Towards a Robust Approach to Analyze Time-Dependent Data in Software Engineering

Nyyti Saarimäki,[1] Sergio Moreschini,[1] Francesco Lomio,[1] Rafael Penaloza,[2] Valentina Lenarduzzi[3]
[1]Tampere University, Finland — [2]University of Milano-Bicocca — [3]University of Oulu, Finland
nyyti.saarimaki@tuni.fi, sergio.moreschini@tuni.fi, francesco.lomio@tuni.fi,
rafael.penaloza@unimib.it, valentina.lenarduzzi@oulu.fi

***Abstract*—*Background.* Several recent software engineering studies use data mined from the version control systems adopted by the different software projects. However, inspecting the data and statistical methods used in those studies reveals several problems with the current approach, mainly related to the dependent nature of the data.**
***Objective.* We analyzed time-dependent data in software engineering at commit level, and propose an alternative approach based on time series analysis.**
***Method.* We identified statistical tests designed for time series analysis and propose a technique to model time dependent data, similarly to what is done in finance and weather forecasting. We applied our approach to a small set of projects of different sizes, investigating the behaviour of the SQALE INDEX, in order to highlight the time and interdependency of the different commits.**
***Results.* Using these techniques, we analysed and model the data, showing that it is possible to investigate this type of commit data using methods from time series analysis.**
***Conclusion.* Based on the promising results, we plan to validate the robustness of the approach by replicating previous works.**

***Index Terms*—Data Analysis, Time Series Analysis, Time Dependent Variables, Empirical Methods, Mining Software Repository**

I. INTRODUCTION

In Software Engineering (SE), and especially in Mining Software Repository (MSR), researchers often investigate the relationships among different variables collected from the history of software projects. As an example, researchers have investigated correlations between two variables such as code smells, design smells, and architectural smells [1]–[3], their trend over time [4], and the impact of different software qualities [5]–[7].

However, a large number of these studies have overlooked the limitations of the temporal dependency of the variables, and the possible threats related to the usage of statistical techniques not designed for analyzing temporally dependent data. As an example, the introduction of a code smell in a commit heavily depends on the code that was present in the repository before the commit. Still, most of the studies have not considered this aspect, mainly due to a lack of clear statistical techniques that can be used in this context.

From previously published MSR papers, we can highlight three main issues: 1) discarding the temporal nature of the commits, 2) assumption of independent data, and 3) the proportional size of projects, where big projects overwhelm small ones. To overcome these issues, we propose identifying a robust approach to analyze dependent data in software engineering considering the dependency and the time effect.

We conceptualize our approach based on time-series analysis techniques [8] adopted in finance [9], [10] and weather forecasting analysis [11] to predict the behavior of a variable based on its previous observations.

We define a roadmap aimed at defining a new data analysis approach for dependent data. In this paper, we provide a small example demonstrating the problems and showing that even a simple time series model is able to fit well to commit data.

Paper Structure. Section II introduces the motivation behind this work. Section III describes the techniques for a new approach, while Section IV presents the achieved preliminary results and Section V highlights the future benefits of a new approach. Section VI depicts our roadmap. Finally, conclusion are presented in Section VII.

II. ISSUES WITH THE CURRENT APPROACH

Several recent papers utilize the possibility to mine data from the projects' version control systems. Specially MSR studies use the individual commits and releases from projects to investigate the different qualities of the source code; see, e.g. [4]–[7], [12], [13].

These studies frequently use the structure presented in Figure 1 for data collection and analysis. The dataset is constructed by selecting a group of projects and extracting a set of commits from their version control systems. In some papers the analyzed dataset is constructed by pooling all of the selected commits together.

However, when inspecting the approach more closely, three main issues emerge caused by the nature of the data and the prerequisites of the commonly used statistical methods.

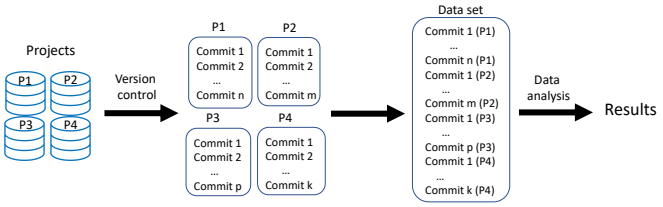

Fig. 1: A **frequent** structure for data collection and analysis in an MSR paper

1) *Discarding the temporal nature of commits*. Commits within a project are ordered and the newer ones are built on the previous ones. Therefore, they depend on each other, forming a time series. This temporal information is completely lost when the commits are grouped together.
2) *The assumption of independent data.* Several commonly applied statistical methods such as Mann-Whitney and one way ANOVA assume that the compared groups are independent of each other. When the comparison groups are sampled from the pooled dataset, both groups are likely to contain commits from the same projects. Therefore, the data in different groups most likely is not completely independent and the results of the data analysis phase could be inaccurate.
3) *Big projects overwhelm small ones.* To assure representativity, the used data sets commonly contain both big and small projects, a practice also instructed by the guideline for selecting a diverse set of projects for an MSR study [14]. A big project can contain tens of thousands of commits while a small one only hundreds. As stated before, in a project the commits build on each other and thus large parts of the code can be the same in several commits. As a consequence, issues and metrics are easily very similar between commits of a project. Therefore, if a dataset contains several big projects, the commits of the smaller ones might not have an impact on the results.

Looking at the methodologies adopted in the MSR, we found that only a limited number of papers adopt some temporal dependent data analysis techniques [15], [16], such as Markov chains [17] or time series analysis [18], [19]. Time series analysis was adopted to forecast future releases in correspondence with which refactoring activities should be performed [18], [19]. These two works are based on the same idea as ours, but the analysis is limited to univariate or bivariate approaches.

Markov chains were applied to understand the evolution of one variable (such as anti-patterns or code smells) [17]. However, while taking into account the temporal evolution of a variable, Markov chains make strong assumptions of the irrelevance of the other measures to the analysis [20].

III. A Robust Approach

A. Steps for a Robust Approach

As we have seen, limiting to static analysis of variables neglecting the influence of time disregards some important intra- and inter-variable dependencies which only become obvious over time. Simply put, today's bad coding practices translate into tomorrow's bugs. We thus believe that *time*—or at least synchronization points—should be considered for the analysis of dependent variables.

In the following, we provide a step-wise description of the process to construct the approach.

Step 1: Identification of timed-data analysis techniques. Several different techniques exist and they depend on the specific properties of the data at hand. In this paper, we describe time series analysis methods. However, these are not the only potentially suitable methods.

Step 2: Applicability of the identified techniques to SE. Several methods exist for analyzing the evolution of different number of variables over time. For example, with one variable it is possible to use an auto regressive (AR) or moving average (MA) model or more advanced model which combine these, like an autoregressive moving average (ARMA) model. In comparison to the current approaches used in SE, time series analysis solves the issues we have described before; namely, it considers the knowledge at each time point, the time sequence, and the local dependencies between variables.

Step 3: Approach validation. Our aim is to evaluate the proposed methodology by selecting a set of published SE studies. We will replicate these studies by applying our methodology and comparing the achieved results. The outcome of the analysis will be discussed with experts in SE to evaluate the validity of the methodology.

B. Techniques for a Robust Approach

We propose using time series analysis methods that utilize the time dependency of the data and fit the SE context.

A time series assumes that the current state of a system depends on the states of the last n time points (this n is often called the *lag*), and tries to understand this dependency [8]. This is also true in SE; specifically, if we talk about data from version control systems (e.g., commit information). Each commit, in fact, depends *at least* on the previous status of the system, and therefore on the previous commits. It might also be the case in which a commit depends on other external factors, which therefore need to be analysed further (multivariate analysis).

The analysis of this dependency from timed data has important applications in different domains; most notably in finance [9], [10] and weather analysis [21], where the goal is to *forecast* [11] the future behavior of a variable based on its previous performance (and that of other relevant variables). The fundamental technique used in these domains is statistical *time series analysis*: robust statistical methods designed for describing time dependencies accurately, considering also the presence of noise and natural variations.

One of the most important concepts in time series analysis methods is the *stationarity* of the data. Stationarity indicates that the statistical properties of the time series, like its mean and variance, are the same regardless of the time of observation. This makes the series more predictable. For example, series which constantly grow or have a seasonal behaviour are not stationary. When the data is non-stationary, it is often possible to remove trends (and regain stationarity) by differencing [22]. Data stationarity can usually be verified using statistical tests like *Augmented Dickey-Fuller* (ADF) [23], [24], or Phillips-Perron test [25].

Another useful concept is the one of *autocorrelation*, which presents the correlation between a variable at time t, i.e. $Y(t)$, and its previous (lagged) value $Y(t-k)$. As with normal

correlation, the value of autocorrelation varies from -1 to 1. In an autocorrelation plot the values are visualized for several lag values. Similarly, the *partial autocorrelation* also measures the correlation between $Y(t)$ and $Y(t-k)$ but the effect of the values between $Y(t)$ and $Y(t-k)$ are removed. Both autocorrelation and partial autocorrelation are used in this study to analyze the data and understand its key characteristics.

IV. Preliminary validation

This section illustrates in practice the problems highlighted in Section II and demonstrates how a simple time series model is able to model SE data.

To allow the replicability of the presented results, we published the complete raw data together with the instructions for running the analysis in the replication package.[1]

A. Data

A preliminary analysis was conducted using the Technical Debt Dataset [26] (version 2.0). The dataset contains 31 projects from Apache Software Foundation. We present results only for three projects; the smallest project (Zookeeper, 222 commits), the median-size project (HTTP Core, 1,901 commits), and the largest project (Cocoon, 10,206 commits) in terms of number of commits. The time series are formed by the commits not the actual times of the commits. The results for all projects are provided in the replication package.

The analysis is done using only one variable, SQALE INDEX, to highlight the time and inter dependency of the commits. The SQALE INDEX is a metric measured by SonarQube and their documentation defines it as the "effort to fix all Code Smells in minutes.[2]" Figure 2 depicts the development of the SQALE INDEX for the selected projects over their evolution.

B. Analysis

In order to check the fitness of the data with the time series modelling, we first considered the stationarity of the data through a Philips-Perron test. The result of the test suggested non-stationarity, and therefore data had to be differenced in order to make it stationary. For this reason, we chose a model able to differenciate the original time series throughout the analysis to handle non-stationarity (see Section IV-C).

After this step, to determine whether the values of SQALE INDEX correlate between the commits, we calculated the autocorrelation and partial autocorrelation. The plots in Figure 3 show the correlation and autocorrelation up to lag 30 for the selected projects. For all three projects, the autocorrelation is high: especially for Cocoon and HTTP Core it is almost 1 for all calculated lag values. This suggests that the values of SQALE INDEX in consequent commits strongly depend on each other. To corroborate this, the partial autocorrelation suggests that the commits are not autocorrelated if the information between the commits is not considered. Therefore, significant amount of information is lost if the time dependency of the data is not utilized in the data analysis.

[1]https://figshare.com/s/2ddb18e02700c20637e5

[2]https://docs.sonarqube.org/latest/user-guide/metric-definitions/

C. Modelling

Given the type of data with which we are dealing, and its non-stationarity, we decided to fit the data using an Autoregressive Integrated Moving Average (ARIMA) model. We use this for the selected project to show its ability to predict the development of the SQALE INDEX. The ARIMA model, as the name states, is composed of three different parts: the *autoregressive* (AR) part, which is used to regress the target variable (SQALE INDEX) based on its past values; the *integrated* (I) part, which is used to apply a differencing step on the data to make it stationary; and the *moving average* (MA) part, which calculates the moving average of the past regression errors, and helps to adjust the model accordingly.

Figure 4 depicts the actual values of the SQALE INDEX and the predictions obtained with the ARIMA model. The performance of the model was measured using Mean Absolute Percentage Error (MAPE). The MAPE values for all three projects are less than 1.5% which suggests a good fitting of the original data. This test was performed using a basic model, and no hyperparameter tuning nor other model selection was done: it does serve though to further corroborate our hypotheses on the need to utilise time series specific tools to analyse data from version control systems.

V. A Possible Future

The example analysis presented in this paper should give further proof of the existence of the issues in the current frequently used data analysis approach. Even if the analysis conducted in this paper does not suggest a new approach, robust time series analysis show potential for understanding the dependencies of different project parameters and their evolution over time. An initial proposal for a revised data analysis procedure is visualized in Figure 5.

The proposal is to analyze each project separately using a time series method which deals with the issues with data dependency and temporal nature and combine the results from the separate analysis which addresses the issues of large projects overwhelming the small ones.

The use of time series analysis allows us to foresee possible improvements and extensions to our framework, based on existing statistical services. First and foremost, we can forecast the evolution of some variables, and evaluate the influence of small changes on the dependent variables over time; that is, their sensitivity to other variables. We are planning to study the life-cycles of software projects, which is noticeable distinct from other temporal phenomena, and formalize its phases (growth, stagnation, decay, acceleration, etc.) according to this evolution. This information will help us to compare different projects (among the same developers, and between groups), and more importantly, identify general behaviors which might teach us something new about the development process.

Among the multiple approaches available to the problem, we plan to further investigate the use of time series analysis tools, including but not limited to the ARIMA model already showed in this study, filters used in signal processing, and *stochastic processes*. The two latter in particular might be

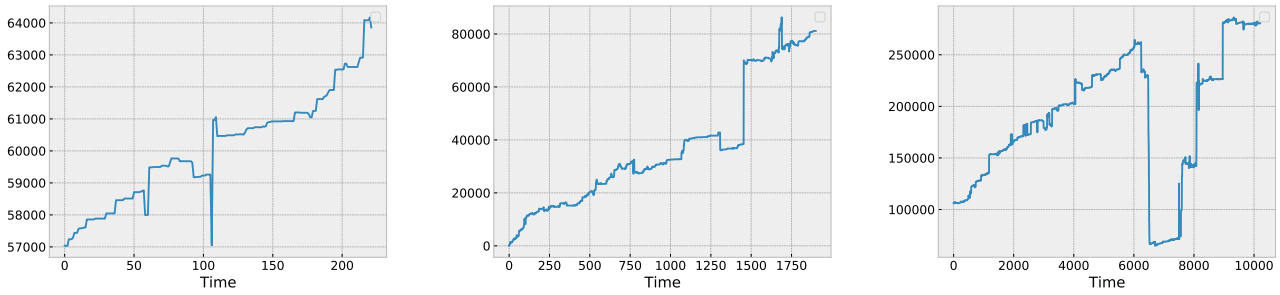

Fig. 2: The development of SQALE INDEX for Zookeeper (left), HTTP Core, and Cocoon (right) over evolution of the projects.

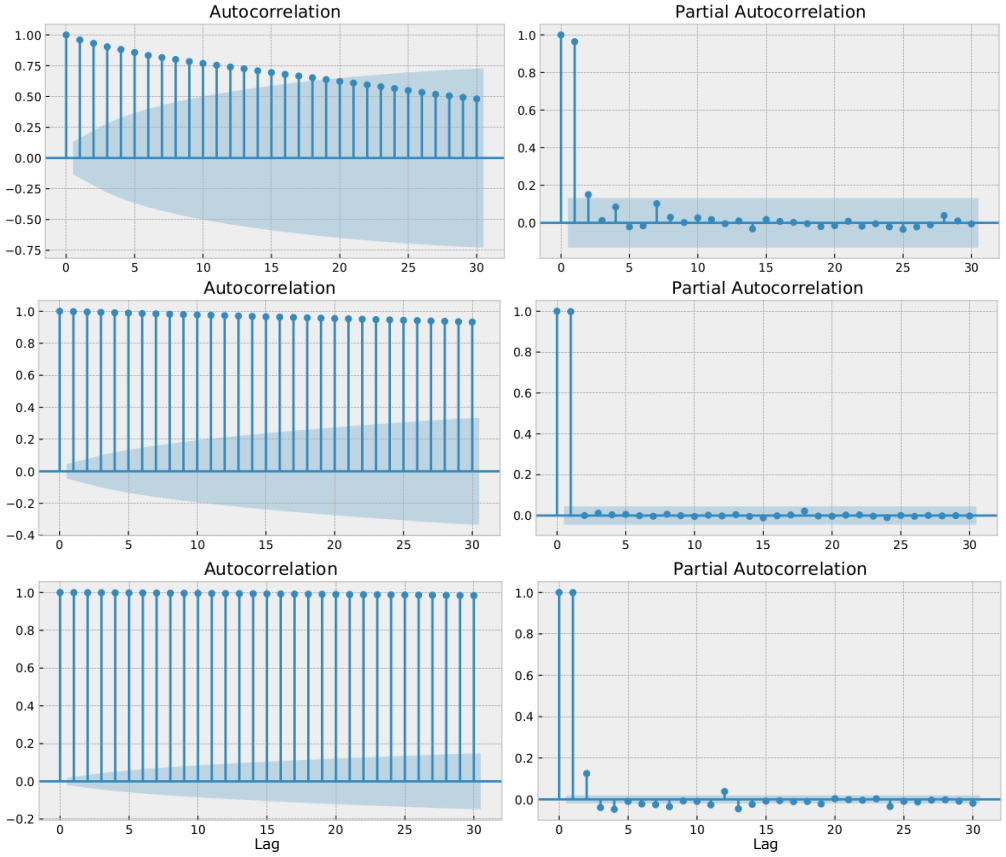

Fig. 3: Autocorrelation and partial autocorrelation for Zookeeper (top), HTTP Core (middle), and Cocoon (bottom).

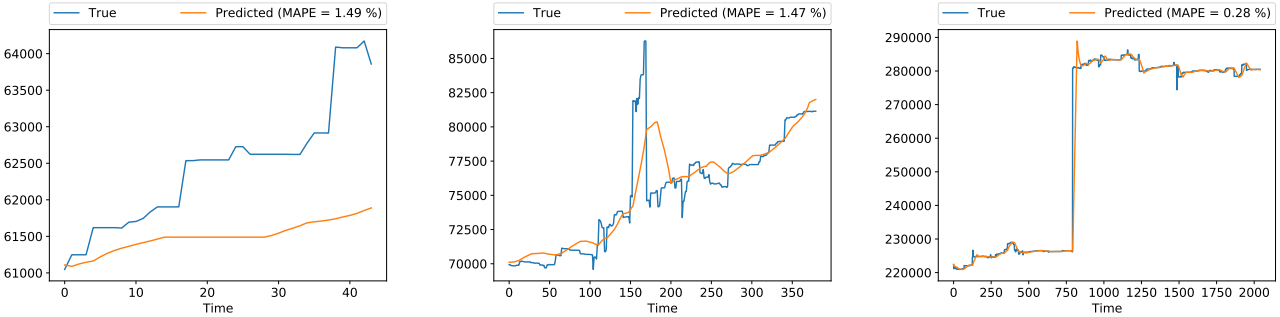

Fig. 4: ARIMA model for Zookeeper (left), HTTP Core (middle), and Cocoon (right) predicting the value of SQALE INDEX.

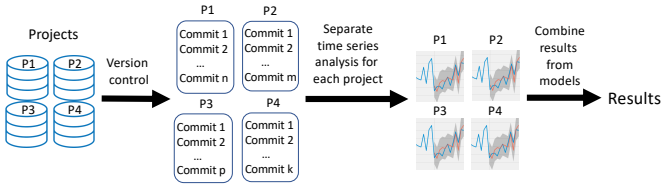

Fig. 5: The proposed data analysis approach.

useful for analyzing the commit data as time series: they have been already proven useful in addressing similar issues in the financial field, as they can take into account multiple components (including a random and noisy component) which might be found also in this data.

We note that, as stated in Section II, none of these tasks can be performed through the existing techniques which disregard the temporal component of the data points. Moreover, our approach provides more fine-grained results, allowing for subtle dependencies. We thus expect our approach to outperform existing methods after additional development, which is already ongoing.

VI. The Roadmap

To adopt this new approach for the data analysis methodology protocol, it is necessary to further develop the foundations established in Section III. Our proposed roadmap includes the following steps:

1) Establish an approach to analyze dependent data in SE.
2) Compare the approach with the traditional ones adopted by SE studies.
3) Improve the proposed approach by applying it to previous SE studies and comparing the obtained results.
4) Elaborate a protocol on how to apply this approach.

VII. Conclusions

In this paper, we conceptualize the problems and a new approach to analyze dependent data in software engineering taking into consideration the time effect. We applied time-series analysis techniques successfully adopted in other domains (e.g. finance) to predict the behavior of a variable based on previous observations of a set of variables.

In order to validate our approach, we define a roadmap that includes four steps. We already started to apply our approach to a small set of projects investigating the relationship between two variables calculated at the commit level applying two different models.

As future work, we are planning to validate the robustness of the approach by replicating previous works.

References

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

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

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

[4] S. Olbrich, D. S. Cruzes, V. Basili, and 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*, 2009, pp. 390–400.

[5] D. I. K. Sjoberg, A. Yamashita, B. Anda, A. Mockus, and T. Dyba, "Quantifying the effect of code smells on maintenance effort," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1144–1156, aug 2013.

[6] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. D. Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, Jun 2018.

[7] V. Lenarduzzi, N. Saarimäki, and 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*, p. 110750, 2020.

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

[9] R. S. Tsay, *Analysis of financial time series*. John wiley & sons, 2005.

[10] J. Arroyo, R. Espínola, and C. Maté, "Different approaches to forecast interval time series: a comparison in finance," *Computational Economics*, vol. 37, no. 2, pp. 169–191, 2011.

[11] C. Chatfield, *Time-series forecasting*. CRC press, 2000.

[12] S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjoberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *IEEE International Conference on Software Maintenance*, ser. ICSM '10, 2010, pp. 1–10.

[13] A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou, and P. Avgeriou, "The effect of gof design patterns on stability: A case study," *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 781–802, 2015.

[14] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Joint meeting on foundations of software engineering*, 2013, pp. 466–476.

[15] C. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 493–509, 1999.

[16] G. Antoniol, V. F. Rollo, and G. Venturi, "Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories," in *Proceedings of the 2005 international workshop on Mining software repositories*, 2005, pp. 1–5.

[17] F. Jaafar, F. Khomh, Y. Guéhéneuc, and M. Zulkernine, "Anti-pattern mutations and fault-proneness," in *International Conference on Quality Software*, 2014, pp. 246–255.

[18] G. Antoniol, M. Di Penta, and E. Merlo, "Predicting refactoring activities via time series," in *First International Workshop on Refactoring (REFACE)*, 12 2003.

[19] H. Wang, M. Kessentini, W. Grosky, and H. Meddeb, "On the use of time series and search based software engineering for refactoring recommendation," in *International Conference on Management of Computational and Collective IntElligence in Digital EcoSystems*, 2015, p. 35–42.

[20] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[21] S. D. Campbell and F. X. Diebold, "Weather forecasting for weather derivatives," *Journal of the American Statistical Association*, vol. 100, no. 469, pp. 6–16, 2005.

[22] A. Sutcliffe, "Time-series forecasting using fractional differencing," *Journal of Forecasting*, vol. 13, no. 4, pp. 383–393, 1994.

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

[24] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented dickey–fuller test," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 277–280, 1995.

[25] P. C. Phillips and P. Perron, "Testing for a unit root in time series regression," *Biometrika*, vol. 75, no. 2, pp. 335–346, 1988.

[26] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Conference on Predictive Models and Data Analytics in Software Engineering*, 2019.