting the highest feature importance score. Additionally, for the sake of consistency in the final set of independent variables, we selected the subset of variables that showed importance in each of the four fetched quartiles. Consequently, the feature selection process resulted in a subset of 15 independent variables that were specifically selected to predict the SQALE index (dependent variable). Table 1 and Figure 1 provide a graphical and quantitative representation of the independent variables.

3.4.2. Data preprocessing: Transforming raw data into time series data

The time-dependent approaches used in this paper require the use of periodically measured time-serialized data. Given

[3] https://incubator.apache.org/policy/process.html
[4] https://github.com/clowee/The-Technical-Debt-Dataset

that commits are not periodic, we relied on two criteria to generate the time series. First, we focused on constructing a sufficiently long time series for each project (study subjects) to provide the models with sufficient data to perform the prediction. Second, we investigated different periodicity levels for the observations in the time series. In this way, we could observe the temporal nature of the data and find the most suitable time frame between observations.

Based on the described criteria, we generated time series data at biweekly and monthly periodicity levels. The number of data points for each project on each periodicity level is presented in Table B.19. TSA models require at least 24 observations to detect existing time-dependent factors such as the temporal trend, and consequently to be trained [25]. Therefore, increasing the size of the time window would reduce the number of observations for the collected projects and would require excluding the projects from the study.

Version control data are stochastically ordered by nature,[5] which means that some time periods (of weeks or months in our study) might not have commits despite being otherwise an active project. Consequently, serializing such data can result in missing data for some time points. Linear interpolation was used to fill in time points that lacked observations in the generated time series. This method presupposes a linear correlation between neighboring data points and calculates the values of the missing data points using the known values surrounding them [37].

Consequently, the analysis was performed using two different samples of the data: *monthly* time series data and *biweekly* time series data. As a potential threat to the validity of this study, we address the interpolation of artificially generated observations in Section 6. Similarly, we provide suggestions for practitioners on the remediation approach to avoid missing data in Section 4.1 to help practitioners generate higher quality data.

3.5. Collecting the practitioners' forecasting perceived valuable

TD is a persistent challenge in software development, affecting maintainability, quality, and long-term sustainability. Although retrospective code TD analysis is widely studied, proactive forecasting remains an emerging area with limited understanding of its perceived value in industry. To bridge this gap, we designed this survey to capture the perspectives of software professionals on the usefulness, preferred forecasting windows, and optimal scope of Code TD predictions. Our target audience consists of software engineers, architects, and technical leaders involved in technical debt management. Our objective is to evaluate whether our forecasting model is perceived as beneficial to the industry. Specifically, we want to determine the most advantageous forecasting time windows that add value to the software development lifecycle (SDLC) within an industrial setting.

3.5.1. Population Description

Table 3 and Table 4 present the professional experience of our the 23 interviewed experts. The surveyed population is predominantly composed of software professionals, particularly Software Developers, and is heavily concentrated in Europe. The respondents largely work in established firms within the ICT, IT Consulting, and Software Engineering sectors. They possess a range of professional experience, with a significant number employing Agile methodologies in their development practice and familiarity with code TD. Hence, our population appears to be a fair sample of the intended audience. We collected the answers anonymously and provided all participants with information regarding the GDPR act for data collection and protection. We also thoroughly followed the ACM publications policy on research involving human participants and subjects [38].

3.5.2. Questionnaire

Table 2 presents the questions used to answer RQ_6. Due to space constraints, we provide the complete questionnaire in the replication package. It should be noted that, before submitting the questionnaire, we have performed a pilot questionnaire to derive the answers to the predefined closed-ended questions (C) predefined answers. Moreover, for most closed-ended questions, we provided an open-ended version (O) to account for possible missing categories or to allow for an in-depth explanation of the selection. We interviewed experts involved in our previous research and, leveraging domain knowledge, reached a consensus on the predefined answer to closed-ended questions. We did not ask the pilot's experts to participate in the final questionnaire to avoid biases.

According to our guidelines [39, 40, 24, 41], we asked participants to answer demographic questions to obtain information about the population under examination. Thus, Q_1 to Q_{11} refer to the personal background and professional activities of the interviewee. We asked participants to respond with predefined choices to facilitate data analysis using domain knowledge [42, 27].

For $RQ_{6.1}$, we asked them for their perceived benefit in predicting the code TD. More specifically, we provide a Likert scale question (L), Q_{12} surveying how much would they value a tool for code TD prediction. A Likert scale is a psychometric scale commonly used in questionnaires to gauge respondents' attitudes, opinions, or perceptions on a particular topic [43]. It typically consists of a series of statements in which respondents indicate their level of agreement or disagreement on a symmetric agree-disagree scale, usually ranging from "strongly agree" to "strongly disagree." This method allows the quantitative measurement of people's attitudes or feelings toward a subject. The values for each Likert scale question are available in the replication package. Furthermore, we asked the interviewee to justify the rationale for their Likert choice in Q_{13}. In addition, we also provide two closed-ended questions, Q_{14} and Q_{16} to choose between

[5] https://www.git-scm.com/docs/git-commit

Table 1: Descriptive statistics of the SQALE index and the resulting independent variables from the data preprocessing stage.

Variable	Mean	Standard deviation	Min	Lower quartile	Median value	Upper quartile	Max	Skewness
SQALE index	137,726	169,927	0	24,378	72,981	182,132	701,914	1.87
S1213	740	1,350	0	38	152	579	6,106	2.22
RedundantThrows DeclarationCheck	345	509	0	15	113	340	2,200	1.91
S00117	1,082	2,380	0	7	55	383	10,395	2.3
S00122	402	658	0	1	67	585	4,119	2.55
S1488	88	163	0	4	20	81	1,196	2.85
S1905	74	148	0	1	12	44	575	2.26
UselessImportCheck	630	1,402	0	7	42	151	7,928	2.58
DuplicatedBlocks	333	510	0	40	142	305	3,099	2.52
S1226	212	293	0	18	62	330	1,479	1.91
S00112	473	848	0	58	183	374	4,987	3.01
S1155	86	169	0	2	12	67	933	2.49
S00108	97	128	0	4	62	127	739	1.95
S1151	472	1,007	0	2	26	200	3,699	2.25
S1132	289	481	0	20	57	436	3,587	2.94
S1481	132	274	0	5	18	71	1,600	2.75

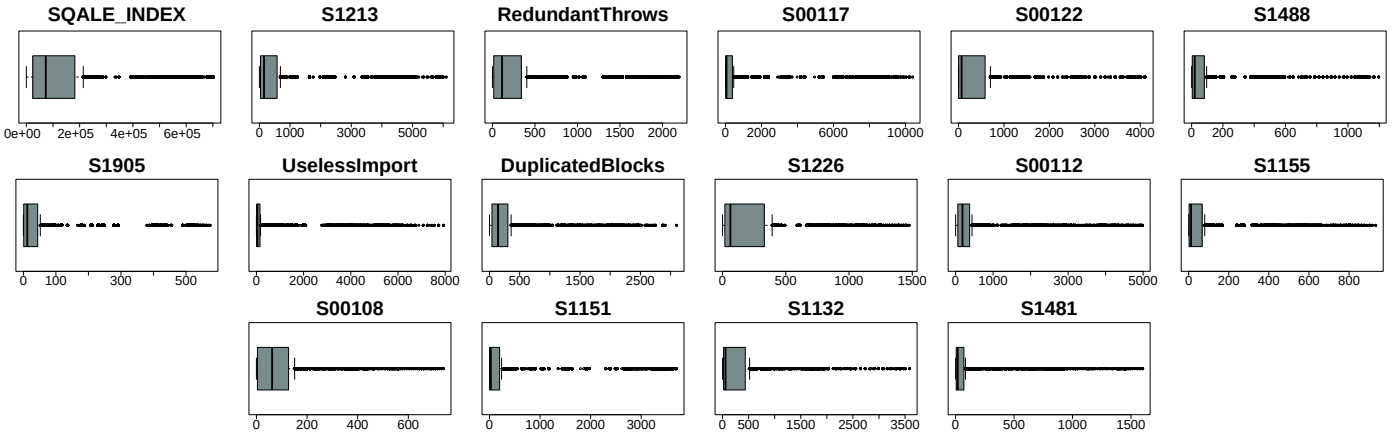


Figure 1: Boxplots for the considered model variables.

pairs of "<*time windows, MAPE*>." To allow the practitioner to grasp our ability to predict Code TD, we decided, based on RQ_5 findings, to select a maximum time frame of 36 months, that is, 3 years. We divided such time frames into biweekly prediction time windows Q_{13} and monthly time window Q_{15} by pairing the time window with its computed MAPE (e.g., 2 weeks - 1.44 %). Since we were interested in understanding the rationale for the choice, we also asked the practitioners to further disclose their motivation for the chosen intervals in the two connected open-ended questions Q_{15} and Q_{17}.

Finally, for $RQ_{6.2}$, we ask whether the interviewee prefers to predict the code TD weekly or monthly Q_{18} based on previous comparisons of time windows and MAPE, and to justify their choice in Q_{19}.

3.6. Data Analysis

In this section, we focus on the designed data analysis based on the definition of the selected models, and we explain the comparison of the seasonally adjusted models against their root model counterparts, their model parameter tuning structure, and the ML models adopted. The selected time-dependent models are based on *multivariate* TSA modeling, a field of statistical modeling commonly used in diverse fields such as econometrics and weather forecasting, for instance [44]. The multivariate nature of the models allows us to consider information about additional independent variables that can assist the model in the prediction process. Figure 2 shows the visual representation of the data analysis process performed in this study.

3.6.1. Setting the stage for seasonally unadjusted TSA model prediction (RQ_1 - RQ_5)

We examine two distinct methods to predict code TD utilizing TSA models, ARIMAX and ARIMA+LM. As version control data is inherently *non-stationary*, i.e. denote a trend across time, the two methods originate from the *Box-Jenkins*

Table 2: Survey: Questions and research questions (RQ_6)
Legend: **C** - Closed, **C*** - Closed with "Other" option, **L** - Linkert, **O** - Open Ended

RQ		Question (Q)	Type
Profiling	Q_1	What is your job title or role?	O
	Q_2	What sector does your organization belong to?	O
	Q_3	What is the size of your organization?	C
	Q_4	Where is your organization located?	C
	Q_5	How many years of experience do you have?	C
	Q_6	What types of projects do you usually work on?	C*
	Q_7	Do you use any agile development practices?	C
	Q_8	Are you familiar with the concept of technical debt?	C
	Q_9	If yes, what agile practice do you employ?	O
	Q_{10}	How often do you address Technical Debt-related issues?	O
	Q_{11}	What are the types of Technical Debt issues you address most frequently?	O
$RQ_{6.1}$	Q_{12}	How much would you value a tool for Technical Debt forecasting?	L
	Q_{13}	Please justify your choice in the previous question.	O
	Q_{14}	Which forecasting window combinations are acceptable and useful for your workflow?	C
	Q_{15}	Please justify your choice in the previous question.	O
	Q_{16}	Among the following options, what forecasting window combinations may you find acceptable and useful for your workflow?	C
	Q_{17}	Please justify why you chose the previous options.	O
$RQ_{6.2}$	Q_{18}	If given the choice, would you rather choose to predict Technical Debt: Weekly, By-Weekly, Monthly, Other?	C*
	Q_{19}	Please justify your choice in the previous question.	O

model. better known as *Auto-Regressive Integrated Moving Average* (ARIMA) model. It is worth noticing that a time series of data is *stationary* when its predictions do not depend on factors such as trend or seasonality existing in the real data [45]. The ARIMA model is a modified version of the *Autoregressive Moving Average* (ARMA) model used for modeling non-stationary time series [46]. Like the ARMA model, ARIMA determines the best autoregressive parameter for the variable to be predicted (AR step) and also identifies the optimal moving average parameter (MA) [44]. In addition, ARIMA executes the integration step (I), which determines the level of differencing required for the time series of the variable in question to achieve stationarity. The methods considered are the following.

- **ARIMAX**: The approach used in [11] was an ARIMA model with a set of independent variables included in the model, commonly referred to as ARIMAX. This model offers a further multivariate approach to the ARIMA model by introducing additional independent variables into the model and, therefore, helping to explain the evolution of the SQALE index. This enables future SQALE index values to be explained both by its past values and by those of the independent variables.
- **ARIMA + LM**: An ensemble technique previously adopted in [12] that comprises constructing a univariate ARIMA model for each independent variable in the model. Each ARIMA model considers the past values of their respective model variable as model parameters and therefore independently predicts the future values of each model variable. The additional step in this methodology relies on the performance of *linear regression* (LM) considering each of the new prediction lag values to predict the new values for the SQALE index variable.

3.6.2. Setting the stage for Seasonally Adjusted TSA model prediction ($RQ_{3,4}$)

In contrast with the previously presented ARIMAX and ARIMA+LM models, the seasonally adjusted models capture the impact of the seasonality pattern in the data while maintaining the model nature of the previously defined TSA model counterparts. Therefore, the analyzed seasonally adjusted methods are as follows.

- **SARIMAX**: The *Seasonally adjusted Auto-Regressive Integrated Moving Average* model stands as the seasonally adjusted extension of the ARIMAX model. Addresses the seasonality of the data within model training through model parameter tuning [13]. More information on the adjustment of the model parameters in the TSA model is provided in Section 3.6.3. Similarly, the SARIMAX model provides a multivariate approach compared to its univariate counterpart, SARIMA, by including independent variables in the model. This enables future SQALE index values to be explained both by its past values and by those of the included independent variables.

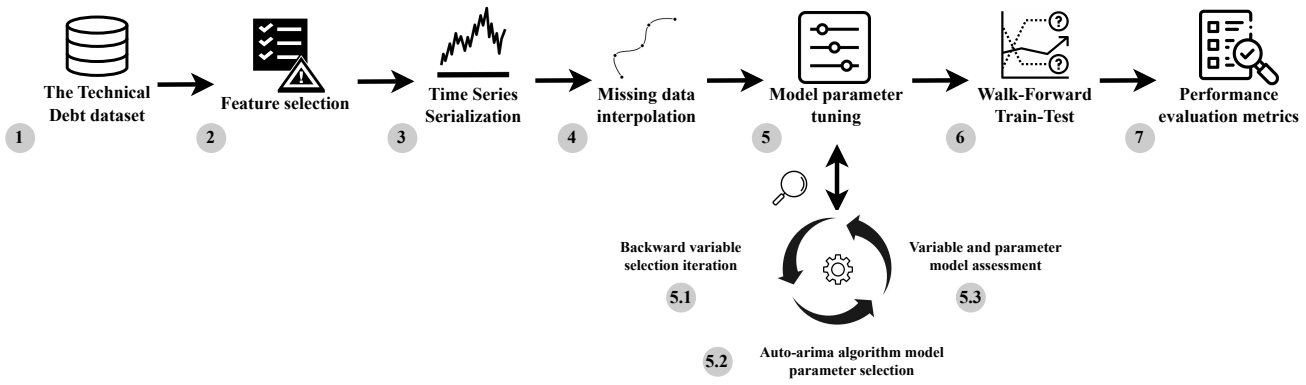


Figure 2: Data analysis process workflow diagram.

- **SARIMA + LM**: The approach is characterized by the combination of the univariate *seasonally adjusted integrated moving average* (SARIMA) model combined with a *linear regression* model (LM). Based on the seasonality adjustment performed in the approach used in [12], a SARIMA model is built for each of the independent variables considered in the study, and therefore each model independently predicts future values of a single variable accordingly. The additional step in this methodology relies on the implementation of *linear regression* (LM) considering each new prediction lag value of the independent variables to predict the new values for the SQALE index variable.

3.6.3. Setting the model parameter order in TSA models (RQ_1 - RQ_5)

Similarly to other families of predictive analysis models, for instance, Linear Models or Generalized Linear Models [33], TSA models are founded on a set of model parameters that are estimated after the model is trained. As mentioned earlier, in this study we adopt as prediction models derivations from the *Box-Jenkins* ARIMA model [13]. Models founded on the ARIMA model follow the same model parameter structure, commonly depicted as

$$(p, d, q)$$

where p stands for the number of steps that a model needs to go back to the history data to predict the next value, that is, the level of autoregression (AR), q denotes the number of moving averages required (MA) and captures the short-term random fluctuations in the data, and d defines the differencing level required to achieve stationarity in the trend observed within the data and therefore be able to make predictions [47]. Models must be adjusted to these factors to reach stationarity in trained data. We provide a detailed description of how parameter tuning is undergone with ARIMA-based models in the appendix.

Similarly, models that capture seasonal patterns, such as SARIMA, are presented with an extended set of parameters

$$(p, d, q)(P, D, Q, m).$$

When the model is adapted to capture the seasonal patterns of the data in models such as SARIMAX and SARIMA+LM, the parameters *P, D, Q* are also calculated. These capture the impact of seasonality on the model parameters explained above, which the model adjusts based on the specified m cycle level in the data. In our study, we consider *monthly* and *biweekly* cycles as previously defined. Thus, the adjusted models provide an additional layer in the model parameter tuning where the seasonality component is treated.

In this study, the model parameters presented are tuned when the TSA models are fitted with the training data. To determine the optimal combination of the model parameters, we conducted an iterative process in which we searched for the best combination of model parameters for each of the project's time series data. Thus, for different combinations of the values of the parameters p and q, we use the augmented Dickey-Fuller test [48] to determine the optimal value d of the diffencing model parameter. Similarly, when the seasonality effect is adjusted, the Canova-Hansen test [49] is used to determine the optimal order of seasonal differentiation D, along with different combinations of P and Q. To execute this iterative process, we adopt the *Auto-arima* algorithm [50, 51] which defines the potential range of parameter values based on the fitted data. We provide a detailed description of the algorithm functioning in the appendix.

Based on the temporal patterns of the SQALE index, the different combinations of models computed for the defined parameters are evaluated using the Akaike Information Criterion [52] (AIC) and the Bayesian Information Criterion [53] (BIC) following the latest in statistical modelling [33]. We provide a theoretical description of the model parameters presented in the appendix. Moreover, we provide the logic implemented in the shared online package (see Section 9).

3.6.4. Setting the stage for Machine Learning models (RQ_2 - RQ_4)

In RQ_2 and RQ_4, we compare the prediction performance of TSA approaches with that of ML algorithms. Therefore, we also investigate the ability of ML models to predict code TD by applying a collection of linear and nonlinear ML al-

Table 3: Interviewees' Professional Experience - Part 1

	Question	Response	%
Q_1	Job title	Software Architect	9
		Software Developer	30
		Software Engineer	4
Q_2	Organization sector	Cloud Architect	4
		ICT	35
		IT Consulting	30
		Software Engineering	26
		Software Development	4
Q_3	Organization size	Large Enterprise	65
		SME	17
		Small	13
		Start-up	4
Q_4	Organization location	Europe	87
		North America	4
		South America	9
Q_5	Years of experience	< 1	9
		1-3	26
		4-7	30
		8-10	13
		> 10	22
Q_6	Type of Project	AI/ML	29
		Web Development	29
		Cloud & DevOps	12
		IoT	8
		Embedded Systems	8
		Mobile Development	4
		Software Architecture & Quality	8

Table 4: Interviewees' Professional Experience - Part 2

	Question	Response	%
Q_7	Use of agile practices	No	26
		Yes	74
Q_8	Familiarity with TD	No	13
		Somewhat	17
		Yes	70
Q_9	Agile Practice Employed	Scrum (SCRUM, Lean + Scrum)	56.25
		CI/CD, Version Control, Test Automation	18.75
		Kanban	6.25
		MLOps	6.25
		Feature-Driven Development (FDD)	6.25
Q_{10}	Frequency addressing TD	Weekly	4
		Per Sprint	13
		Monthly	13
		Quarterly	4
		Project Start	4
		As Needed	9
		Occasionally	22
		Rarely	13
Q_{11}	TD type addressed	Refactoring (general & systematic)	26.09
		Testing Debt (test smells, lack of unit tests)	17.39
		Code smells	13.04
		Bug fixing (small, unforeseen, product bugs)	13.04
		Architectural Debt (smells, poor planning, scalability issues)	13.04

gorithms. To run the comparison in equal conditions with all the models considered in this study, the same time series data used to train and test the TSA methods are also used to train and test the selected ML algorithms. Further descriptions of the training and performance evaluation of the models studied can be found in Section 3.6.6.

The ML methods used for the comparison are selected to provide a wide predictive perspective, as each of them textualizes different aspects of the data. Likewise, the chosen ML algorithms have been extensively used in the recent literature for their ability to predict software quality characteristics, such as code TD [12, 9, 10, 54, 55].

The included ML models assuming ***linearity*** **(L)** in the data are the following.

- **Multiple Linear Regression (MLR)** is a powerful statistical model commonly used to analyze the relationship between a continuous dependent variable and a set of independent explanatory variables. Through this relationship, the model can provide insight into how changes in the independent variables affect the value of the response variable while maintaining the linearity assumption [56].
- **Stochastic Gradient Descent (SGD)** algorithm is based on an iterative decrease in prediction loss by changing the model parameters. Specifically, for each iteration, the SGD algorithm randomly selects a training set to bring randomness to the optimization problem, and the algorithm continues to optimize the result until it converges [57].
- **Lasso regression (L1)** is a linear regression technique specialized in high-dimensional explanatory variable sets. This technique performs an optimization problem where model-independent variables are penalized if they are not important to the model. Thus, through model simplification, L1 achieves linear regression with the most relevant explanatory variables [58].
- **Ridge regression (L2)** is another linear regression technique that, through a regularization term, avoids overfitting the model. The L2 technique brings a penalization for the given model parameters that denote extreme values, and thus stabilizes the model, which results in advantages for applications where the number of model-independent variables is high [59].

The selected ML models assuming ***nonlinearity*** **(NL)** are:

- **Support Vector Machine (SVM)** is a well-established ML algorithm used for classification and regression tasks. The regression model aims to find the most optimal hyperplane that considers the characteristics of the existing explanatory variables to explain the values of the output response variable. For this, the model performs a minimization process of the deviation between the real value and the predicted value [60]. We considered using the so-called Gaussian kernel or Radial Basis Function as the function to build the model to get the perspective of a non-linear approach.

- **Extreme Gradient Boost (XGB)** is an ML algorithm that is part of the family of gradient boost methods. Based on the ensemble learning methodology, the XGB algorithm employs multiple decision trees in a greedy format, where each of the subsequent trees corrects or refines the wrong results from the previous tree. Through this technique, the algorithm defines weights for the included independent variables that require a higher emphasis [61].

- **Random Forest regression (RF)** also belongs to the ensemble learning family and is built based on the ensemble of decision trees. Based on random sampling, the RF algorithm performs multiple decision trees and the output is the aggregated value from the results obtained in the trees performed [62].

3.6.5. Performance metrics (RQ_1 - RQ_5)

We used three performance metrics to compare and evaluate the prediction approaches and models.

The first performance metric is the *Mean Absolute Percentage Error (MAPE)*. MAPE is a statistical metric commonly used to measure prediction precision. Quantifies the magnitude of errors between the predicted values and the actual observed values by calculating the mean of the absolute value of the prediction error. Its formula is the following.

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \frac{| (Y_i - \hat{Y}_i)^2 |}{Y_i} \quad (1)$$

where n is the number of observations, Y_i is the actual observed value, and $\hat{Y}_i$ is the i-th predicted value. As shown by the formula, MAPE is represented as a percentage metric, where a smaller percentage of error means better predictive performance, and the opposite is true with a higher percentage.

MAPE has limitations when the actual observations are small or close to zero, which introduces bias into the model training. Therefore, we selected alternative accuracy measurement statistics to cover this issue. One of them, and the second metric to present is the *Mean Absolute Error (MAE)*, which is characterized by measuring the average magnitude from the absolute value of prediction errors. Its equation is

$$MAE = \frac{\sum_{i=1}^{n} | Y_i - \hat{Y}_i |}{n} \quad (2)$$

where the variables follow the same notation as the ones addressed for MAPE MAE expresses the prediction performance error in absolute value; therefore, a smaller result signifies better predictive performance, while higher results represent poorer performance.

We adopt as the third performance metric *root mean squared error* (RMSE) which captures the error value in the same value unit as the variable being predicted. Its formula is

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2} \quad (3)$$

where the variables follow the same notation as the ones explained previously. Although the definitions have value unit differences, a low error value indicates a high predictive performance in RMSE, and similarly, a high error value depicts a poor predictive performance. We provide further results on AIC and BIC model selection criteria in the online Appendix (see Section 9).

3.6.6. Performance evaluation (RQ_1 - RQ_5)

To evaluate the performance of the confronted models, we trained and tested the TSA and ML models on the two time-series datasets generated during the data serialization in Section 3.4.2. We adopted the Walk-Forward Train-Test validation technique [63] for the performance evaluation of all models considered in this study. This technique was chosen because it respects the temporal nature of the data and does not rely on randomness in the order of the observations. This helped to produce a fair comparison between the confronted methods, as especially ML models are not structurally designed to understand the temporal order existing in the fitted data.

The Walk-Forward Train-Test technique trains and tests a model greedily and iteratively. Each iteration trains a model, predicts the value of the next data point of the time series, and compares the prediction against the real value. The real value is consequently added to the train set, and the model is trained again with the newly extended train set data. This cycle continues until all test data points are predicted. In our study, the models start training with 80% of the data as training data and then predict and test every new data point. Figure 3 shows a graphical representation of the approach, where i denotes the new data point in each iteration. The described model training and testing phases performed in the performance evaluation stage of this study were implemented equally in each of the adopted models, with the same data testing set, for the sake of fairness in the predictive performance comparison.

3.6.7. Practitioners' forecasting perceived valuable (RQ_6)

This section presents the data analysis of our work. Our survey includes closed and open questions. Therefore, we select different analysis methods for the two types of survey output. To analyze the responses to the closed questions, we initially employed descriptive statistics to gain a clearer understanding of the data. For ordinal and interval data, we focused on the mode and median to assess central tendency,

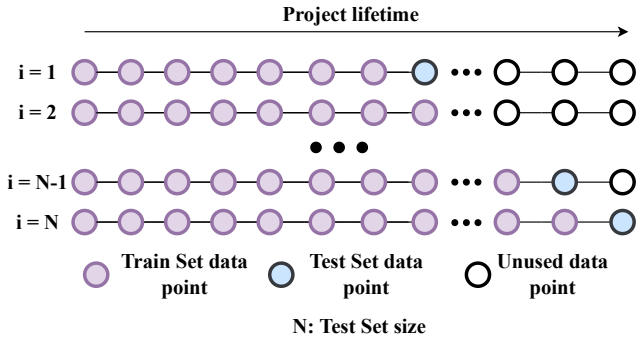


Figure 3: Walk-Forward Train-Testing approach. (*i*: New observation.)

while for nominal data, we calculated the distribution of participants' choices for each option.

Regarding the Likert scale question, Q_{12}, the possible value ranged from 0 ("not at all useful") to 10 ("extremely useful"). We present both the values as they have been reported by the interviewees and also the interpretation of the Net Promoter Score (NPS) [64]. NPS is a widely used metric for measuring customer loyalty and satisfaction. It classifies respondents into three categories based on their likelihood to recommend a product, service, or company:

- **Promoters (9-10):** Highly satisfied and likely to advocate for the product.
- **Passives (7-8):** Moderately satisfied but not enthusiastic enough to promote it.
- **Detractors (0-6):** Unhappy customers who may discourage others from using the product.

The NPS is calculated as: NPS = Percentage of Promoters – Percentage of Detractors. Therefore, a higher NPS indicates stronger customer loyalty. Hence, we employ such a score to measure the practitioner's perceived value of our model.

Regarding open-ended questions, we employ qualitative data analysis techniques suggested by Strauss and Corbin [65] and Seaman and Yuepu [66]. Qualitative analysis assists in addressing questions such as "What is happening in this situation?" when we aim to uncover how individuals understand their experiences and manage them over time in amidst evolving conditions [41].

We adopted an inductive approach to develop a new theory based on qualitative data. The open questions for $RQ_{6.1}$ and $RQ_{6.2}$ were manually coded as follows. Two authors independently coded responses to related questions adopting thematic analysis [65, 27]. We addressed disagreement through discussion, and codes were organized into a hierarchy of benefits and limitations until saturation was reached.

3.7. Model Execution (RQ_1 - RQ_5)

In this section, we textualize the description of the model execution process of the defined TSA models to facilitate practitioners' understanding of the results presented in Section 4 and encourage the use of the proposed models in the SE community. However, the prediction process described in Section 3.6.6 is followed similarly for all projects with each of the prediction models considered using different data sets. After the prediction process is completed, the results are visualized and, therefore, organized by aggregating the results provided by each model with all the collected projects for the two defined data sets (biweekly data and monthly data). The aggregation of the results per data set for each model combination is done by taking the average performance of the different projects in the respective data set as displayed in Table 5 and Table 6 for instance.

3.7.1. ARIMAX & SARIMAX models (RQ_1 - RQ_4)

Within the context of the modeling process, both the ARIMAX and SARIMAX models are built similarly in their initial stages. The last one stands as an adjusted version of the former counterpart. Therefore, for each project, we built both an ARIMAX model and a SARIMAX model. We computed the exploratory analysis of the model variables to examine their distributional characteristics (see Table 1 and Figure 1). All model variables denoted high skewness values, which was visually represented in the displayed distribution boxplots. Therefore, we applied the *log-transform* technique on the independent variables of the model to standardize their distribution and thus reduce the complexity of the model variables during the learning process. Variable transformation is a common preprocessing technique used in the field of predictive analytics due to its benefits in reducing the complexity of the model and increasing the prediction performance [67].

The first step of the modeling process consisted of leveraging the *Backward variable selection* technique [33]. The stepwise backward procedure begins with fitting the model with the set of important independent variables initially considered. Subsequently, it sequentially assesses the model and removes variables from the model to train it back. In each sequence, it selects the variable whose removal improves the goodness of fit of the model. The process stops when any further removal leads to a poorer model fit, therefore selecting the set of independent variables that provides the most optimal model results. To assess the impact of each combination of variables accordingly and following the concepts described in Section 3.6.3, we performed the model parameter tuning approach with the *Auto-arima* algorithm to obtain the results of the criteria AIC and BIC. We provide a concise description of the parameter tuning process in appendix A.

Figure 4 provides a graphical representation of the seasonal decomposition process performed for the SARIMAX model with the biweekly SQALE index data of the *Apache httpcore* project to achieve the best model parameters Described in a top-down structure, the first plot provides the observed progress of the SQALE index across time, the second and the third plots provide the existing trend and seasonality pattern, and the fourth one depicts the remaining residuals or *noise* in the data after de-trending and de-serializing has been performed The seasonal decomposition process is conducted every time a seasonally adjusted model is built Hence, we provide the visualization of the seasonal decomposition for

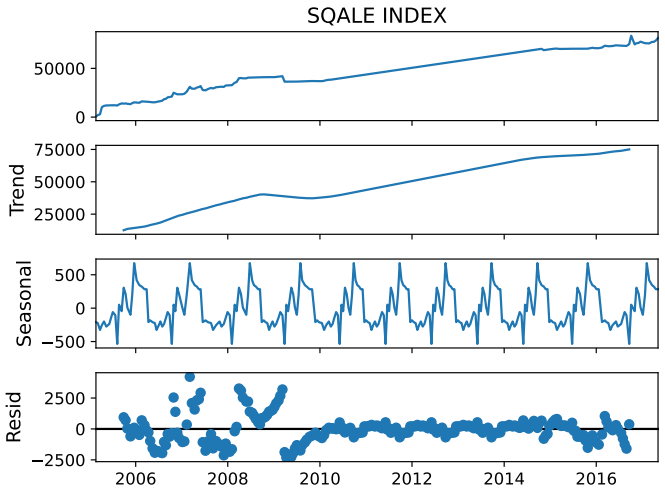


Figure 4: Seasonal decomposition of the SQALE index for project *httpcore* with *biweekly* data.

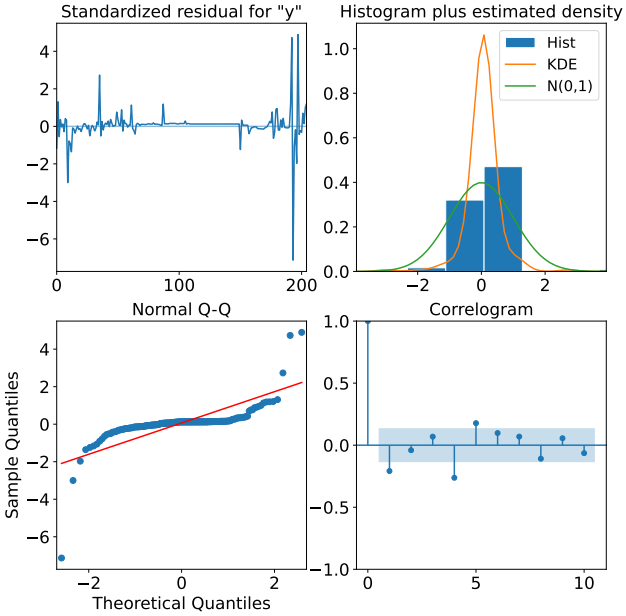


Figure 5: Final model diagnostics of the SARIMAX model built for project *httpcore* with *biweekly* data.

each of the considered projects in the provided online appendix (see Section 9).

For each of the studied software projects, we performed the multivariate TSA model fitting following the explained stepwise process. After each process, the model would provide the best goodness-of-fit results for the fitted data, and normally distributed model residuals, therefore showing good model quality. We provide a graphical representation of this stage in Figure 5, where the model diagnostics are presented for the same example project as in Figure 4.

The standardized residuals help to identify the existing anomalies or obvious patterns if the former ones are not centered around zero, therefore meaning that some patterns remain uncaptured by the model. The second plot provides the histogram of the model residuals, where in comparison with the standard normal distribution the kernel density estimate seems to be approximately normally distributed. In the third plot, the model residuals are compared against the Q-Q plot standards, showing a deviated distribution of the residuals from normality. The correlogram or Auto Correlation Function (ACF) plot shows the residuals autocorrelations, which should not be present after the model is fitted. Hence, they should not be exaggerated outside the marked confidence interval.

Finally, we tested the resulting models through the performance evaluation process explained in Section 3.6.6, and similarly, we calculated the results from the performance metrics as described in Section 3.6.5.

3.7.2. ARIMA+LM and SARIMA + LM models (RQ_1 - RQ_4)

Similarly to the previous section, the ARIMA+LM and SARIMA+LM approaches share the same model-fitting logic, with the difference being the adjustment of the seasonality effect. Therefore, for the sake of brevity, if we consider, for instance, the SARIMA+LM approach, it fits a SARIMA univariate model for each of the independent variables, thus predicting the future values of each variable independently and only based on the past historical observations. The LM model is then fitted with the predicted values of the independent variables to predict future values of the SQALE index (see Section 3.6.1). Then, we perform the parameter turning process for each of the models to achieve the best combination of model parameters to be used for prediction, as shown in 3.6.3. We then predicted the future values of each of the independent variables through their trained SARIMA models, respectively, following the performance evaluation methodology described in Section 3.6.6. With the future values of the independent variables already predicted, we trained the LM model with the training set of the historical SQALE index values along with the historical values of the independent variables. Thus, during the testing stage, we used the independently predicted values of the independent variables to predict future SQALE index values. The performance evaluation of the linear regression model was calculated in the same format as previously done in Section 3.7.1.

3.7.3. Performing long-term forecasting with TSA methods (RQ_5)

Following the motivation stated in Section 3 for RQ_5, this section presents the process executed to perform long-term forecasting with TSA methods. For that, we selected the TSA methods that show better performance from the results obtained in the model comparison analysis performed to answer RQs 1 to 4. To provide the models considered with a longer forecast horizon, we selected an initial data split of 70% for training and 30% to test the data set in which each model presented their best performance accordingly

(biweekly or monthly data). We used the best model parameter and variable configurations (see Table B.18 obtained from the previous analysis stages to provide the forecasts with the best detected model settings.

To put ourselves in the shoes of system engineers, the MAPE metric was chosen to quantify the prediction error obtained in each of the forecasted time-steps, given its easier interpretability. Since we are interested in evaluating the long-term forecast performance of the models considered, we did not retrain the models with each subsequent time point tested as described in Section 3.6.6.

Moreover, since we are calculating the performance level of TSA models across multiple software projects, we aggregated the MAPE results in each time-point, and computed mean, variance, maximum, minimum and median statistics from the drawn distributions. Since the task of assessing "once and for all" what the right precision would be due to the "magic" nature of thresholds, we also extend the interpretation of our results in the discussion section.

4. Results

In this section, we report the results obtained by answering our RQs. The data set included original commit data for 31 OS Java projects, and serialization and posterior linear interpolation created data serialized in biweekly and monthly time series. We successfully processed 14 software projects with all the models used in this study. We encountered convergence issues with linear algebra while fitting the TSA models. We report this issue in the model fitting stage as a threat to the validity of this study, which we formally state in Section 6 and provide further details on it later in this section. Therefore, since we encountered this issue in the execution of different models across the collected projects and for the sake of fairness in the models' results, we excluded projects that could not provide results for all the models considered in the study. To answer our RQs, we present the results of the analysis performed as the average score of the outputs obtained with each prediction model for the defined prediction performance metrics. Similarly, we perform the aggregation step separately for the biweekly and monthly data results. Figure 7 illustrates the MAE results for the models executed in the projects analyzed. We provide all results and tables before aggregation in the shared online package (see Section 9).

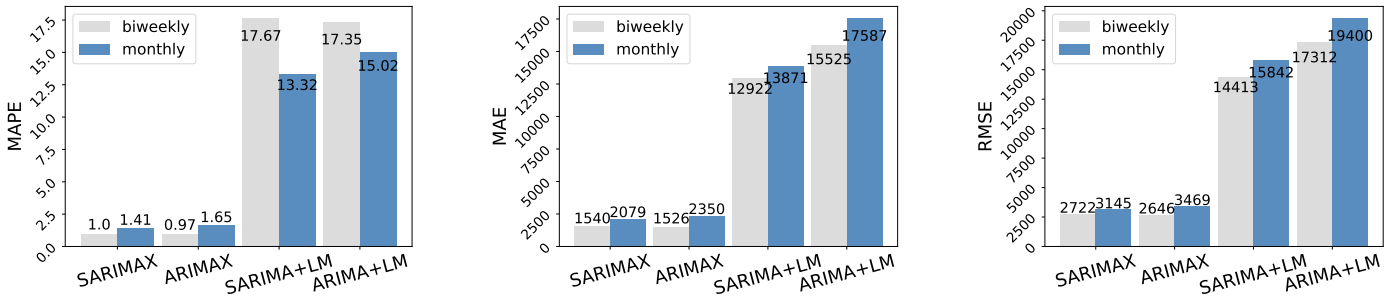
4.1. Descriptive Analysis

We now present the descriptive statistics of the subjects initially considered before building our prediction models. Table B.19 (see the appendix) presents the statistical descriptive characteristics of the projects based on the raw data set, as well as the generated biweekly and monthly time series data. According to Table B.19, we identified two opposite trends in the number of SQ analysis executions carried out by the study projects. Certain projects exhibited a decrease in the number of SQ analysis executions when the original observations from the raw data set were serialized into time series data (e.g., *Felix* in Table B.19). This finding indicated that these projects registered brief periods of time in which an elevated number of SQ analysis executions were performed, which subsequently decreased over time, resulting in the absence of SQ analysis executions during extended periods. Therefore, while generating the time series data, multiple executions of the SQ analysis that occurred within a biweekly or monthly time observation were converted into a single observation, as explained in Section 3.4.2. However, some other projects presented the opposite scenario, as their number of observations increased when the time series data were generated (e.g. *fileupload* in Table B.19). This finding indicated that these projects registered a limited number of SQ analysis executions during the time frame used to collect data for the Technical Debt dataset. Consequently, this type of project required data interpolation for some of the time-series observations due to the absence of real SQ analysis executions.

At this stage of the analysis, we realized that, depending on the frequency of the SQ analysis executions, the quality of the data would vary. The variations presented suggested that performing static analysis with tools like SQ in a fixed frequency could result in concise, authentic, and well-monitored time series data on the state of the quality of the code base of projects. In contrast, less frequent or randomly executed SQ analyzes provide insight into the code base throughout time but leave notable periods with missing data, thus requiring the action of data interpolation to perform TSA. Moreover, due to missing data for some of the projects, the algorithms of the adopted TSA models failed to converge during the model fitting stage described in Section 3.6.1. Due to the existing complexity in the data, we encountered several linear algebra decomposition issues [68] involved in the parameter estimation process while fitting the TSA models. As mentioned earlier, we leveraged preprocessing the set of independent variables within the model training through the backward variable selection criteria and standardization. However, Table B.20 (see the Appendix) reports the cases in which the adopted models reported linear algebra decomposition issues, which impeded obtaining results for all the models in each project that reported issues, respectively.

4.2. On the performance of Code TD prediction with existing time-dependent approaches (RQ_1)

We conducted the prediction analysis described in Section 3.7 for each of the 31 projects in our data set. Figure 6 shows the unified graphical illustration of the predictive performance results of the performance metrics adopted. The displayed results demonstrate a clear superiority of ARIMAX compared to ARIMA+LM with biweekly and monthly serialized time series data. Taking into account the results for MAE and RMSE, the differences in prediction error between ARIMAX and ARIMA + LM are 14,000 and 14,666 within the biweekly data and 15,237 and 15,931 within the monthly data, thus denoting a remarkable difference between both approaches. Hence, we can affirm that **given the existing TSA models for code TD prediction, the ARIMAX model overcomes the predictive performance of the ARIMA+LM model**. Finally, both models performed better with biweekly training data, as shown by MAE and RMSE. However,



(a) **MAPE** (b) **MAE** (c) **RMSE**

Figure 6: Prediction performance (MAPE, MAE, and RMSE) for the considered TSA models.

MAPE indicated an improved predictive performance for the ARIMA+LM model with monthly training data.

4.3. On the comparison of time-dependent approaches and ML models for Code TD prediction (RQ_2)

We compared the ARIMAX and ARIMA+LM models against the ML algorithms commonly adopted in the field. Tables 5 and 6 present the prediction performance results for the biweekly and monthly data, respectively. More specifically, **the tables provide the average results calculated from the prediction results obtained for the 14 projects**. The results of the individual projects are included in the replication package. It should be noted that, in both cases, using biweekly and monthly data, **ARIMAX model demonstrated a clear superiority over the rest of the models**, thus providing a more accurate prediction outcome for future code TD observations. Results such as 1,525.96 for MAE and 2,646.20 for RMSE within the biweekly results, compared to those presented by the following model after the ranking of results, RF, denote a quantified improvement of 717 and 1,232 in the described prediction errors accordingly. In addition, RF proved to be the second-best prediction model. As described previously, RF assumes nonlinear relationships between variables. Consequently, despite the indications that TSA models such as ARIMAX may be appropriate for predicting code TD, the potential for nonlinear models to capture intricate relationships cannot be discounted. Therefore, we observe that such a finding is especially relevant given the possibility of incorporating additional data and model predictors. Except for the special case of SVM, non-linear models consistently outperform linear models across all calculated metrics.

4.4. On the impact of seasonality on Code TD prediction in time-dependent approaches (RQ_3)

Following the set of defined research questions, we wanted to analyze the impact of addressing the seasonality effect on the prediction performance of TSA models through our RQ_3. Thus, we performed the prediction analysis described in Section 3.7 with SARIMAX and SARIMA+LM models. The process was similar to RQ_1 and used the exact data for this RQ. Figure 6 shows the unified graphical representation of

Table 5: Comparison of the predictive performance of TSA models against ML models for *biweekly* data. (*L*: Linear, *ML*: Non-linear)

Approach	MAPE (%)	MAE	RMSE
SARIMAX	1	1,539.92	2,721.95
SARIMA + LM	17.67	12,921.79	14,412.90
ARIMAX	**0.97**	**1,525.96**	**2,646.20**
ARIMA + LM	17.35	15,525.21	17,312.14
$MLR_{(L)}$	4.85	4,895.50	6,759.43
$SVM_{(NL)}$	23.87	38,929.00	39,904.36
$XGB_{(NL)}$	1.72	3,006.66	4,686.55
$RF_{(NL)}$	1.44	2,242.57	3,878.84
$SGD_{(L)}$	5.04	5,494.12	7,275.05
$L1_{(L)}$	5.18	5,431.16	7,112.04
$L2_{(L)}$	4.86	4,898.60	6,709.81

Table 6: Comparison of the predictive performance of TSA models against ML models for *monthly* data. (*L*: Linear, *ML*: Non-linear)

Approach	MAPE (%)	MAE	RMSE
SARIMAX	**1.41**	**2,078.64**	**3,145.27**
SARIMA + LM	13.32	13,871.48	15,842.22
ARIMAX	1.65	2,349.81	3,469.03
ARIMA + LM	15.02	17,586.97	19,400.41
$MLR_{(L)}$	4.62	3,978.65	5,567.30
$SVM_{(NL)}$	23.12	27,822.99	28,620.02
$XGB_{(NL)}$	5.80	4,600.92	5,950.18
$RF_{(NL)}$	1.90	2,572.85	4,202.80
$SGD_{(L)}$	4.89	4,478.40	5,956.63
$L1_{(L)}$	5.09	4,609.15	6,067.31
$L2_{(L)}$	4.50	3,706.95	5,192.83

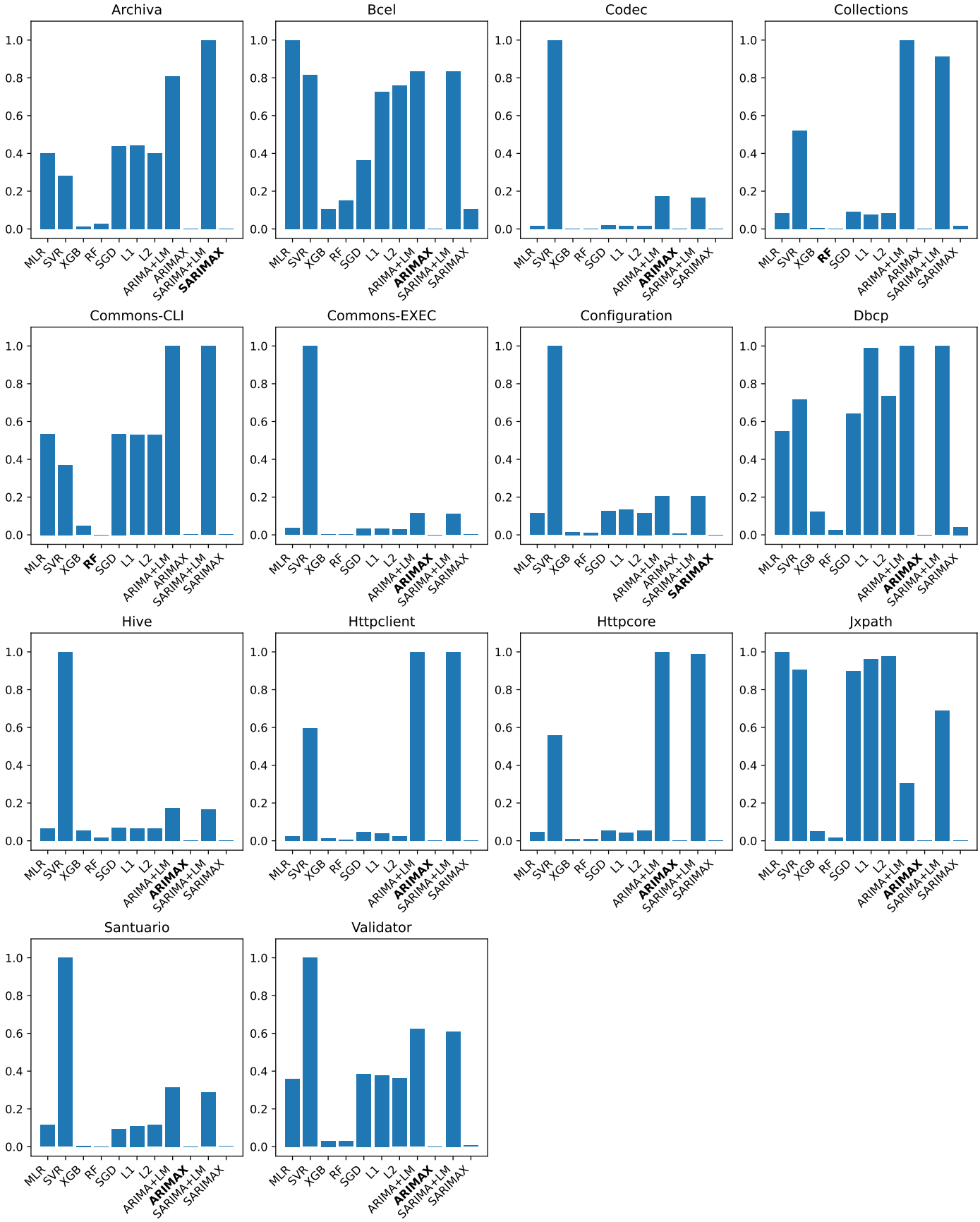


Figure 7: MAE prediction performance results using the biweekly data for the 14 analyzed projects.

the predictive performance results of the performance metrics adopted. To avoid overwhelming the reader with too many plots, we displayed the results for the seasonally adjusted models and their former counterparts, ARIMAX and ARIMA+LM. The results displayed demonstrate a systematic difference in prediction performance between the multivariate single model approach, i.e., ARIMAX and SARIMAX, compared to that of the multivariate combined model approach, i.e., ARIMA+LM and SARIMA+LM. Therefore, we can affirm that **the superiority in the prediction performance demonstrated on ARIMAX compared to ARIMA+LM is maintained with SARIMAX and SARIMA+LM when adjusting the models to capture seasonality**. Figure 6 demonstrates the clear superiority of SARIMAX over SARIMA + LM given serialized time series data biweekly and monthly. Hence, we can affirm that given the adjustment of the existing TSA models for code TD prediction, **the SARIMAX model outperforms the predictive performance of the SARIMA+LM model**. Specifically, for MAE and RMSE, the prediction error difference between SARIMAX and SARIMA+LM is 11,382 and 11,691 for biweekly data and 11,792 and 12,697 for monthly data, respectively. These values highlight a significant gap between the two approaches.

Hence, to answer RQ_3, the results displayed provide slightly similar results between the SARIMAX and ARIMAX models. The results for the biweekly data provided better prediction performance for ARIMAX than SARIMAX, while the opposite happened for the monthly data. MAPE results provided a worse prediction performance for SARIMA+LM than for ARIMA+LM when using biweekly data. However, SARIMA+LM overtook ARIMA+LM for the MAPE results when using monthly data and in the other prediction performance evaluation metric results. Therefore, we can affirm that **adjusting the TSA models to capture the seasonality effects of the data improves their prediction performance to some extent**.

4.5. On the comparison of seasonally adjusted time-dependent models and ML models for Code TD prediction (RQ_4)

We compared the previously confronted SARIMAX and SARIMA+LM models against the adopted ML algorithms commonly used in the field. Tables 5 and 6 present the prediction performance results for the SARIMAX and SARIMA + LM models compared to the adopted ML models. As expected from the results obtained for RQ_3, where SARIMAX provided similar prediction results to ARIMAX when using biweekly data and overtook ARIMAX when using monthly data, in this case, **the SARIMAX model also demonstrated a clear superiority among the rest of the compared models**, thus providing a more accurate prediction outcome for future code TD observations. It should be noted that capturing the seasonality effect did not change the key results observed for RQ_2. SARIMAX did not improve the results of the ARIMAX model using biweekly data, and the improvement experimentally compared to ARIMAX, with low monthly results. In addition, RF was shown to be the second-best prediction model. Therefore, even if the results suggest the suitability for linear models, nonlinear models may also offer the potential to capture more complex relationships. Looking at the rest of the ML models and excluding the special case of the SVM model, non-linear models present more accurate results than linear models in all the calculated metrics.

4.6. On the long-term forecasting accuracy with TSA models (RQ_5)

Considering the single-step prediction performance presented in Section 4.4 and Section 4.5, the results of this study provided practitioners with two main TSA models that performed well in predicting future code TD one month ahead and two weeks ahead, we extend the result of the resulting best models, ARIMAX and SARIMAX, by exploring their long-term forecast performance.

Following the data analysis process described in Section 3.7.2 the ARIMAX model was trained with the biweekly dataset, and the SARIMAX model was trained with the monthly data set. These models were selected based on their demonstrated superior performance among the models compared in the study, and each serialized data set was used accordingly. For each project time series data, the models were trained with 70% of the series, the remaining 30% being used to assess the performance of the forecasts without retraining the models. Figure 8 illustrates the results of the long-term prediction of the SQALE index for the SARIMAX model visualized through the quantified MAPE errors, considering the monthly time series data for the 14 projects. Similarly, Figure 9 presents the MAPE errors resulting from the long-term SQALE index forecasting for the ARIMAX model, considering the biweekly time series data in the equivalent number of projects.

It is important to note that each time point in the X-axis of both figures represents a new time period; for instance, time point 1 in the first figure represents the forecast for the future next month from the current time point. We displayed a box plot for each future time observation with the distribution of the resulting MAPE values from forecasts made on the resulting projects. Within the displayed boxplots we can find the median value of the obtained MAPE result, as well as the maximum and minimum values. Given the nature of the data, code TD, we considered limiting the total forecasting window length to 3 years, counted as 36 months and 72 biweekly periods.

Also, given that some projects had longer history data than others (see Table B.19 in the appendix), we could observe sudden drops in the magnitude of the boxplot in some time points, which depicted the end of the historical data of one of the resulting projects. However, this fact did not affect the evidence of a constant low-variance increasing cross-project code TD trend, which suggested a clear decline in the forecasting performance of the trained models, yet balanced over time. Furthermore, following the reported analysis design, we also calculated summary statistics from the aggregated MAPE results to quantitatively complement the displayed

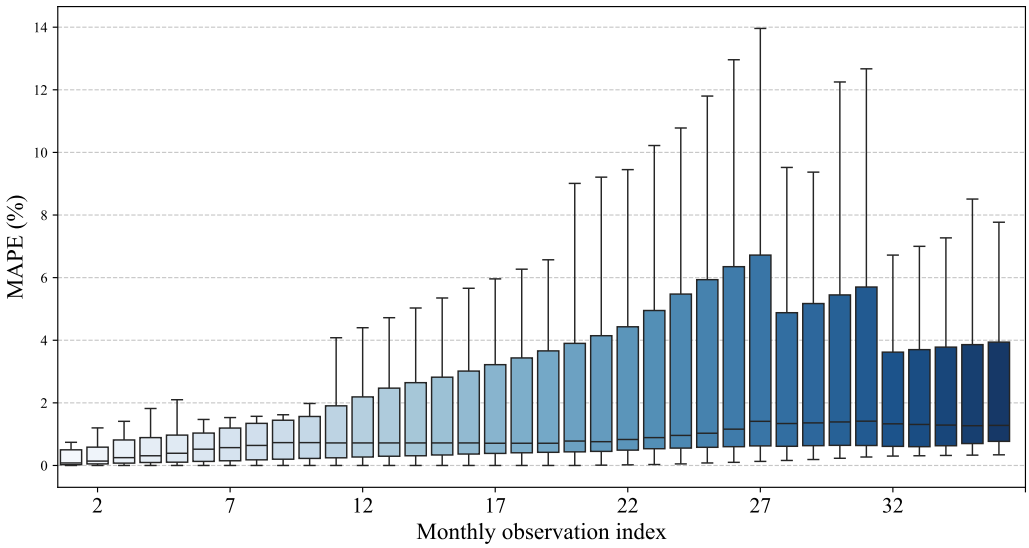


Figure 8: SARIMAX 36-month MAPE results with monthly data.

forecast trends. Table B.21 and Table B.22 in the appendix present the results of the summary statistics.

The resulting values of the forecasts performed reaffirmed the key takeaways observed from the temporal trends previously displayed in Figure 8 and Figure 9. On the one hand, we observed a clear increase in the forecasting error quantified through the average MAPE values, a trend equally observed in the values for the median and the variation. On the other hand, we observed a relatively balanced trend in all the descriptive statistics mentioned. For example, interpreting the results from the perspective of a practitioner, the mean of the resulting aggregated monthly MAPE error from the SARIMAX model (see Table B.22) increased only to 1.08% for a forecast window of six months, 2.33% for a forecast of twelve months, and 4.19% for forecasts 36 months ahead of the current point. Regarding the variance of the results in the same table, its value was maintained below 5% six months ahead of the current time point and almost reached less than 15% of the variance being twelve months ahead. Looking at the forecasting results from the ARIMAX model with biweekly time series data, the increase in statistics such as the mean and the median remained balanced, although the variance of the distribution was already from the first forecast biweekly time point. However, the mean of the MAPE error values obtained remained at 3.15% after 12 periods of biweekly (six months) and 4.56% for 24 periods (twelve months). Even if the mean value oscillated over higher values in the subsequent biweekly periods, the final value after 72 periods (thirty-six months) from the current time point remained at 5.97%.

4.7. On the practitioners' forecasting perceived valuable (RQ_6)

Table 7 shows the score for the perceived value of our model. Using the Likert scale provided in the questionnaire and reported in Table 8, according to Table 7, 70% of the participants perceived our model as useful to extremely useful, thus showing great appreciation for its possible effectiveness in the prediction of TD. 21% of the interviewees deemed our model moderately useful and only 8% deemed it very lowly useful. More specifically, Table 9 presents the distribution of responses according to the Net Promoter Score (NPS) scale. The table categorizes the respondents into three groups: *Promoters* (who rate the tool between 9 and 10), *Passives* (who rate it between 7 and 8), and *Detractors* (who rate it between 0 and 6). These categories reflect varying levels of enthusiasm and the likelihood of recommending the tool.

According to Table 9, the majority of respondents fall into the **Passives** category (47.83%), indicating a generally satisfied user base that does not strongly advocate for the tool. The **Promoters** constitute 21.74% of responses, representing those who are highly enthusiastic and are likely to recommend it. Conversely, **Detractors** account for 30.43%, suggesting a notable portion of users who may be dissatisfied or unlikely to endorse the tool. The computed NPS is 39.1, which falls into the "**Neutral to Good**." This means that while there is a reasonable level of positive sentiment, there is still room for improvement in converting passive and detracting users into promoters.

Table 10 also describes the justifications for these ratings. The strongest justification for positive ratings is that 17.39% of the respondents identify the effectiveness the model could bring to the monitoring and treatment of TD. Another 13.04% value its predictive aspect, citing its ability to prevent and control. Some of the participants (4.35%) liked its application in effort estimation, planning, and risk reduction, reaffirming the real-world advantages of our solution. In addition, none of the participants rejected the model as completely useless (0%), validating the usefulness of our model.

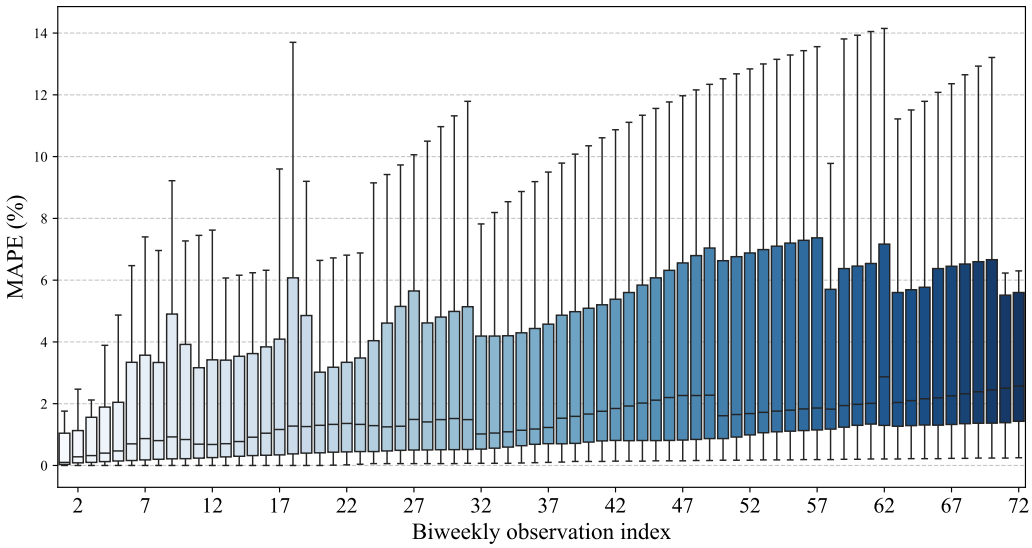


Figure 9: ARIMAX 72-biweekly (36 months) MAPE results with biweekly data.

Table 7: Perceived Value of Our model - (Q_{12})

Value	0-1	2-4	5-6	7-10
%	0	8	21	70

Table 8: Survey Scale for Model Usefulness

Score	**Interpretation**
0	Not useful at all – The model provides no value and does not meet any expectations.
1-2	Very low usefulness – The model is barely functional or relevant, offering minimal benefit.
3-4	Low usefulness – Some minor utility, but the model does not sufficiently address user needs.
5-6	Moderate usefulness – The model is somewhat helpful, but it has limitations that impact its effectiveness.
7-8	Useful – The model is generally valuable and meets expectations, though some improvements could enhance its impact.
9	Very useful – The model is highly effective, providing strong value with only minor areas for improvement.
10	Extremely useful – The model fully meets or exceeds expectations, offering exceptional value.

Table 9: NPS of the Perceived Value of Our model - (Q_{12})

NPS Category	**Percentage (%)**
Promoters (9-10)	21.74
Passives (7-8)	47.83
Detractors (0-6)	30.43
NPS	39.13
Interpretation	Neutral to Good

Table 10: Motivation for the chosen value (Table 7) - (Q_{13})

Motivation	**%**
A precise tool would improve effectiveness of addressing & tracking TD	17.39
Forecasting is useful for proactive management & prevention	13.04
Useful for estimating effort, scheduling & reducing risks	4.35
Mixed opinion (useful but with limitations)	4.35
Not useful or low relevance	0

Table 11 presents the trade-off between the desired forecast horizon and MAPE, and the percentage of practitioners who preferred each forecast horizon. The findings show a clear trade-off between prediction accuracy and the desired forecasting horizon, capturing the balance that practitioners seek between accuracy and practicability.

The most favored forecast horizons are 4 weeks (1.83% MAPE, chosen by 48% of the respondents), 2 weeks (1.44% MAPE, 30%), and 6 weeks (2.14% MAPE, 30%). They are the most preferred group, where respondents realize the best balance between forecast accuracy and ease of use. Interestingly, the four-week period stands as the most favored duration overall, presumably since it has the smallest prediction error coupled with the length that most closely resembles common sprint lengths for agile methodologies.

Beyond this range, as the forecast horizon extends beyond 8 weeks, MAPE increases linearly, exceeding 3% at 10 weeks and 5% at 14 weeks. The declining interest of practitioners with longer forecast horizons indicates that, although longer-term forecasts continue to be provided, the growing uncertainty reduces their functional attractiveness.

Table 12 classifies the explanations for interval preference

Table 11: Practitioners' Preferred Forecasting Intervals and Prediction Accuracy (Weeks) - (Q_{14})

Weeks	MAPE (%)	%
2	1.44	30
4	1.83	48
6	2.14	30
8	2.96	26
10	3.51	13
12	4.41	4
14	5.09	9
16	5.62	9
18	6.09	4
22	3.00	4
24	3.15	4
26	3.29	4
28	3.43	4
30	3.54	4
32	3.69	4
34	3.84	4
36	4.06	4
38	4.01	4
40	3.84	4
42	4.03	4
44	4.21	4
46	4.39	4
48	4.56	4

forecasting into four general themes. The most common category, Short-Term Preference (33%), explains that the majority of practitioners prefer shorter forecast horizons (e.g., 2-6 weeks), which is aligned with agile processes and regular technical debt fixing. Suggest an industrial context that requires timely, actionable information instead of long-term forecasts. The second highest theme, Medium-Term Balance (25%), implies that there are practitioners who prefer a moderate forecasting period (e.g., 8-14 weeks) and therefore achieve a moderate balance between accuracy and actionable planning. These respondents probably aim to align technical debt planning with milestone-based development cycles. The accuracy and forecast utility (21%) contains reasons centered on accuracy and error minimization. These professionals regard precise forecasts with minimal deviation so that decisions made with the model remain actionable and trustworthy.

Technical and Business Constraints (21%) express external constraints such as hardware obsolescence, product stability, or the feasibility of resolving types of technical debt. This would imply that forecasting needs to be domain-specific and reflect the constraints on the software being developed.

Table 13 shows the correlation between the forecast horizons in months, the accuracy of the forecast (MAPE), and the desires of the practitioner. The findings reveal that very accurate near-term forecasts are desired, with 52% of the practitioners requesting a 1-month forecast (0.30% MAPE) and 48% wanting a 2-month horizon (0.34% MAPE). This reveals that practitioners desire very accurate near-term forecasts for their technical debt planning.

Table 12: Thematic Coding of Forecasting Preferences - (Q_{15})

Theme	Percentage (%)
Short-Term Preference	33
Medium-Term Balance	25
Accuracy and Forecasting Utility	21
Technical and Business Constraints	21

With a growing forecast horizon, preferences continue to decrease, with only 39% opting for a 3-month forecast (0.50% MAPE) and 22% for 4 months (0.61% MAPE). For more than 6 months, the preferences drop below 13%, which shows that longer-term forecasts are unrealistic because the error levels rise. The interest is maintained only by the practitioners 9% for up to 12 months, where MAPE is up to 2.33

Table 14 summarizes the drivers of the selection of the forecast window into four primary themes. The most common theme, Short-Term Preference (30%), is that practitioners like shorter forecasting windows (1-3 months) so that they can maintain pace with agility and responsiveness in their processes. This is also aligned with sprint-based planning and periodic technical debt pay-down. Medium-Term Planning (26%) is the second most frequent theme, and respondents indicated the 1-6 month timescale as the optimal balance of predictability and feasibility in plans. Practitioners generally accept that taking it beyond 6 months makes the forecast less useful because uncertainty increases. Accuracy and Practicality in Forecasting (22%) includes those that emphasize the importance of precision in prediction. Practitioners prefer shorter horizons with low MAPE so that the predictions remain valid and practical. They are also concerned that the long-term forecasts are not concrete and practical. Lastly, Business and Workflow Constraints (22%) are domain-specific issues such as ensuring product availability, prioritization of backlogs, and customer needs. The development cycles of most practitioners require technical debt to be tackled before large releases or in formalized reporting cycles, most practitioners claim.

Table 15 illustrates the intervals preferred by practitioners for forecasting technical debt. The responses are distributed equally in three large bins: Weekly (32%), Biweekly (32%) and Monthly (32%). It mirrors the fact that practitioners prefer frequent forecasting, possibly due to the changing nature of software development and the need to constantly re-estimate technical debt. Only a minority of 5% prefer a Sprint-Based Approach (Agile), which suggests that for others, the best forecast window is precisely in synchronization with sprint cycles and not hard-coded time frames. This supports agile practices in which technical debt management is integrated into iterative development phases.

Table 16 categorizes the motivations for the preferred forecasting windows of technical debt professionals. The most

Table 13: Practitioners' Preferred Forecasting Intervals and Prediction Accuracy (Months) - (Q_{16})

Months	MAPE (%)	%
1	0.30	52
2	0.34	48
3	0.50	39
4	0.61	22
5	0.73	13
6	1.08	26
7	1.38	13
8	1.61	9
9	1.79	13
10	1.98	9
11	2.15	9
12	2.33	9

Table 14: Thematic Coding of Forecasting Window Preferences - (Q_{17})

Theme	%
Short-Term Preference	30
Medium-Term Planning	26
Forecasting Accuracy & Practicality	22
Business and Workflow Constraints	22

common theme, Short-Term Preference (32%), suggests that most practitioners prefer TD predictions to occur often, which emphasizes the importance of quick feedback loops and the ability to act on evolving issues before they accumulate. The second most prevalent category, Regular Review and Planning Cycles (27%), focuses on alignment with existing workflows, such as sprints and formalized reviews. The majority of respondents favor forecast intervals that are aligned with agile iterations so that technical debt can be monitored and dealt with regularly within development cycles.

Balanced Practicality and Granularity (23%) recognizes drivers who seek a compromise between high frequency and low overhead. Monthly forecasting, for instance, is interruptive enough to be actionable but not so interruptive as to cause problems. Practitioners know that lower frequencies are more precise but are not always practical for a long-term strategy. Lastly, Domain-Specific Constraints (18%) incorporate idiosyncratic considerations in terms of practitioners' work environments. This includes energy-efficient systems that must be updated regularly, varied levels of interaction with TD prediction, and other context constraints that affect how frequently TD forecasting is possible.

5. Discussion

The objective of the presented study was to analyze the impact of temporal factors in code TD prediction. Code TD was approximated using the SQALE index metric calculated by SQ, and the predictions were made including the set of different types of code smell issues in the projects within the

Table 15: Preferred Forecasting Window for TD Prediction - (Q_{18})

Preferred Forecasting Window	%
Weekly	32
Bi-Weekly	32
Monthly	32
Sprint-Based (Agile)	5

Table 16: Thematic Coding of Forecasting Window Motivations - (Q_{19})

Theme	%
Short-Term Preference	32
Regular Planning and Review Cycles	27
Balanced Practicality & Granularity	23
Domain-Specific Constraints	18

prediction models. The adoption of TSA prediction models enabled empirical demonstration of the suitability of time-dependent techniques for code TD prediction, as well as comparative analysis with commonly used ML prediction algorithms. Consequently, 11 different prediction models were constructed for each project. The analysis included a comprehensive data collection, preprocessing, and examination of 31 open-source Java projects. However, to ensure the integrity and fairness of the results, only the outcomes of the 14 projects that provided complete results with the adopted prediction models were considered.

Our study aimed to answer six research questions. The first research question aimed to compare the predictive performance of two time-dependent approaches based on ARIMA already implemented in the academic literature given the same experimental setting. The confronted models were the ARIMAX model previously implemented by Mathioudaki et al. [11], and the ARIMA+LM model previously implemented by Zozas et al. [12]. Within the same setting for our first research question, we also assumed before running the study that we expected the accuracy from the ARIMAX model, given its existing theoretical background and the factor that it considers as a model variable the autoregression of its dependent variable, would be superior to that of the ARIMA+LM, which comprises the mixture of two different families of statistical models, and performs a linear regression on the dependent variable without considering its autoregressive component as an additional variable within the model. Thus, looking at the results, our initial takeaway depicted the superiority of ARIMAX over the ARIMA+LM model in predicting the code TD. These results suggest that, given a time-dependent prediction context, **multivariate models that include and simultaneously predict independent variables as well within the prediction model, such as ARIMAX, succeed on providing robust results** as shown in Figure 6.

Furthermore, in the second research question, we proved the robustness reported with ARIMAX by comparing its Code TD prediction performance against a set of the most common ML algorithms adopted in the SE field. The results pro-

vided in Tables 6 and 5 corroborated this evidence. These results suggest the necessity of adopting models that consider the time dependence factor in the dependent variable to be predicted. However, it is worth noticing the close results reported by models such as Random Forest or Extreme Gradient Boost, prediction algorithms widely used in academia as well as in industry. Given the non-linear distributional assumption of these models and the oppositely linear distributional assumption, which is the foundation for TSA models such as ARIMAX, **leveraging the combination of these models through ensemble learning approaches might further improve the prediction performance reported in this study**.

In addition, we analyzed the impact of addressing the seasonal component within the time-series data on the obtained results. For that, we used the model extensions provided in ARIMA-founded models to analyze the results for the SARIMAX and SARIM + LM models. As a result of the analysis performed, we observed slight improvements in the initially adopted TSA models, thus depicting **a small impact on prediction performance by controlling the seasonality component**. This evidence suggests a small dependence on seasonality from the perspective of the OS software development community, which might not follow structured development schedules compared to the software development activity performed in the industry.

As the key takeaway of this study, we can confirm that time-dependent models **are applicable and competitive in predicting the TD code**, not only at the level of a single project, but also at the aggregate level, as we have demonstrated. More research is still needed to understand the methodology required to prepare the most suitable data for TSA models. It has been demonstrated that if the data fits well in the TSA model, the latter can provide a high prediction performance. Therefore, this study suggests **the adoption of periodic software quality analysis practices within the project development process, thus providing a complete monitoring of metrics such as Code TD and better quality data for the improvement of prediction performance**. Similarly, it is important to note at this stage that when we make our predictions, these values are the obtained most likely estimates; therefore, we are not comparing our real values against the result of a simulation or an execution. TSA models can have a considerable prediction performance, but there is a need for further research in modeling Code TD prediction while considering time as a relevant factor.

5.1. Forecasting Code TD

The observed results from the long-term Code TD forecasting performance resulted in a set of key takeaways. Through the analysis implemented that addresses long-term forecasting, we could confirm **that time-dependent models can provide competitive code TD forecasting for single-step and multistep forecasting scenarios**, and therefore highlight **the importance of considering techniques that treat the temporal nature of the dependent variable** when performing data analysis on time-dependent variables. Still, more work needs to be done on long-term code TD forecasting to provide robust enough long-term maintenance guarantees to practitioners, especially given the volatility advertised in the variance observations of the results we obtained. Similarly, practitioners should benefit from different approaches of Code TD prediction, i.e. single-step forecasting or multi-step forecasting, based on the specific characteristics and needs of the software project workflows. On this basis, our work emphasizes **the benefits of structuring Code TD analysis executions in a serialized style** when working on maintenance of the software quality with tools like SQ.

5.2. Impact on Practitioners

We surveyed the opinion of **23** practitioners and their expertise profile. We asked them to evaluate the practical usability of our model and the perceived benefit or limitations it may have in an industrial context. Referring to our findings in the RQ_5, we presented the practitioner with the same time window of 36 months and 72 weeks. In particular, no one exceeded the 48-week mark, while only 4% on average selected a time window beyond 12 weeks. Furthermore, no instances surpassed the 12-month threshold.

According to Tables 7 and Table 10 describe the external validity of our TD prediction model, with overwhelming percentages of respondents knowing its ability to improve TD management. The high ratio of Promoters and Neutral-to-Promoter respondents indicates our approach is well perceived, justifying its use in real software engineering practice. Table 11 confirms the precision of our TD forecasting model within short- to medium-term horizons, specifically the 2-6 week horizon, where precision is best and trust from practitioners highest.

Generally speaking, our survey findings confirm that our forecasting model best suits industry needs, particularly in the case of short- to medium-term planning, where practitioners tend to prefer its forecasts. In fact, according to Table 12, practitioners favor a good accuracy with a medium time window over the planning horizon, where the ideal is between 1 and 3 months. This aligns perfectly with typical agile development cycles and decision-making milestones, confirming that our approach is meaningful within substantial periods in industry.

Moreover, Table 14 validates the accuracy of our model's performance, particularly in short- to medium-term forecasting, which has been widely accepted to be informative, accurate, and actionable for real software development processes. Table 15 reiterates that short-range forecasting is crucial to practitioners because it places technical debt in continuous observation and is resolved within appropriate actionable time frames. The lack of preference for longer-range forecasting (e.g., quarterly or yearly) also emphasizes the need for near-term and actionable forecasts over long-horizon predictions.

Finally, Table 16 shows that our forecasting approach aligns more closely with what the industry desires, particularly in short- to medium-term window lengths, where usability, precision and integration are the most valued by practitioners.

6. Threats to Validity

Construct validity. Some projects have been affected by only a few code smell rule issues, so we decided to exclude them because their independent variables were considered uninformative. Additionally, irregularities in commit data can cause missing data for the generated periodical time series. The threat was minimized by linearly interpolating the values for the missing periods. Approximations cannot be as accurate as using real data; therefore, we could expect different results if periodic complete real data existed. An additional potential threat to construct validity is identification and TD quantification, namely to the validity of code smells as predictors for TD. Although SonarQube's remediation time is widely used, other work [69, 70, 71] demonstrated that it is not always a good estimator of the actual effort to remove TD. So, our results may be affected by potential TD measurement errors. To restrict this risk, we compared our approach with existing research on TD assessment and ensured that our approach was consistent with current best practice. However, we acknowledge that variability in the estimation of TD is a feature of automated analysis and that further research should explore other methods of quantification of TD to further validate our results.

Internal validity. We selected the SQALE index as the variable of interest, given its capacity to depict technical debt. We then established a set of code smell violation rules reported by SQ as the underlying reason for the existence of the given SQALE index metric. This served as the initial set of independent variables in our study. However, we did not consider collecting additional variables that could have an impact on the results. Consequently, alternative choices of independent variables may yield different results. Concerning the impact on the remediation time estimate provided by the collected SQALE index data, previous research has demonstrated that the SQALE index provides overestimations of the real code TD. Therefore, this study uses the SQALE index as a proxy for technical debt, as well as the independent variables for code smell rule violations in our models. We acknowledge that potential overestimations in the SQ estimated SQALE index data could affect the practical applicability of the study findings. In future research, we will mitigate this threat by incorporating more accurate estimates of remediation time into models, or complementing the models with real-effort data to improve the model's reliability.

External validity. The study subjects were mature open source projects written in Java that met the rigorous quality standards of ASF. The included projects represent a wide spectrum of application domains, including external libraries, frameworks, web utilities, and substantial computational infrastructures. Hence, the obtained results can be generalized to projects with similar characteristics, and therefore nonmature projects, retired projects, and projects not using SonarQube are excluded.

Conclusion validity. This study applied commonly known statistical and ML techniques. During data analysis, we ensured that the assumptions of the techniques were fulfilled. However, a low number of data points in some projects can reduce the prediction power of the models used. Additionally, the ARIMA + LM model may not capture the time-dependent nature of the SQALE index, as its prediction is performed through a Multiple Linear Regression model, which does not consider the time-dependent factor. Regarding the reliability of the measures, the variables' measurements were collected from the data set and the construction of the final set of model variables was carried out through an automated process. For the sake of the reliability of the results, we excluded the collected projects which did not allow the algorithms from the adopted models to converge within the model fitting. Therefore, we report only the aggregated results from the projects that provided results for the entire set of models considered in this study.

7. Related Work

This section reports related work, including the current state of Technical Debt (code TD) prediction and a detailed comparison with our work in Table 17.

Tsoukalas et al. [10] investigated the effectiveness of ML techniques in modeling and predicting the evolution of the code TD. Specifically, they collected weekly commit-level snapshot observations over three years for a total of 15 software projects. They used the correlation analysis technique to perform feature selection and the grid search method to tune the model hyperparameters. They implemented a set of different well-known ML models (see Table 17) and trained and tested them through the Walk-Forward (WF) Train-Test method, considering MAPE as a performance assessment metric. They provided the results, scripts, and data sets for the study. Similarly, they introduced an industrial survey to empirically assess the significance of the TD approach introduced in their work.

In a second work, Tsoukalas et al. [9] adopted a different set of ML models to classify code TD classes (High/Not-High TD), analyzing 25 open-source projects. They performed hypothesis testing for the selection of the most significant independent variables and tuned the parameters of the selected models through the grid-search method. In this second work, they used commonly known classification assessment methods such as AUC, F2 or Recall, among others, in a repeated stratified cross-validation setting to assess their results. Similarly, as in their previous work, they shared the study's results, scripts, and datasets.

Mathioudaki et al. [54] explored the application of deep learning methods (DL) compared to univariate approaches already implemented such as ARIMA or RF models to improve the precision of long-term code prediction. TD. To achieve this, they created, evaluated, and juxtaposed DL models using a data set that includes five prominent real-world software applications sourced from the Technical Debt dataset version 1 [28]. They preprocessed the adopted

Table 17: Related work on the existing research on TD prediction

	Tsoukalas et al. [10]	Tsoukalas et al. [9]	Mathioudaki et al. [54]	Aversano et al. [55]	Mathioudaki et al. [11]	Zozas et al. [12]	Our work
Seasonality							✓
Time Series (TS) processing methodology	Weekly snapshots across 3 years & Walk-Forward Train-Test	No serialization & 10-K CV (No TS methodology)	Sliding Window over project commits & Walk-Forward Train-Test	Raw commit-level observations & Repeated CV (No TS methodolofy)	Last commit of the week as weekly observation & Walk-Forward Train-Test	Raw release data & Random train-test split & Walk-Forward Train-Test	Biweekly and Monthly serialization of SQ analyses & Walk-Forward Train-Test
Feature selection	Correlation analysis	Hypothesis testing		Correlation analysis		BSR	Feature importance, Variance thresholding, Zero percentage, Correlation analysis
Hyperparameter tuning	Grid-search method	Grid-search method	Grid-search method		ACF, PACF	ACF, PACF	Auto-ARIMA
Univariate models		Univariate logistic regression	Multi-layer perceptron		ARIMA	ARIMA	ARIMA, SARIMA
Multivariate models	MLR, L1, L2, SGD, SVR (linear), SVR (rbf), RF (regression)	LR, NB, DT, KNN, SVM, RF, XGB		MLR, BDT, RF	ARIMAX	LM	ARIMAX, SARIMAX, ARIMA+LM, SARIMA+LM L1, L2, MLR, SGD, SVR, RF, XGB
Industrial survey	✓					✓	✓
Long-term forecasting	1 to 40 weeks		5 to 150 commits		4, 8, 12 steps	12 iterations	1 to 36 months
Time Series models			ARIMA		ARIMA, ARIMAX	ARIMA	ARIMAX, ARIMA+LM, SARIMAX, SARIMA+LM
ML models	MLR, L1, L2, SGD, SVR (linear), SVR (rbf), RF (regression)	LR, NB, DT, KNN, SVM, RF, XGB	RF	MLR, BDT, RF			L1, L2, MLR, SGD, SVR, RF, XGB
DL models			Multi-layer perceptron				
Cross-validation	Walk-Forward Train-Test	Repeated stratified CV	Walk-Forward Train-Test	Repeated CV	Walk-Forward Train-Test	Walk-Forward Train-Test	Walk-Forward Train-Test
Performance metrics	MAPE	Precision, Recall, F2, AUC, Module inspection	MAPE, MAE, RMSE	RMSE	MAPE, MAE, RMSE	MAPE, MAE, RMSE	MAPE, MAE, RMSE
Sample size	15 projects	25 projects	5 projects	8 projects	5 projects	105 projects	14 projects
Results discussed	✓	✓	✓	✓	✓	✓	✓
Scripts available	✓	✓					✓
Datasets available	✓	✓		✓		✓	✓

commit-level data through the Sliding Windows technique to transform the raw data into a processable time series for the chosen models. They also adopted the Grid Search method for model parameter tuning. The long-term prediction horizon ranged from 5 to 150 commit-ahead lengths. To assess the performance of the predictions, they considered the WF Train-Test technique and the MAPE, MAE, and RMSE performance metrics. The results revealed that DL techniques produce code TD prediction models with commendable predictive accuracy, extending up to 150 steps ahead in future prediction.

Aversano et al. [55] investigated the potential utility of software system quality metrics to accurately predict the TD code. The authors examined quality metrics from 8 distinct open source software systems and then fed those commit-level metrics into different ML algorithms to predict TD. They performed a correlation analysis for the selection of the features of the independent variables used in their models. They used the repeated cross-validation technique to assess their results, which were quantified through the RMSE. The results demonstrated strong predictive performance, and the suggested approach offers a valuable method to comprehend the practical aspects of the technical debt phenomenon.

Focusing on the work that adopted TSA models to predict code TD, we concentrated on two papers. Mathioudaki et al. [11] investigated the predictive capabilities of TSA models for the prediction of the code TD. They aimed to determine whether the incorporation of independent variables as predictors, known as Code TD predictors, into ARIMAX models [25, p. 451] could produce more precise Code TD predictions compared to conventional univariate Autoregressive Integrated Moving Average (ARIMA) models. Their investigation used five datasets that captured the historical evolution of software metrics derived from static code analysis in five long-standing projects. The raw data was serialized into weekly observation points by choosing the last commit performed in each weekly time window. They used Auto-Correlation Functions (ACF) and Partial Auto-Correlation Functions (PACF) for the parameter tuning of the chosen models. They generated predictions using both ARIMA and ARIMAX models for various time horizons on these data, exploring long-term forecasting horizons of 4, 8 and 12-week ahead periods. The results yielded a clear conclusion. Across the open-source software projects examined, the accuracy of the ARIMAX models exceeded that of the ARIMA models by a significant margin.

More recently, Zozas et al. [12] combined two models to predict code TD: Supervised Linear Regression (LM) and ARIMA. The authors used backward stepwise regression (BSR) to identify the most significant predictors to describe the TD response variable code. They used Auto-Correlation Functions (ACF) and Partial Auto-Correlation Functions (PACF) for the parameter tuning of the chosen models. They performed a univariate TSA prediction through ARIMA for each of the selected predictors to predict future values. Then, they used the regression model previously de-

veloped to estimate the future values of the response variable. To assess the performance of their models, they adopted the MAPE, MAE, and RMSE performance metrics within a WF Train-Test setting. They made 12-week long-term predictions ahead to explore the long-term prediction performance of the models studied. The results looked promising as an additional approach to predict TD efficiently, and they provided the results and data sets from their work. In addition, they organized a short survey with 15 JavaScript developers with medium experience to evaluate their findings.

In our work, instead of comparing univariate and multivariate TSA, we aim to directly compare different multivariate TSA approaches to investigate which approach obtains better prediction performance. Furthermore, we explore the impact of addressing seasonality as a critical component impacting the performance of TSA models on code TD prediction, and confront the performance of the considered TSA multivariate models towards a set of previously implemented ML techniques used for code TD. The selected predictors will be used to predict the value of the SQALE index, which will be our target value. Our approach relies on performing different multivariate applications of the ARIMA model. To test its performance, we aim to split the data for training and testing the model through the WF Train-Test approach and performance metrics such as MAPE, MAE and RMSE. We also aim to perform long-term forecasting with the best resulting TSA models with forecasting windows reaching the 3-year horizon. Moreover, we are interested in what preferred code TD forecasting horizon length, and we collected opinions from practitioners in a shared industrial survey.

Furthermore, since training ML models with a single feature can compromise their quality [72], we determine the optimal number of features and compare the performance of selected multivariate TSA models against a set of ML models chosen based on previous research.

8. Conclusions

We conducted an empirical study to compare two TSA approaches, their respective version adjusted to treat the seasonality component, and seven ML algorithms to code the prediction of TD as the SQALE index calculated by SQ. In addition, we conducted an industrial survey to empirically analyze the implications of our findings from the perspective of practitioners and tested the accuracy of the resulting best time-dependent models when performing long-term forecasting.

We trained the compared methods using Code TD observation data that were serialized into biweekly and monthly periodicity levels from the resulting number of 14 open source Java projects. The comparison denoted a clear superiority of the ARIMAX model compared to the other adopted models when training the models with biweekly time series. At the same time, SARIMAX provided better results with monthly time series. However, the results provided by SARIMAX suggested that the impact of seasonality could not substantially improve the predictive performance of ARIMAX, thus demonstrating that the ARIMAX model generates the main improvement between the performance of the ML algorithms and the TSA models. Our findings show that seasonality has little impact on the predictive performance of time-dependent techniques. However, incorporating time dependence improves the predictions over time-agnostic methods. Moreover, coupled with our data serialization process, it highlights the benefits of sequential software quality analysis in improving Code TD prediction.

Using the TSA models with the resulting best predictive performance, we performed long-term forecasting up to a forecasting horizon of three years translated into 36 months and 72 biweekly periods accordingly. The resulting forecasts looked promisingly balanced, which reaffirms the suitability of time-dependent techniques for Code TD predictions. Finally, our survey reveals good industry confidence in our tool and its effectiveness for short- to medium-term forecasting windows, which aligns closely with industry needs and agile methodology practice.

In future work, our aim is to analyze the implementation of transformer-driven prediction techniques, which allow the inclusion of low-level models into the multiple deep learning layers nested within the transformer architecture. Adapting TSA techniques with these models provides great modeling ability for long-range dependencies and interactions in sequential data and TSA models, thus further contributing to the research community with advanced predicting techniques and anticipating Code TD.

9. Data Availability Statement

To allow verifiability and replicability, we made the raw data and analysis files available in our online appendix. Furthermore, we provide instructions on how to use the released replication package in Appendix B, as well as the README file in our online appendix.[6]

References

[1] F. Palomba, D. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, A. Serebrenik, Beyond technical aspects: How do community smells influence the intensity of code smells?, IEEE Transactions on Software Engineering (2018) 1–1.

[2] F. Arcelli Fontana, V. Lenarduzzi, R. Roveda, D. Taibi, Are architectural smells independent from code smells? an empirical study, Journal of Systems and Software 154 (2019) 139–156.

[3] T. Sharma, P. Singh, D. Spinellis, An empirical investigation on the relationship between design and architecture smells, Empirical Software Engineering 25 (2020).

[4] S. Olbrich, D. S. Cruzes, V. Basili, N. Zazworka, The evolution and impact of code smells: A case study of two open source systems, in: International Symposium on Empirical Software Engineering and Measurement, pp. 390–400.

[5] D. I. K. Sjoberg, A. Yamashita, B. Anda, A. Mockus, T. Dyba, Quantifying the effect of code smells on maintenance effort, IEEE Trans. Softw. Eng. 39 (2013) 1144–1156.

[6] https://doi.org/10.5281/zenodo.14974421

[6] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, A. D. Lucia, On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation, Empirical Software Engineering 23 (2018) 1188–1221.
[7] V. Lenarduzzi, N. Saarimäki, D. Taibi, Some sonarqube issues have a significant but small effect on faults and changes. a large-scale empirical study, Journal of Systems and Software (2020) 110750.
[8] N. Saarimäki, S. Moreschini, F. Lomio, R. Peñaloza, V. Lenarduzzi, Towards a robust approach to analyze time-dependent data in software engineering, in: International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 36–40.
[9] D. Tsoukalas, N. Mittas, A. Chatzigeorgiou, D. Kehagias, A. Ampatzoglou, T. Amanatidis, L. Angelis, Machine learning for technical debt identification, IEEE Transactions on Software Engineering 48 (2021) 4892–4906.
[10] D. Tsoukalas, D. Kehagias, M. Siavvas, A. Chatzigeorgiou, Technical debt forecasting: An empirical study on open-source repositories, Journal of Systems and Software 170 (2020) 110777.
[11] M. Mathioudaki, D. Tsoukalas, M. Siavvas, D. Kehagias, Comparing univariate and multivariate time series models for technical debt forecasting, in: Computational Science and Its Applications, p. 62–78.
[12] I. Zozas, S. Bibi, A. Ampatzoglou, Forecasting the principal of code technical debt in javascript applications, IEEE Transactions on Software Engineering 49 (2023) 2498–2512.
[13] G. E. Box, S. C. Hillmer, G. C. Tiao, Analysis and modeling of seasonal time series, in: Seasonal analysis of economic time series, NBER, 1978, pp. 309–344.
[14] P. Bruce, A. Bruce, P. Gedeck, Practical statistics for data scientists: 50+ essential concepts using R and Python, O'Reilly Media, 2020.
[15] G. James, D. Witten, T. Hastie, R. Tibshirani, et al., An introduction to statistical learning, volume 112, Springer, 2013.
[16] J. Durbin, S. J. Koopman, Time series analysis by state space methods, Oxford University Press (UK), 2012.
[17] R. J. Hyndman, G. Athanasopoulos, Forecasting: principles and practice, OTexts, 2018.
[18] V. Lenarduzzi, A. Sillitti, D. Taibi., Analyzing forty years of software maintenance models, in: International Conference on Software Engineering Companion, ICSE-C '17, pp. 146–148.
[19] V. Lenarduzzi, A. Sillitti, D. Taibi, A survey on code analysis tools for software maintenance prediction, in: International Conference in Software Engineering for Defence Applications, Springer International Publishing, 2020, pp. 165–175.
[20] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, A. Zaidman, How developers engage with static analysis tools in different contexts, in: Empirical Software Engineering.
[21] F. Martin, B. Kent, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co., Inc. (1999).
[22] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, P. Vaillergues, The squale model — A practice-based industrial quality model, in: International Conference on Software Maintenance (ICSME), pp. 531–534.
[23] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Springer Publishing Company, Incorporated, 2000.
[24] V. Basili, G. Caldiera, D. Rombach, The goal question metric approach, Encyclopedia of Software Engineering (1994).
[25] K.-S. Chan, J. D. Cryer, Time series analysis with applications in R, Springer, 2008.
[26] J. Çarka, M. Esposito, D. Falessi, On effort-aware metrics for defect prediction, Empirical Software Engineering 27 (????) 152.
[27] M. Esposito, F. Palagiano, V. Lenarduzzi, D. Taibi, On large language models in mission-critical it governance: Are we ready yet?, arXiv preprint arXiv:2412.11698 (2024).
[28] V. Lenarduzzi, N. Saarimäki, D. Taibi, The technical debt dataset, in: Conference on PREdictive Models and data analycs In Software Engineering, PROMISE '19.
[29] M. Patton, Qualitative Evaluation and Research Methods, Sage, 2002.
[30] M. A. F. A. Fida, T. Ahmad, M. Ntahobari, Variance threshold as early screening to boruta feature selection for intrusion detection system, in: 2021 13th International Conference on Information & Communication Technology and System (ICTS), IEEE, pp. 46–50.
[31] P. D. Allison, Missing data, The SAGE handbook of quantitative methods in psychology (2009) 72–89.
[32] J. L. Speiser, M. E. Miller, J. Tooze, E. Ip, A comparison of random forest variable selection methods for classification prediction modeling, Expert systems with applications 134 (2019) 93–101.
[33] A. Agresti, Foundations of linear and generalized linear models, John Wiley & Sons, 2015.
[34] C. Spearman, The proof and measurement of association between two things. (1961).
[35] L. Yu, H. Liu, Feature selection for high-dimensional data: A fast correlation-based filter solution, in: International conference on machine learning, pp. 856–863.
[36] J. H. Kim, I. Choi, Choosing the level of significance: A decision-theoretic approach, Abacus 57 (2021) 27–71.
[37] T. Blu, P. Thévenaz, M. Unser, Linear interpolation revitalized, IEEE Transactions on Image Processing 13 (2004) 710–719.
[38] A. for Computing Machinery, Acm publications policy on research involving human participants and subjects, `https://www.acm.org/publications/policies/research-involving-human-participants-and-subjects`, 2021.
[39] C. Wohlin, P. Runeson, M. Höst, et al., Experimentation in Software Engineering, Springer, 2012.
[40] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2009) 131–164.
[41] N. Rios, R. O. Spínola, M. Mendonça, C. Seaman, The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from brazil, Empirical Software Engineering 25 (2020) 3216–3287.
[42] M. Esposito, F. Palagiano, Leveraging large language models for preliminary security risk analysis: A mission-critical case study, in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, ACM, pp. 442–445.
[43] R. Likert, A technique for the measurement of attitudes., Archives of psychology (1932).
[44] G. E. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time series analysis: forecasting and control, John Wiley & Sons, 2015.
[45] P. J. Brockwell, R. A. Davis, Time series: theory and methods, Springer science & business media, 1991.
[46] D. A. Dickey, W. A. Fuller, Distribution of the estimators for autoregressive time series with a unit root, Journal of the American statistical association 74 (1979) 427–431.
[47] R. H. Shumway, D. S. Stoffer, D. S. Stoffer, Time series analysis and its applications, volume 3, Springer, 2000.
[48] W. A. Fuller, Introduction to statistical time series, John Wiley & Sons, 2009.
[49] F. Canova, B. E. Hansen, Are seasonal patterns constant over time? a test for seasonal stability, Journal of Business & Economic Statistics 13 (1995) 237–252.
[50] R. J. Hyndman, Y. Khandakar, Automatic time series forecasting: the forecast package for r, Journal of statistical software 27 (2008) 1–22.
[51] X. Wang, K. Smith, R. Hyndman, Characteristic-based clustering for time series data, Data mining and knowledge Discovery 13 (2006) 335–364.
[52] H. Akaike, A new look at the statistical model identification, IEEE transactions on automatic control 19 (1974) 716–723.
[53] A. D. McQuarrie, C.-L. Tsai, Regression and time series model selection, World Scientific, 1998.
[54] M. Mathioudaki, D. Tsoukalas, M. Siavvas, D. Kehagias, Technical debt forecasting based on deep learning techniques, in: Computational Science and Its Applications – ICCSA 2021, pp. 306–322.
[55] L. Aversano, M. Bernardi, M. Cimitile, M. Iammarino, D. Montano, Forecasting technical debt evolution in software systems: an empirical study, Frontiers of Computer Science 17 (2022).
[56] R. J. Freund, W. J. Wilson, P. Sa, Regression analysis, Elsevier, 2006.
[57] A. Bordes, L. Bottou, P. Gallinari, Sgd-qn: Careful quasi-newton stochastic gradient descent, Journal of Machine Learning Research 10 (2009) 1737–1754.
[58] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society Series B: Statistical Methodology 58

(1996).
[59] A. E. Hoerl, R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, Technometrics 12 (1970) 55–67.
[60] L. De Raedt, P. Flach, Machine Learning: ECML 2001: 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001. Proceedings, volume 2167, Springer, 2003.
[61] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794.
[62] L. Breiman, Random forests, Machine learning 45 (2001) 5–32.
[63] M. Stone, Cross-validatory choice and assessment of statistical predictions, Journal of the royal statistical society: Series B (Methodological) 36 (1974) 111–133.
[64] F. F. Reichheld, The one number you need to grow, Harvard Business Review 81 (2003) 46–55.
[65] A. Straus, Techniques and procedures for developing grounded theory, Basics of Qualitative Research. (1998) 265–274.
[66] C. Seaman, Y. Guo, Measuring and monitoring technical debt advances in computers, 2011.
[67] P. Flach, Machine learning: the art and science of algorithms that make sense of data, Cambridge university press, 2012.
[68] G. H. Golub, C. F. Van Loan, Matrix computations, JHU press, 2013.
[69] M. Esposito, S. Moreschini, V. Lenarduzzi, D. Hästbacka, D. Falessi, Can we trust the default vulnerabilities severity?, in: L. Moonen, C. D. Newman, A. Gorla (Eds.), 23rd IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2023, Bogotá, Colombia, October 2-3, 2023, IEEE, 2023, pp. 265–270.
[70] V. Lenarduzzi, N. Saarimäki, D. Taibi, The technical debt dataset, in: Conference on Predictive Models and Data Analytics in Software Engineering.
[71] M. Esposito, V. Falaschi, D. Falessi, An extensive comparison of static application security testing tools, in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, EASE 2024, Salerno, Italy, June 18-21, 2024, ACM, 2024, pp. 69–78.
[72] M. Esposito, D. Falessi, Uncovering the hidden risks: The importance of predicting bugginess in untouched methods, in: 2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, pp. 277–282.
[73] P. J. Brockwell, R. A. Davis, M. V. Calder, Introduction to time series and forecasting, volume 2, Springer, 2002.

Appendix

In this Section, we detail parameter tuning of the Box-Jenkins models and how to replicate our study.

Appendix A. Parameter tuning of the Box-Jenkins models

In this section of the appendix we describe the mathematical definitions of the model parameters composing the TSA models included in the work, as well as the summary of the steps to be followed in order to implement model parameter tuning of the *Box-Jenkins* models. The models adopted in this study are multivariate versions of the classic Box-Jenkins model, better known as ARIMA [44], therefore their theoretical background reside on the definition of the ARIMA model. This model is a univariate TSA model composed of the autoregressive component, the moving average component, and the differencing component, as explained below.

Appendix A.1. Autoregression (AR) order (p / P parameter)

The autoregressive process of order p, AR(p), denotes a stationary time series $\{X_t\}_{t\in Z}$ such that

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t \tag{A.1}$$

where $\{\phi_1, \cdots, \phi_p\}$ are fixed autoregression coefficients and $\{\epsilon_t\}$ are independent random noises of mean μ_0 and constant variance σ^2. An AR(p) process is *stationary* if the p roots of $\phi(z) = 1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p$ fall within the random state unit circle [44].

Seasonality adjustment occurs when the autoregressive pattern occurs at multiple lags of the defined periodicity (m). Thus, a seasonal autoregression (SAR) process is defined by

$$X_t = \sum_{i=1}^{p} \Phi_i X_{t-S} + \varepsilon_t \tag{A.2}$$

where Φ is defined as $\Phi(z) = 1 - \Phi_1 z - \Phi_2 z^2 - \cdots - \Phi_P z^P$.

Appendix A.2. Moving Average (MA) order (q / Q parameter)

The moving average process of order q, MA(q) denotes a stationary time series $\{X_t\}_{t\in Z}$ such that

$$X_t = \sum_{k=1}^{q} \theta_k \varepsilon_{t-k} \tag{A.3}$$

where $\epsilon_t \sim N(0, \sigma^2)$ and θ are defined as $\theta(z) = 1 + \theta_1 z + \theta_2 z^2 + \cdots + \theta_q z^q$. Every MA($q$) process is stationary, but there can be two different MA(q) processes with the same autocovariance [73]. We avoid this issue by enforcing invertible MA processes (see details in Section 9).

As before, a seasonality adjustment is performed when the moving average pattern occurs at multiple lags of the defined periodicity (m). Thus, seasonal moving average (SMA) processes are defined by

$$X_t = \sum_{k=1}^{q} \Theta_k \varepsilon_{t-k} \tag{A.4}$$

where Θ is defined as $\Theta(z) = 1 + \Theta_1 z + \Theta_2 z^2 + \cdots + \Theta_Q z^Q$.

Appendix A.3. Differecing order (d / D parameter)

Among the existing approaches to reach stationarity, models such as ARIMA apply differencing repeatedly to the series of data until the differenced observations resemble a realization of stationary time series, required for accurate forecasting [73]. Considering, for instance, an original time series $\{X_t\}$ we can define the first differencing order ($d = 1$) as:

$$Y_t = \nabla X_t = X_t - X_{t-1} \tag{A.5}$$

Hence, the differencing operator or lag-d can be formulated as

$$\nabla_d X_t = X_t - X_{t-d}. \tag{A.6}$$

Moreover, differencing can also be used in cases where the seasonality of the series is addressed. If the series has a seasonal pattern, then d-lag differencing order can remove this pattern. Thus, if $X_t = T_t + S_t + \varepsilon_t$ and S_t had period d, then:

$$\nabla_d X_t = T_t - T_{t-d} + \nabla_d \varepsilon_t \tag{A.7}$$

where S_t stands as the seasonality component at series observation t, T_t is the trend components accordingly, and ε_t is the existing random noise.

Appendix A.4. Model parameter tuning

The combination of the model parameter components explained previously in this appendix results in the following formula, which we provide in its multivariate format by including the $X_{t,i}$ independent variables given the claim of our work supporting this type of models:

$$Y_t = \phi_0 + \sum_{i=1}^{k} \beta_i X_{t,i} + \sum_{j=1}^{p} \phi_j X_{t-j} + \sum_{k=1}^{q} \theta_k \varepsilon_{t-k} + \varepsilon_t \quad \text{(A.8)}$$

where:

- ϕ_0 is a model constant.
- ϕ_j and θ_k are the AR and MA coefficients.
- β_i are the regression coefficients for the $X_{t,i}$ independent variables included after the backward variable selection criteria.
- $\varepsilon_t \sim N(0, \sigma^2)$.

This ARIMAX formula can be easily transformed (i) into an univariate ARIMA model, by removing the independent variable regression component and (ii) into seasonally adjusted models such as the studied SARIMAX by integrating the seasonal components (P, Q, D) into the regression. For each combination of independent variables, this study leveraged the *Auto-arima* algorithm as mentioned in Section 3.5.3, to tune the model parameters. This algorithm automates the autoregression and moving average components to provide optimal forecasting results. Initially, it identifies the optimal differencing (d) order to provide stationarity to the time series. Its stepwise process identifies the best AR and MA combinations, respectively. For each possible model parameter combination, it assesses its estimation capability by measuring the *Akaike Information Criterion* (AIC) [48] and *Bayesian Information Criterion* (BIC) [48], both common measures to assess the relative quality of statistical models. The AIC assesses the model goodness of fit based on the estimated likelihood, while it includes a penalty on the increase of the number of parameters included in the model to prevent overfitting. This could be defined as follows:

$$AIC = 2k - 2ln(\hat{L}) \quad \text{(A.9)}$$

where k is the number of included parameters and L is the estimated likelihood.

Similarly to the AIC, the BIC criterion penalizes the increase of model parameters to prevent model over-fitting, while also accounting for the sample size as a component in the calculated penalty, thus favoring stronger prevention of over-fitted models:

$$BIC = kln(n) - 2ln(\hat{L}) \quad \text{(A.10)}$$

where k is the number of included parameters, n is the sample size and L is the estimated likelihood.

Therefore, the stepwise parameter tuning search is repeated until the model cannot be further improved. Consequently, this process generates a model with the most efficient set of independent variables and model parameters. Table B.18 provides a summary containaining the resulting best ARIMAX and SARIMAX model parameters (p, d, q) and (P, D, Q, m), as selected for each project, thus facilitating the reproducibility of our results. We also provide the table with the best model parameter combinations for models ARIMA+LM and SARIMA+LM univariate models in the published replication package.

Appendix B. Replication of the study

In this section of the appendix, we explain the instructions needed to replicate the work of this study. For that, we will emphasize the needed requirements to execute the replication code, the architecture of the shared code as well as the possible code execution options the reader can opt to if wished. The codes used to run this study were written in Python 3.9. We provide instructions on how to install a virtual environment where to install the necessary dependencies to run the code, these are also provided in a requirements file within the replication package. We used well-known statistics and ML libraries such as *statsmodels*, *scikit-learn* and *scipy* to implement the selected models. Additionally, we used the *pmdarima* library to implement the *Auto-arima* algorithm.

The code is organized and structured in a format that facilitates replication for practitioners. All the Python modules are connected to the *main* module. Therefore, practitioners only need to run this module to replicate our work. Moreover, if practitioners are interested in key parts of the implementation, the *commons* module allows them to activate and deactivate sections of the program execution so that these are skipped. We provide further instructions on the program execution sections and a detailed implementation description in the online appendix.

Table B.18: Summary table containing the resulting best ARIMAX/SARIMAX model parameters (p,d,q) and (P,D,Q,m).

Project	Model	Serialization	p	d	q	P	D	Q	m	AIC	Regressors
archiva	Arimax	monthly	0	0	1				12	777.97	S1213, S1192, RedundantThrowsDeclarationCheck, S00117, S1488, DuplicatedBlocks, S1186, S1132
bcel	Arimax	monthly	1	0	1				12	1194.46	S1213, ModifiersOrderCheck, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S00112, S00108, S1186, S1151, S1132
codec	Arimax	monthly	0	1	1				12	1312.76	ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S00108, S1186, S1151, S1132
collections	Arimax	monthly	0	0	1				12	1634.78	S1213, ModifiersOrderCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151, S1132
cli*	Arimax	monthly	0	0	1				12	1775.66	S1213, ModifiersOrderCheck, S1192, S00117, S134, S1066, S1226, S00112, S1186, S1132
exec*	Arimax	monthly	1	1	0				12	903.43	S1213, ModifiersOrderCheck, S1192, S00117, S1488, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151, S1132
configuration	Arimax	monthly	0	1	1				12	1533.28	ModifiersOrderCheck, RedundantThrowsDeclarationCheck, S00122, MethodCyclomaticComplexity, S00112, S00108
dbcp	Arimax	monthly	1	0	1				12	1352.23	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, DuplicatedBlocks, S1066, S00112, S1155, S00108
hive	Arimax	monthly	0	0	2				12	885.95	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S1488, S1226, S1186, S1151, S1132
httpclient	Arimax	monthly	0	0	1				12	1301.53	ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S134, S1186
httpcore	Arimax	monthly	2	2	1				12	1374.66	S1213, ModifiersOrderCheck, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108
jxpath	Arimax	monthly	0	0	1				12	1592.83	S1213, ModifiersOrderCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S1186, S1151, S1132
santuario	Arimax	monthly	0	0	1				12	1848.2	S1213, S1192, RedundantThrowsDeclarationCheck, DuplicatedBlocks, S134, S1066, S00112, S00108, S1132
validator	Arimax	monthly	1	0	1				12	1473.72	S1213, ModifiersOrderCheck, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, S1226, S00112, S1155, S00108, S1186, S1151, S1132
archiva	Arimax	biweekly	1	0	1				26	1599.0	S1192, S1488, S1226, S1155, S1186
bcel	Arimax	biweekly	1	1	1				26	2459.66	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151, S1132
codec	Arimax	biweekly	1	0	1				26	2815.72	S1213, ModifiersOrderCheck, RedundantThrowsDeclarationCheck, S00122, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S1186
collections	Arimax	biweekly	5	5	1				26	3663.77	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, DuplicatedBlocks, S1066, MethodCyclomaticComplexity, S1155, S1151
cli*	Arimax	biweekly	1	0	0				26	4032.23	ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, DuplicatedBlocks, S134, S1066, S1226, S00112, S1155, S1151, S1132

Continued on next page

Project	Model	Serialization	p	d	q	P	D	Q	m	AIC	Regressors
exec*	Arimax	biweekly	2	0	0				26	1799.48	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S1155, S00108, S1186, S1151, S1132
configuration	Arimax	biweekly	2	0	0				26	3113.02	S1213, RedundantThrowsDeclarationCheck, S00117, S00122, S1488, S134, MethodCyclomaticComplexity, S1226, S00112, S00108, S1186, S1132
dbcp	Arimax	biweekly	2	1	0				26	1728.23	S1213, ModifiersOrderCheck, S1192, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S00112, S1155, S1186, S1151, S1132
hive	Arimax	biweekly	0	0	1				26	1796.37	S1213, S1192, RedundantThrowsDeclarationCheck, S00122, DuplicatedBlocks, S134, S00112, S1155, S00108, S1151, S1132
httpclient	Arimax	biweekly	0	0	1				26	2692.53	S1192, S00117, S1066, MethodCyclomaticComplexity, S00112, S1155, S1132
httpcore	Arimax	biweekly	1	1	1				26	2931.74	S1192, RedundantThrowsDeclarationCheck, S1488, S134, S1066, MethodCyclomaticComplexity, S1186, S1151, S1132
jxpath	Arimax	biweekly	0	3	1				26	3064.28	S1213, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1132
santuario	Arimax	biweekly	0	2	1				26	4123.08	S1213, RedundantThrowsDeclarationCheck, S00117, S134, S00112, S1186, S1151
validator	Arimax	biweekly	1	0	1				26	2981.88	S1213, ModifiersOrderCheck, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, S1226, S00112, S1155, S1186
archiva	Sarimax	monthly	0	1	2	0	0	0	12	761.79	S1192, S00122, S1488, MethodCyclomaticComplexity, S00112, S00108, S1186, S1151
bcel	Sarimax	monthly	1	0	1	0	0	0	12	1194.46	S1213, ModifiersOrderCheck, S1192, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S00112, S00108, S1186, S1151, S1132
codec	Sarimax	monthly	1	0	1	0	1	0	12	1309.26	ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S00108, S1186, S1151, S1132
collections	Sarimax	monthly	2	2	1	0	0	0	12	1635.7	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, S00122, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151
cli*	Sarimax	monthly	0	0	1	0	0	0	12	1775.66	S1213, ModifiersOrderCheck, S1192, S00117, S134, S1066, S1226, S00112, S1186, S1132
exec*	Sarimax	monthly	0	4	1	0	0	0	12	935.31	S1213, MethodCyclomaticComplexity, S00108, S1151, S1132
configuration	Sarimax	monthly	0	1	1	0	0	0	12	1536.79	S1192, RedundantThrowsDeclarationCheck, MethodCyclomaticComplexity, S00108, S1151
dbcp	Sarimax	monthly	1	0	1	0	0	0	12	1323.82	S1213, S1192, S00122, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00112, S1155
hive	Sarimax	monthly	0	0	2	0	0	0	12	885.95	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S1488, S1226, S1186, S1151, S1132
httpclient	Sarimax	monthly	0	0	1	0	0	0	12	1301.53	ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S134, S1186
httpcore	Sarimax	monthly	2	2	1	1	1	0	12	1376.82	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151, S1132
jxpath	Sarimax	monthly	0	0	1	0	0	0	12	1592.83	S1213, ModifiersOrderCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S1186, S1151, S1132
santuario	Sarimax	monthly	0	0	1	0	0	0	12	1848.2	S1213, S1192, RedundantThrowsDeclarationCheck, DuplicatedBlocks, S134, S1066, S00112, S00108, S1132

Continued on next page

Project	Model	Serialization	p	d	q	P	D	Q	m	AIC	Regressors
validator	Sarimax	monthly	1	0	1	0	0	0	12	1460.96	S1213, S00117, S00122, DuplicatedBlocks, S134, S1226, S00112, S1155, S00108, S1132
archiva	Sarimax	biweekly	1	0	1	0	0	0	26	1599.0	S1192, S1488, S1226, S1155, S1186
bcel	Sarimax	biweekly	3	0	1	0	0	0	26	2460.57	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S00117, S00122, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S00112, S1155, S00108, S1186, S1151, S1132
codec	Sarimax	biweekly	1	0	0	1	0	0	26	728.68	S1213, RedundantThrowsDeclarationCheck, S00122, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S1226, S00108
collections	Sarimax	biweekly	4	3	1	0	0	0	26	3693.5	S1213, S1192, DuplicatedBlocks, S134, MethodCyclomaticComplexity, S00112, S00108, S1186
cli*	Sarimax	biweekly	1	1	1	1	0	0	26	4031.03	S00112, S1132
exec*	Sarimax	biweekly	1	0	1	1	0	0	26	1900.19	S1213, ModifiersOrderCheck, S1192, S00117, S1488, DuplicatedBlocks, S134, S1066, MethodCyclomaticComplexity, S1226, S1155, S00108, S1151, S1132
configuration	Sarimax	biweekly	3	1	1	0	0	0	26	3105.48	S1213, RedundantThrowsDeclarationCheck, S00117, MethodCyclomaticComplexity, S1226, S00108, S1186
dbcp	Sarimax	biweekly	2	0	1	0	0	0	26	2703.93	S1213, DuplicatedBlocks, S134, S1155, S00108, S1186
hive	Sarimax	biweekly	0	0	1	0	0	0	26	1796.37	S1213, S1192, RedundantThrowsDeclarationCheck, S00122, DuplicatedBlocks, S134, S00112, S1155, S00108, S1151, S1132
httpclient	Sarimax	biweekly	0	0	1	0	0	0	26	2693.89	RedundantThrowsDeclarationCheck, S00117, MethodCyclomaticComplexity, S00112, S1155, S1151
httpcore	Sarimax	biweekly	1	2	1	0	0	0	26	2921.25	S1213, ModifiersOrderCheck, S1192, RedundantThrowsDeclarationCheck, S1488, DuplicatedBlocks, S134, S1066, S00112, S1155, S00108, S1151
jxpath	Sarimax	biweekly	5	0	1	1	0	0	26	3029.53	S1213, ModifiersOrderCheck, S1192, S00117, S134, S1226, S00112, S1155, S00108, S1186, S1151, S1132
santuario	Sarimax	biweekly	2	0	1	0	0	0	26	4099.67	S1213, RedundantThrowsDeclarationCheck, S00122, S134, S00112, S1155, S00108, S1186, S1132
validator	Sarimax	biweekly	1	0	1	0	0	0	26	3018.51	S1213, ModifiersOrderCheck, S1192, S00117, S00122, DuplicatedBlocks, S134, S1066, S1226, S00112, S00108, S1151

Table B.19: Descriptive statistics from the projects used in the analysis.

Apache projects	Time frame◇	Original data				Monthly data				Biweekly data			
		#	Min/Max *SI*	Mean *SI*	Std *SI*	#	Min/Max *SI*	Mean *SI*	Std *SI*	#	Min/Max *SI*	Mean *SI*	Std *SI*
archiva	05/11-02/12	1,630	0–172,634	71,068	34,803	65	0–116,422	68,924	32,639	136	0-121,632	68,916	31,354
batik	10/00-08/02	941	65,831–175,362	132,111	29,957	21	71,725–174,077	138,100	30,581	44	71,725–174,968	140,551	29,246
bcel	10/01-04/18	291	47,854–59,608	54,304	3,262	173	47,869–58,821	54,348	3,223	394	47,869–58,821	54,426	3,233
beanutils	03/01-10/18	324	1,380–58,250	39,025	14,346	190	1,380–58,142	47,057	12,111	427	1,380–58,142	47,154	11,839
cayenne	01/07-07/08	353	179,813–223,285	205,411	12,555	17	179,813–222,828	203,314	13,481	33	179,813–222,875	203,402	13,344
cocoon	03/03-02/07	2,311	64,810–286,277	183,528	65,634	46	66,640–282,280	181,404	75,407	95	65,370–283,313	179,951	73,980
codec	05/03-05/18	229	2,078–17,707	10,618	4,567	164	2,088–17,365	10,216	4,595	365	2,078–17,707	10,317	4,553
collections	04/01-08/16	729	1,494–70,377	40,864	17,315	155	1,504–68,115	46,948	13,180	349	1,504–67,959	47,377	12,689
*cli**	06/02-03/17	240	0–17,714	8,610	4,770	162	0–17,519	8,281	3,800	365	0–17,519	8,282	3,743
*exec**	08/05-10/17	100	2,126–5,537	3,638	916	136	2,258–5,537	3,908	1,047	303	2,126–5,537	3,876	1,041
*fileupload**	07/02-12/14	88	768–6,107	3,687	1,719	140	788–5,698	4,292	1,315	312	788–5,848	4,289	1,299
*io**	01/02-11/17	396	249–42,328	15,782	8,402	160	249–42,328	18,970	10,420	361	249–42,328	19,134	10,128
*jelly**	02/02-09/17	311	1,209–42,910	24,178	11,563	182	1,209–42,910	40,079	6,850	394	1,209–42,910	40,192	6,437
*jexl**	04/02-01/18	351	3,338–20,048	12,831	5,562	164	4,363–19,994	14,537	4,414	368	3,567–20,040	14,605	4,423
configuration	12/03-09/18	764	4,498–44,634	27,754	10,463	147	5,182–44,634	29,486	9,959	323	4,890–44,634	29,923	9,817
daemon	09/03-11/17	28	2,290–4,168	3,133	652	168	2,290–4,062	3,150	775	362	2,290–4,062	3,148	773
dbcp	04/01-11/16	298	5,364–19,459	13,570	3,506	168	5,364–19,386	13,626	3,445	368	5,364–19,319	13,636	3,376
dbutils	11/03-04/05	63	3,357–7,954	4,935	1,339	129	3,357–6,782	4,671	1,193	288	3,357–6,782	4,696	1,192
digester	05/01-08/17	340	20–31,542	13,420	6,825	179	20–31,525	21,636	9,356	397	20–31,525	21,627	6,700
felix	08/05-07/09	1,149	22,667–283,638	144,966	73,749	44	22,667–283,530	121,831	75,758	88	24,405–265,380	120,187	74,220
hive	09/08-03/15	3,001	73,317–701,914	434,703	201,776	75	73,332–701,210	288,531	187,519	154	73,332–687,974	289,030	186,604
httpclient	12/05-05/17	902	55–85,666	48,009	25,366	114	55–85,666	49,809	26,022	244	55–85,666	49,205	25,701
httpcore	02/05-05/17	666	547–86,264	36,145	24,797	131	547–81,133	47,585	19,596	284	547–83,497	47,663	19,368
jxpath	09/01-05/18	148	23,071–32,029	28,448	2,500	188	24,708–32,029	29,013	1,840	422	23,071–32,029	29,032	1,896
net	04/02-10/18	543	12,516–34,898	23,536	3,347	169	12,706–30,433	24,283	4,163	374	12,706–30,433	24,372	4,088
ognl	05/11-06/18	72	20,410–34,077	27,278	5,615	85	20,437–34,006	26,774	1,855	183	20,444–34,006	26,712	1,717
santuario	10/01-10/18	914	2,201–126,431	71,472	25,716	162	2,201–126,431	80,389	27,220	371	2,201–126,431	82,370	27,046
thrift	05/08-10/12	145	0–21,395	15,965	3,629	42	0–21,026	16,486	4,347	96	0–21,395	16,757	4,057
validator	01/02-10/18	249	39–17,541	17,851	4,090	184	39–17,541	12,726	3,616	407	39–17,541	12,791	3,549
vfs	07/02-07/18	428	4,175–29,340	17,851	7,193	168	4,175–29,340	21,431	6,168	373	4,175–29,340	21,667	5,862
zookeeper	07/14-12/16	58	57,033–64,173	60,068	1,926	24	57,033–64,090	60,116	1,971	52	57,033–64,173	60,150	1,997

#: Number of data points, *SI*: SQALE Index, *: Apache Commons project

◇: The dates are approximations as they slightly differ between the projects in some cases. The date format is *mm/yy*

Table B.20: Resulting and discarded projects from the model training stage. (**Bold**: Projects included in the final results.)

Apache projects	**Time frame**◇	**Monthly data**				**Biweekly data**			
		ARIMAX	**ARIMA+LM**	**SARIMAX**	**SARIMA+LM**	**ARIMAX**	**ARIMA+LM**	**SARIMAX**	**SARIMA+LM**
archiva	05/11-02/12	✓	✓	✓	✓	✓	✓	✓	✓
batik	10/00-08/02	✓	✓	✓	✓				✓
bcel	10/01-04/18	✓	✓	✓	✓	✓	✓	✓	✓
beanutils	03/01-10/18	✓	✓	✓	✓	✓	✓		✓
cayenne	01/07-07/08	✓	✓			✓	✓		
cocoon	03/03-02/07	✓	✓	✓	✓	✓	✓		✓
codec	05/03-05/18	✓	✓	✓	✓	✓	✓	✓	✓
collections	04/01-08/16	✓	✓	✓	✓	✓	✓	✓	✓
cli*	06/02-03/17	✓	✓	✓	✓	✓	✓	✓	✓
exec*	08/05-10/17	✓	✓	✓	✓	✓	✓	✓	✓
*fileupload**	07/02-12/14		✓		✓	✓	✓		✓
*io**	01/02-11/17	✓	✓	✓	✓		✓		✓
*jelly**	02/02-09/17	✓	✓	✓	✓	✓	✓		✓
*jexl**	04/02-01/18			✓	✓	✓	✓	✓	✓
configuration	12/03-09/18	✓	✓	✓	✓	✓	✓	✓	✓
daemon	09/03-11/17	✓	✓		✓	✓	✓	✓	✓
dbcp	04/01-11/16	✓	✓	✓	✓	✓	✓	✓	✓
dbutils	11/03-04/05	✓	✓	✓	✓		✓		✓
digester	05/01-08/17	✓	✓	✓	✓	✓	✓	✗	✓
felix	08/05-07/09	✓	✓		✓	✓	✓	✓	✓
hive	09/08-03/15	✓	✓	✓	✓	✓	✓	✓	✓
httpclient	12/05-05/17	✓	✓	✓	✓	✓	✓	✓	✓
httpcore	02/05-05/17	✓	✓	✓	✓	✓	✓	✓	✓
jxpath	09/01-05/18	✓	✓	✓	✓	✓	✓	✓	✓
net	04/02-10/18	✓	✓	✓	✓	✓	✓		✓
ognl	05/11-06/18	✓		✓		✓	✓	✓	✓
santuario	10/01-10/18	✓	✓	✓	✓	✓	✓	✓	✓
thrift	05/08-10/12	✓	✓				✓		
validator	01/02-10/18	✓	✓	✓	✓	✓	✓	✓	✓
vfs	07/02-07/18	✓	✓	✓	✓		✓	✓	✓
zookeeper	07/14-12/16	✓				✓		✓	

*: Apache Commons project

◇: The dates are approximations as they slightly differ between the projects in some cases. The date format is *mm/yy*

Table B.21: Summary statistics for the aggregated MAPE forecasting results from the ARIMAX model with biweekly time series data

Biweekly periods	Mean	Median	Max	Min	Variance
1	1.44	0.12	17.82	0	13.06
2	1.83	0.28	26.56	0	25.25
3	2.14	0.31	28.83	0	29.38
4	2.96	0.4	47.36	0	73.58
5	3.51	0.47	58.31	0	110.16
6	4.41	0.7	65.47	0	142.34
7	5.09	0.87	70.46	0	171.62
8	5.62	0.83	74.1	0	202.68
9	6.09	0.96	76.85	0	224.2
10	5.46	0.92	78.65	0	216.74
11	3.0	0.73	18.54	0	21.95
12	3.15	0.73	19.81	0	24.43
13	3.29	0.76	20.85	0	26.79
14	3.43	0.82	21.72	0	28.94
15	3.54	0.97	22.46	0	30.69
16	3.69	1.1	23.11	0	32.25
17	3.84	1.22	23.68	0	33.94
18	4.06	1.34	24.19	0	35.83
19	4.01	1.39	24.76	0	37.72
20	3.84	1.3	25.28	0	38.52
21	4.03	1.46	25.75	0.01	40.71
22	4.21	1.5	26.18	0.02	43.03
23	4.39	1.47	26.58	0.04	45.21
24	4.56	1.44	26.95	0.06	47.36
25	4.72	1.39	27.29	0.06	49.5
26	4.88	1.35	27.61	0.06	51.76
27	5.04	1.49	27.9	0.06	53.88
28	4.78	1.48	28.16	0.06	54.26
29	4.88	1.56	28.41	0.06	55.73
30	4.97	1.6	28.63	0.06	56.95
31	5.06	1.57	28.84	0.07	58.04
32	4.83	1.17	29.02	0.07	59.27
33	4.89	1.15	29.2	0.07	59.88
34	4.94	1.12	29.35	0.07	60.4
35	4.98	1.16	29.5	0.08	60.62
36	5.03	1.2	29.63	0.09	60.74
37	5.08	1.25	29.8	0.1	61.25
38	5.33	1.54	29.99	0.11	63.82
39	5.39	1.6	30.21	0.13	64.53
40	5.46	1.67	30.45	0.13	65.33
41	5.53	1.77	30.72	0.13	66.23
42	5.6	1.86	31.0	0.14	67.19
43	5.67	1.94	31.21	0.14	67.99
44	5.74	2.03	31.39	0.14	68.76
45	5.81	2.12	31.56	0.15	69.55
46	5.87	2.21	31.56	0.15	69.85
47	5.93	2.28	31.55	0.15	70.15
48	5.99	2.27	31.54	0.16	70.45
49	6.06	2.28	31.54	0.16	70.79
50	4.91	1.61	23.98	0.17	40.79
51	4.97	1.64	24.08	0.17	41.23
52	5.04	1.68	24.19	0.17	41.69
53	5.1	1.71	24.29	0.18	42.09
54	5.17	1.75	24.39	0.18	42.49
55	5.24	1.79	24.48	0.18	42.87
56	5.32	1.82	24.57	0.19	43.23
57	5.39	1.86	24.66	0.19	43.62
58	5.22	1.82	24.74	0.19	44.95
59	5.5	1.93	24.81	0.2	46.92
60	5.57	1.97	24.89	0.2	47.28
61	5.64	2.01	24.96	0.21	47.62
62	5.91	2.9	25.04	0.21	49.97
63	5.5	2.13	25.11	0.21	49.02
64	5.57	2.21	25.18	0.22	49.36
65	5.64	2.29	25.25	0.22	49.76
66	5.74	2.29	25.32	0.22	53.49
67	5.8	2.36	25.38	0.23	53.94
68	5.87	2.42	25.45	0.23	54.41
69	5.93	2.48	25.51	0.24	54.83
70	5.99	2.55	25.57	0.24	55.35
71	5.91	2.52	25.64	0.24	59.57
72	5.97	2.58	25.7	0.25	60.07

Table B.22: Summary statistics for the aggregated MAPE forecasting results from the SARIMAX model with monthly time series data

Monthly periods	Mean	Median	Max	Min	Variance
1	0.3	0.08	1.52	0	0.18
2	0.34	0.14	1.2	0	0.15
3	0.5	0.25	2.1	0	0.32
4	0.61	0.31	2.72	0	0.51
5	0.73	0.39	3.32	0	0.76
6	1.08	0.52	6.51	0	2.42
7	1.38	0.57	10.37	0	5.36
8	1.61	0.64	13.2	0	8.43
9	1.79	0.73	15.39	0	11.4
10	1.98	0.73	17.43	0	14.37
11	2.15	0.72	19.07	0	17.04
12	2.33	0.72	20.4	0	19.46
13	2.49	0.72	21.5	0	21.67
14	2.63	0.72	22.41	0	23.69
15	2.76	0.72	23.2	0	25.6
16	2.89	0.72	23.89	0	27.21
17	3.03	0.71	24.49	0	28.83
18	3.15	0.71	25.03	0	30.44
19	3.29	0.71	25.61	0	32.2
20	3.44	0.78	26.13	0	33.6
21	3.59	0.76	26.59	0.01	34.88
22	3.73	0.83	27.01	0.02	36.16
23	3.88	0.89	27.4	0.03	37.61
24	4.03	0.96	27.74	0.05	39.06
25	4.17	1.03	28.05	0.08	40.56
26	4.31	1.16	28.34	0.1	42.21
27	4.45	1.41	28.6	0.13	43.8
28	4.11	1.34	28.85	0.16	42.31
29	4.19	1.36	29.07	0.19	43.19
30	4.26	1.39	29.28	0.23	43.87
31	4.31	1.42	29.46	0.27	44.55
32	3.96	1.33	29.63	0.3	43.58
33	4.01	1.31	29.79	0.31	44.0
34	4.06	1.29	29.93	0.32	44.36
35	4.11	1.27	30.06	0.33	44.77
36	4.19	1.28	30.19	0.34	45.35