Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 83–92
DOI: 10.15439/2023F5474
ISSN 2300-5963 ACSIS, Vol. 35

Defect Backlog Size Prediction for Open-Source Projects with the Autoregressive Moving Average and Exponential Smoothing Models

Paulina Anioła (Sielicka)
Email: paulina.aniola23@gmail.com

Sushant Kumar Pandey, Miroslaw Staron
0000-0003-1882-2435
0000-0002-9052-0864
Dept. of CSE Chalmers | University of Gothenburg, Sweden

Mirosław Ochodek
0000-0002-9103-717X
Poznan University of Technology,
ul. Piotrowo 2, 60-695 Poznan, Poland
Email: miroslaw.ochodek@put.poznan.pl

***Abstract*—Context: predicting the number of defects in a defect backlog in a given time horizon can help allocate project resources and organize software development. Goal: to compare the accuracy of three defect backlog prediction methods in the context of large open-source (OSS) projects, i.e., ARIMA, Exponential Smoothing (ETS), and the state-of-the-art method developed at Ericsson AB (MS). Method: we perform a simulation study on a sample of 20 open-source projects to compare the prediction accuracy of the methods. Also, we use the Naïve prediction method as a baseline for sanity check. We use statistical inference tests and effect size coefficients to compare the prediction errors. Results: ARIMA, ETS, and MS were more accurate than the Naïve method. Also, the prediction errors were statistically lower for ETS than for MS (however, the effect size was negligible). Conclusions: ETS seems slightly more accurate than MS when predicting defect backlog size of OSS projects.**

I. Introduction

DEFECT backlog is the collection of all project defect reports that need to be handled. The size of this collection changes over time. The problem of monitoring defect backlogs is important in all modern software development organizations. In agile software development, it is important to correctly prioritize defects to continuously deliver business value. Also, especially in large organizations, the assignment of developers and testers to projects is often done dynamically, on demand. When a situation in a project demands more human resources for quality improvements, developers shift their focus from feature implementations to defect removal [2]. Because of these dynamic changes, knowing in advance that a project may require more human resources in the following week is valuable information for the managers, developers, testers, and other project stakeholders.

In particular, the managers need to know defect backlogs for the coming weeks. Therefore defect prediction models that forecast the number of defects that will need to be handled in a given time horizon are needed. There have been multiple studies on designing such models in industrial contexts [3], [4], [5], [6]. One of the successful studies on defect backlog prediction was conducted at Ericsson by Staron and Meding [3]. They proposed an autoregressive model (MS) that is based on the moving average and predicts defect backlog size within a weekly horizon. Although the MS model turned out to be very accurate at Ericsson, there are several other state-of-the-art autoregressive models for time series forecasting that have not been tried out for defect-backlog predictions. Two of them are Autoregressive integrated moving average (ARIMA) [7] and Exponential Smoothing (ETS) [8], [9], which are the two most widely used approaches to time series forecasting [10].

Although most of the previous studies on defect prediction were based on open-source software (OSS) datasets, the studies on defect backlog predictions in Ericsson have not been replicated in the OSS context. The flow of OSS projects differs from the flow of their industrial counterparts, however, they are often larger in terms of the number of involved contributors. Predicting the number of defects in a backlog is not easy due to uncertainties in identifying all the defects. The dynamic nature of software development, with its changing requirements and iterative cycles, adds to the complexity. Moreover, the accuracy of predictions can be affected by the quality and relevance of historical data used for analysis. Most existing methods rely on data from classical repositories like NASA and PROMISE, which may have limitations. Lastly, current predictive models may not consider all the contextual factors and unique project characteristics that impact defect discovery.

The goal of this study is to design defect-backlog prediction models based on the ARIMA and ETS methods and validate their accuracy in the context of large OSS projects. We use the state-of-the-art MS model developed at Ericsson [3] as a baseline for comparison since it has been reported as an accurate defect backlog prediction model validated in an industrial setting.

The structure of this paper is as follows. Section II provides a brief overview of the ARIMA and ETS methods, while Section III discusses the related work. Section IV describes the research methodology of our study. The results are presented

This work was supported by the Poznan University of Technology within the project 0311/SBAD/0738. Some of the paper's contents come from the corresponding author's master thesis [1].

and discussed in Section V. Finally, Section VI summarizes the main findings of our study.

II. Background

A. Defect Backlog

In many software projects, defects that need to be resolved are collected in defect backlogs. There are many defect-tracking tools on the market (e.g., Bugzilla, Jira, ReQtest). This kind of software helps the entire team and managers to get a view of how many defects remain in the software and what they are. To help developers work on the project, the defects in the backlog can be ordered according to priority. Often each issue includes additional information which differs between projects. If the software is regularly tested the size of the defect backlog changes over time. The number of defects that have been reported in a specific period of time is called *defect inflow*. Similarly, the number of defects that have been resolved in that period is referred to as *defect outflow*. The defect backlog size change within a given period of time (for the sake of this study, a week) is the difference between its inflow and outflow.

B. Autoregressive Integrated Moving Average

ARIMA stands for Autoregressive Integrated Moving Average. As the name suggests, it combines two time-series techniques, namely, the Autoregressive model and Moving Average.

ARIMA requires the time series to be stationary. The values of stationary time series do not depend on time. Thus, if we can see a trend or seasonality in time series, it means that it is non-stationary—its value depends on the time. Non-stationary time series have to be first transformed into stationary time series by using the differencing operation. The differenced time series is calculated as changes between subsequent observations [10]—see Equation 1. The differencing operation can be repeated multiple times if the obtained time series is still non-stationary.

$$y_t^{'} = y_t - y_{t-1} \tag{1}$$

where:
$y_t^{'}$ - value of the differenced series at time t,
y_t - value of the original series at time t,
y_{t-1} - value of the original series at time $t-1$.

The Autoregressive model is based on multiple linear regression. What distinguished it from other linear regression models is that it predicts the outcome variable (y) using past values as predictor variables (x). This approach assumes that there is some correlation between subsequent values in a time series (autocorrelation). The Autoregressive model is defined by Equation 2 [10].

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{2}$$

where:
c - constant value,
ϕ - model parameter,
ϵ_t - error,
p - order of the model.

The p value in Equation 2 is called *the order* of the Autoregressive model. It determines how many past values will be considered to calculate the outcome. The autoregressive model of order p can be referred to as AR(p).

The Moving Average model calculates the outcome variable as a linear combination of past forecast errors. The formula of the model is presented in Equation 3 [10].

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} \tag{3}$$

where:
c - constant value,
θ - model parameter,
ϵ - forecast error,
q - order of the model.

It shows that the value of y_t can be considered as a weighted moving average of the past forecast errors. The model order value q determines how many past forecast errors will influence the outcome. The moving average model of order q can be referenced as MA(q).

The equation of a non-seasonal ARIMA model presented in Equation 4 shows that it combines components of autoregressive and moving average models, which are lagged values and lagged errors.

$$y_t^{'} = c + \phi_1 y_{t-1}^{'} + ... + \phi_p y_{t-p}^{'} + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t \tag{4}$$

where:
$y_t^{'}$ - differenced series.
The outcome of the model is a differenced series. To get the actual predicted values time series need to be integrated. Integrating is the reverse of differencing. The transformation aims to add the trend or seasonality which were previously removed.

The non-seasonal ARIMA model is characterized by 3 parameters:

- p - order of autoregression part,
- d - degree of involved differencing,
- q - order of moving average part.

The model of some specific parameters can be referenced as ARIMA(p, d, q).

We use the ARIMA implementation provided by the R forecast package [11]. The function `auto.arima()` estimates the model parameters by analyzing the training data.

C. Exponential Smoothing

The general idea behind Exponential Smoothing (ETS) forecasting methods is that predicted values are weighted averages of past observations. The weight which is associated with the observation depends on how old the observation is. Thus, the oldest observations will have a smaller impact on the outcome than the recent ones.

The simplest version of the exponential smoothing method, called Simple Exponential Smoothing, is expressed by Equa-

tion 5 [10]. The application of this version of the method is limited to the data with no clear trend or seasonality.

$$\hat{y}_{T+1} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + ... \quad (5)$$

where:

- $0 \geq \alpha \leq 1$ is smoothing parameter.

The smoothing parameter regulates how the weights change with the change in the distance of observation. If α is small, more weight is given to the observations from the past. If it is large, more weight is associated with the recent observations.

Equation 5 [10] can be described in the component form as presented in Equation 6.

$$\begin{aligned} \text{Forecast Equation} \quad & \hat{y}_{t+h} = l_t \\ \text{Smoothing Equation} \quad & l_t = \alpha y_t + (1-\alpha) l_{t-1} \end{aligned} \quad (6)$$

where:

- h - number of steps to forecast,
- l - level component.

A single component is called level (smoothed value) l_t of the series at time t. From the forecast equation, we can see that the predicted value at time $t+1$ is the level of the time series at time t. Replacing the level component in the smoothing equation according to the relation $\hat{y}_{t+h} = l_t$ leads to the exponential smoothing form presented in Equation 5 [10].

To extend the application of the simple exponential smoothing method for data with trend, an additional component has been added to the equations. The extended method's name is Holt's linear trend method and is expressed by 3 equations presented in Equation 7 [10].

$$\begin{aligned} \text{Forecast Equation} \quad & \hat{y}_{t+h} = l_t + hb_t \\ \text{Level Equation} \quad & l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ \text{Trend Equation} \quad & b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \end{aligned} \quad (7)$$

where:

- b - the estimate of the trend,
- β^* - smoothing parameter for the trend, $0 \geq \beta^* \leq 1$.

The trend (slope) forecast function is no longer flat as it was in the case of Simple Exponential Smoothing. However, this method is still not especially useful because of the fact that the trend is constant. The method assumes that the outcome values always increase or decrease in the same way. Thus, an additional parameter called damping parameter has been introduced to deal with that. Because of this modification, the trend can be flattened in the future. The form of the method which includes the damping parameter is expressed by Equation 8. As we can see with damping parameter $\phi = 1$, the method is the same as Holt's linear method presented in Formula 7.

$$\begin{aligned} \text{Forecast Equation} \quad & \hat{y}_{t+h} = l_t + (\phi + \phi^2 + ... + \phi^h)b_t \\ \text{Level Equation} \quad & l_t = \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1}) \\ \text{Trend Equation} \quad & b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)\phi b_{t-1} \end{aligned} \quad (8)$$

where:

- ϕ - damping parameter, $0 \geq \phi \leq 1$.

Holt's method can be extended with the seasonal component. This version is called Holt-Winters' seasonal method. There are two versions of the method: additive and multiplicative. The additive method is suitable for the series with constant seasonal variations. On the other hand, the multiplicative method is preferred when the variations change in proportion to the series. The component form of Holt-Winters' additive method is expressed by Formula 9 [10].

$$\begin{aligned} \text{Forecast Equation} \quad & \hat{y}_{t+h} = l_t + hb_t + s_{t+h-m(k+1)} \\ \text{Level Equation} \quad & l_t = \alpha(y_t - s_{t-m}) \\ & \quad + (1-\alpha)(l_{t-1} + b_{t-1}) \\ \text{Trend Equation} \quad & b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\ \text{Seasonal Equation} \quad & s_t = \gamma(y_t - l_{t-1} - b_{t-1}) \\ & \quad + (1-\gamma)s_{t-m} \end{aligned} \quad (9)$$

where:

- m - number of seasons in a year,
- k - integer part of $(h-1)/m$,
- γ - smoothing parameter for the seasonality, $0 \geq \gamma \leq 1-\alpha$.

The component form of Holt's-Winters' multiplicative method is expressed by formulas 10.

$$\begin{aligned} \text{Forecast Equation} \quad & \hat{y}_{t+h} = l_t + hb_t + s_{t+h-m(k+1)} \\ \text{Level Equation} \quad & l_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1}) \\ \text{Trend Equation} \quad & b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\ \text{Seasonal Equation} \quad & s_t = \gamma\frac{y_t}{l_{t-1} - b_{t-1}} + (1-\gamma)s_{t-m} \end{aligned} \quad (10)$$

The difference between those two versions of Holt-Winters' method is how they express the seasonal component and then take it into account. In the additive method, it is expressed in absolute terms and then is seasonally subtracted from the series. In contrast to this in the multiplicative method, the seasonal component is expressed in relative terms and then the series is seasonally divided by it.

There are 9 different exponential smoothing methods. Those presented so far are examples of different combinations of components. Each method is defined by the type of trend and seasonal components.

The types of trend components are:

- None (N),
- Additive (A),
- Additive damped (A_d) .

The types of seasonal components are:

- None (N),
- Additive (A_d),
- Multiplicative (M).

Each exponential smoothing method can be labeled with two letters which refer to the type of trend and seasonal components. Table I presents the classification of exponential smoothing methods.

TABLE I: Classification of exponential smoothing methods.

Trend component	Seasonal Component		
	None (N)	Additive (A)	Multiplicative (M)
None (N)	(N, N)	(N, A)	(N, M)
Additive (A)	(A, N)	(A, A)	(A, M)
Additive damped (A_d)	(A_d, N)	(A_d, A)	(A_d, M)

For each of the 9 presented methods, there are two models differing in the way of expressing the errors. The first model with additive errors and the second one with multiplicative errors. For each method, the forecast points of two different models are the same. However, they generate different prediction intervals. To make the distinction the classification in Table I is extended by the third letter. Every exponential smoothing model is labeled with three letters as ETS(Error, Trend, Seasonal). Thus, the model that includes additive error, none trend component, and multiplicative seasonal component would be denoted as ETS(A, N, M).

We use the ETS implementation provided by the R forecast package [11]. The function `ets()` estimates the model parameters by analyzing the training data.

III. Related Work

A. Software Reliability Growth Models

Software Reliability Growth Models are equations used to model the growth of software reliability using defect inflow data gathered during the development process. Researchers use SRGMs to make defects forecasting. The side effect of predicting defects themselves is the knowledge about the number of defects in defect backlog. Using this relationship and applying SRGMs to predict the size of the defect backlog is a popular technique [12]. There is no standard way of selecting the most appropriate SRGMs for given defect data. There are studies that reveal the best-fitted models for reliability in some types of projects. In [6] researchers investigated the distribution of defect inflow in automotive domain projects which could aid in finding the best-fitting SRGMs. This work presents that selecting the appropriate model is the most challenging part of the forecast. There are more than 100 SRGMs.

B. Linear Regression

Linear regression modeling was also applied to the problem of defect backlog prediction. The examples of independent variables which are used in the regression models are [13]:

- program metrics (such as program size, number of variables),
- number of defects found in the earlier phase,
- testing time,
- design methodology.

Yu, Shen, and Dunsmore [13] investigated the correlation between those variables and the number of defects that remain in the software. They discovered the strongest relationship between a number of defects identified during earlier phases of development and those discovered later.

C. Defect Backlog Prediction at Ericsson AB

A research program executed at Ericsson AB company resulted in a few studies on defect backlog predictions. In the first study [4], Software Reliability Growth Models were designed to defect inflow prediction after release. The results were not satisfying. Defects profile described by the model significantly deviated from the profile of defects in the studied project.

In the follow-up study on defect inflow predictions in a large-size software project [5], the prediction accuracy of different methods (e.g., multivariate linear regression or method which used the moving average of defect inflow) was compared. Table II presents average prediction errors depending on number of predictor variables and the used method. To evaluate the accuracy of methods they used the Mean Magnitude of Relative Error (MMRE). The error of none of the evaluated methods was good enough to reach the required accuracy level by the organization.

TABLE II: Extract of prediction accuracy in a large-size project for 1-week interval [5].

Model	Type of model	MMRE (%)
Project milestone progress	Multivariate linear regression + PCA over milestone progress	52
2 weeks moving average	Moving average	34
3 weeks moving average	Moving average	38
Test progress – best statistical models	Multivariate linear regression + PCA over test progress	58
Test progress – statistics and expert combined	Multivariate linear regression over test progress (variables chosen by experts)	28
Expert estimations	Expert estimates based on historical data	375

To improve prediction accuracy researchers from Ericsson decided to conduct a more detailed study on medium-size project [3]. This time they evaluated the prediction accuracy of three methods:

- Multivariate linear regression,
- Analogy-based prediction,
- Expert estimations.

Seven variables identified as most influential on defect inflow were chosen from a set of over 50 and used to construct a multivariate linear regression model. For analogy-based prediction, researchers collected an analogy database (projects that they found the most similar to the one that they were working on). The variables used for calculating similarity were [3]:

- the number of test cases planned in integration testing 4 weeks before the predicted week,
- the number of test cases executed in integration testing 4 weeks before the predicted week.

Also, they decided to enrich analogy-based predictions by involving experts and asking them to choose variables that

they found the most influential on defect inflow and assign them weights.

In the following study at Ericsson AB [3], the problem was reframed to predict defect backlog size instead of predicting defect inflow. A new method was proposed by Meding and Staron (MS) that relied on the moving average of defect inflow and defect outflow and the previous backlog size (see Equation 11). The proposed model allowed for predicting defect backlog size with the highest accuracy (MMRE of 16%) compared to the previous studies.

$$db(i) = db(i-1) \frac{di(i-1)+di(i-2)+di(i-3)}{3} - \frac{do(i-1)+do(i-2)+do(i-3)}{3} \quad (11)$$

where:

- db(x) - defect backlog in week x,
- di(x) - predicted defect inflow in week x,
- do(x) - predicted defect outflow in week x.

IV. Research Methodology

A. Research Goal and Questions

We perform a Simulation-Based-Study (SBS) [14] using the data from OSS projects to *compare the accuracy* of two new defect-backlog prediction models based on Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing (ETS) with the state-of-the-art Meding-Staron model (MS). We formulate the following research questions:

- RQ1: *Are the MS, ARIMA, and ETS models more accurate than the Naïve prediction method?*
- RQ2: *Are ARIMA and/or ETS more accurate than the MS model when predicting the number of defects in defects backlogs of OSS projects?*

The question RQ1 could be considered a sanity test for the models. Shepperd and MacDonell [15] recommend performing such a test against "random guessing," however, we decided to use the so-called Naïve method instead of guessing. This method uses the actual observed values from the last week as the forecast for the next week. Although very simple, the Naïve method is reported to "work remarkably well for many economic and financial time series" [10]. Therefore, our sanity test is more demanding than the one proposed by Shepperd and MacDonell. However, for practical reasons, if a given model does not outperform the Naïve method, there is no point in considering it for real-life applications. The latter question (RQ2) is the central research question of this study. We compare the accuracy of the MS model, which according to the literature is the most accurate model for defect-backlog predictions with two models which are state-of-the-art in time-series predictions.

The replication package for this study is available on GitHub.[1]

[1]https://github.com/paulinaaniola/DefectBacklogPrediction.

B. Dataset

We collected defect backlogs from 20 Bugzilla instances of OSS projects managed by Apache Foundation, Eclipse, Mozilla Foundation, Linux, Open Office, and Libre Office. We selected only the projects that had a sufficiently long defect reporting period. The shortest defect-tracking period was 8 years (Libre Office Draw), while the longest was 22 years (Mozilla Core).

In the first step, we fetched defect reports from the *Bugzilla* service instances of OSS projects. The reports were grouped based on the dates when they were submitted or resolved and their severity level. By counting the number of defects submitted, resolved, or remaining in the backlog, we calculated defect backlog level (number of defect reports still opened at the end of the week), defect inflow (number of defects reported in a given week), and defect outflow (number of defects resolved in a given week) for every *week*.

The resulting dataset consisted of 20 defect backlogs presented in Table III. The beginning of each defect backlog is determined by the date of the first reports submitted to Bugzilla. The end of the bug tracking period is the same for all projects (01-01-2019). The average defect backlog size presented in Table IV ranges from 112 defects/week for Kernel Networking to 29,050 defects/week for Mozilla Core.

TABLE III: Dataset of OSS projects under study.

Project	Defect-tracking period
Eclipse Platform	10-10-2001 - 01-01-2019
Eclipse Birt	15-03-2005 - 01-01-2019
Eclipse Jdt	03-03-2005 - 01-01-2019
Eclipse Data tools	03-03-2005 - 01-01-2019
Eclipse PDE	20-11-2001 - 01-01-2019
Mozilla Firefox	30-07-1999 - 01-01-2019
Mozilla Core	28-03-1997 - 01-01-2019
Mozilla Thunderbird	02-01-2000 - 01-01-2019
Mozilla Calendar	09-11-2000 - 01-01-2019
Kernel File System	18-11-2002 - 01-01-2019
Kernel Networking	15-11-2005 - 01-01-2019
Kernel IO Storage	14-11-2002 - 01-01-2019
Open Office Writer	30-10-2000 - 01-01-2019
Open Office Calc	23-10-2000 - 01-01-2019
Open Office Draw	30-10-2000 - 01-01-2019
Apache Ant	11-09-2000 - 01-01-2019
Apache Apache2	15-01-2001 - 01-01-2019
Libre Office Writer	15-01-2001 - 01-01-2019
Libre Office Calc	08-10-2010 - 01-01-2019
Libre Office Draw	15-01-2011 - 01-01-2019

While visualizing the change in defect backlogs over time, we observed a suspicious phenomenon of rapid, significant drops in the number of defects in the backlog. We perceive them as anomalies that could result from "cleaning" processes of Bugzilla instances from irrelevant defect reports. Figure 1 presents an example of defect backlog level change over time in the Open Office Draw project with at least two sudden major drops in the number of defects around weeks 650 and 860, which are unlikely to be caused by the real defect fixing activities. Unfortunately, such drops are unexpected and poorly predicted by the considered prediction methods. On

TABLE IV: Average defect backlog sizes in OSS projects (defects/week).

Product	Mean backlog size
Kernel Networking	112.38
Eclipse Platform	7,240.95
Eclipse Data Tools	159.05
Eclipse Birt	1,000.21
Eclipse JDT	3,329.15
Eclipse PDE	850.86
Mozilla Calendar	1,219.78
Mozilla Firefox	11,354.33
Mozilla Core	29,050.79
Mozilla Thunderbird	3,708.5
Kernel IO Storage	161.42
Kernel File System	190.59
Open Office Writer	8,123.17
Open Office Calc	3,245.73
Open Office Draw	810.53
Apache Ant	1,213.95
Apache Apache 2	976.48
Libre Office Writer	3,245.32
Libre Office Calc	1,816.79
Libre Office Draw	370.36

the contrary, they have a less visible impact on the prediction errors made by the Naïve method since the error is present only for a single week following such a drop.

To mitigate the effect of sudden drops in the level of the backlog, we decided to split the defect backlogs based on the presence of unexpected drops in the number of defects. The process of dividing backlogs into fragments was the same for all projects and started with differencing the defect backlog level which result is presented in Figure 2. The peaks in the differenced time-series plot correspond to the sudden falls in backlog level from Figure 1. To consistently determine the weeks for which there are sudden decreases in the backlog level and to set the boundaries between fragments in those weeks, the 99.5 percentile of the difference in the backlog between successive weeks was calculated. For instance, Open Office Draw backlog was divided into 5 fragments presented in Figure 3. The final prediction error of a given method for a divided backlog is counted as the average of errors for individual fragments.

C. Predictions And Accuracy Evaluation

We performed training and accuracy evaluation for each individual defect backlog. We used data from all the previous weeks to train the ARIMA and ETS models and predict the number of defects in the backlog for the following week. We based the accuracy evaluation on Absolute Error calculated according to Equation 12 and calculated Mean Absolute Error (MAE) for each backlog.

$$AE = |actual\ value - predicted\ value| \tag{12}$$

We also calculated a variant of standardized accuracy measure (SA_m) [15] that shows a relative improvement in accuracy in comparison to the Naïve method, which was calculated according to Equation 13.

$$SA_m = (1 - \frac{MAE_m}{MAE_n}) * 100\% \tag{13}$$

where:
– MAE_m - Mean Absolute Error for the method m,
– MAE_n - Mean Absolute Error for Naïve method.

We used a non-parametric Wilcoxon signed-rank test to compare AE between prediction models with significance level $\alpha = 0.05$ and Cliff's δ effect-size coefficient to quantify the strength of the observed difference. Cliff's δ evaluates how often the values of one set are larger than the ones from the second set. The thresholds used for Cliff's δ coefficient interpretation proposed by Kitchenham et al. [16] are as follows: δ < 0.112 – negligible, $0.112 \geq \delta$ < 0.276 – small, $0.276 \geq \delta$ < 0.428 – medium, and $\delta \geq 0.428$ – large.

We also calculate the number needed to treat (NNT = δ^{-1}) measure, which is commonly used in the field of medical science. NNT indicates how many patients need to be treated with a drug to heal one patient and is the measure of the medicine's effectiveness. The lowest the NNT value the fastest we achieve the improvement. In the context of this study, NNT could be interpreted as the number of weeks one would have to use a given prediction method A instead of method B to observe improvement in the accuracy for at least one week. We also calculated the average NNT to aggregate information from all the projects.

V. Results And Discussion

Mean prediction errors for the three considered methods and the Naïve method are presented in Table V, while the mean errors transformed to standardized accuracy measures (SA_m) are presented in Table VI.

ARIMA, ETS, and MS predicted backlog sizes with mean errors lower than the Naïve method for most of the projects (mean SA equal to ca. 17.1%, 17.7%, and 10.3%, respectively). However, there were three projects for which the Naïve method performed better than all three other methods. The effect size and the results of Wilcoxon signed-rank tests for comparison between AE (intra-project level) for the considered models and the naive one are presented in Table VII. For every of the considered models that were at least 10 projects for which a statistically significant difference in the central tendency of AE was detected. The effect size was at least "small" for nearly half of the projects, i.e., 10/20 (ARIMA vs. Naïve), 9/20 (ETS vs. Naïve), and 11/20 (MS vs. Naïve). That translates to NNT at the levels of 8 weeks for ARIMA, 14 weeks for ETS, and 36 weeks for MS. Finally, we performed Wilcoxon signed-rank test at the level of MAE (dataset level). In all three cases, the difference in the central tendency for MAE between considered models and the Naïve method turned out to be statically significant. **Therefore, we conclude that all of the considered methods outperform the Naïve method (RQ1).**

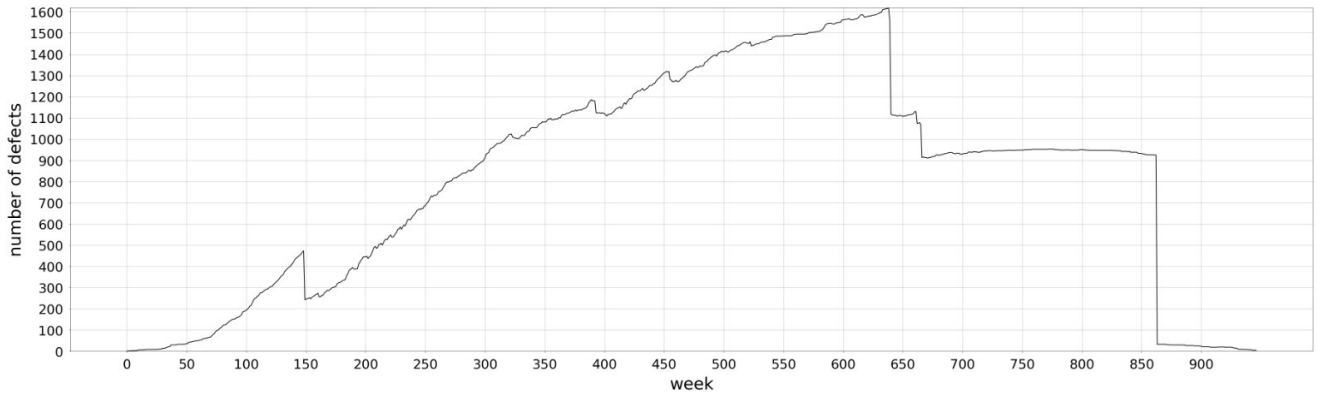

Fig. 1: Defect backlog level changes in Open Office Draw.

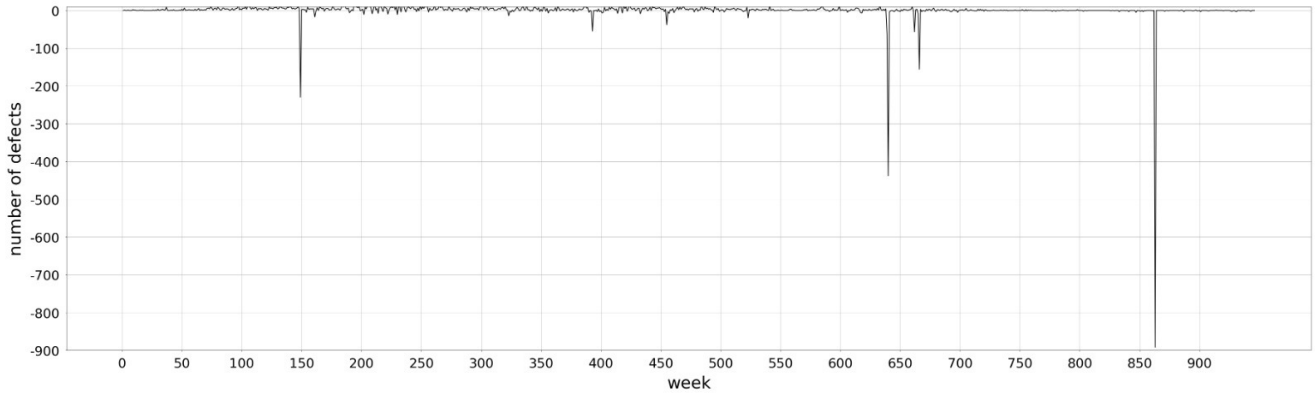

Fig. 2: The differenced time series of defect backlog level in the Open Office Draw project.

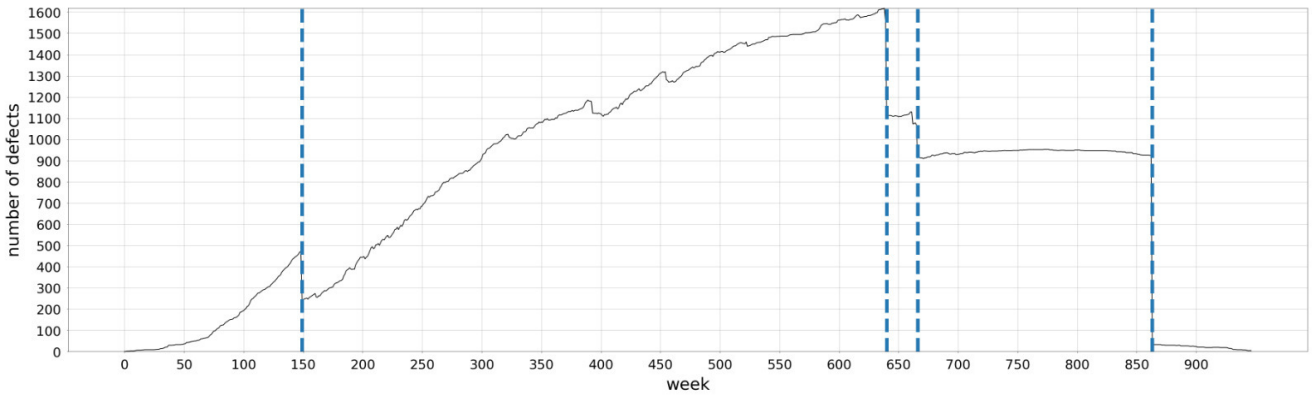

Fig. 3: Open Office Draw defect backlog divided into 5 fragments.

TABLE V: Defect backlog size prediction errors.

Product	MAE_{ARIMA}	MAE_{ETS}	MAE_{MS}	MAE_{Naive}
Kernel Networking	1.97	2.02	2.3	1.81
Eclipse Platform	44.99	42.5	43.39	47.6
Eclipse Data Tools	3.43	3.63	3.23	3.63
Eclipse Birt	15.17	14.82	17.74	16.36
Eclipse Jdt	21.07	20.36	25.19	22.62
Eclipse Pde	7.61	7.6	8.63	7.88
Mozilla Calendar	7.92	7.46	7.78	9.85
Mozilla Firefox	56.81	56.45	57.44	70.11
Mozilla Core	87.04	84.59	85.89	110.63
Mozilla Thunderbird	17.4	18	17.62	21.05
Kernel IO Storage	2.61	2.58	2.88	2.55
Kernel File System	2.95	2.88	2.86	2.79
Open Office Writer	11.73	10.54	14.65	23.6
Open Office Calc	5.74	5.67	7.77	10.69
Open Office Draw	2.23	2.16	2.89	2.71
Apache Ant	3.35	3.01	3.42	6.27
Apache Apache 2	4.53	4.59	5.82	6.23
Libre Office Writer	20.88	21.73	20.76	27.97
Libre Office Calc	13.31	12.94	12.1	17.57
Libre Office Draw	3.24	3.63	3.32	3.81
mean MAE	**16.70**	**16.36**	**17.28**	**20.79**
standard deviation MAE	**22.02**	**21.44**	**21.68**	**27.20**

TABLE VI: Defect backlog prediction improvement (SA) compared to the Naïve method predictions.

Product	SA_{ARIMA} [%]	SA_{ETS} [%]	SA_{MS} [%]
Kernel Networking	-8.84	-11.6	-27.07
Eclipse Platform	5.48	10.71	8.84
Eclipse Data Tools	5.51	0	11.02
Eclipse Birt	7.27	9.41	-8.44
Eclipse Jdt	6.85	9.99	-11.36
Eclipse Pde	3.43	3.55	-9.52
Mozilla Calendar	19.59	24.26	21.02
Mozilla Firefox	18.97	19.48	18.07
Mozilla Core	21.32	23.54	22.36
Mozilla Thunderbird	17.34	14.49	16.29
Kernel IO Storage	-2.35	-1.18	-12.94
Kernel File System	-5.73	-3.23	-2.51
Open Office Writer	50.3	55.34	37.92
Open Office Calc	46.3	46.96	27.32
Open Office Draw	17.71	20.3	-6.64
Apache Ant	46.57	51.99	45.45
Apache Apache 2	27.29	26.32	6.58
Libre Office Writer	25.35	22.31	25.78
Libre Office Calc	24.25	26.35	31.13
Libre Office Draw	14.96	4.72	12.86
mean SA	**17.08**	**17.69**	**10.31**
standard deviation	**16.72**	**18.11**	**19.11**

TABLE VII: Effect size (n-negligible, s-small, m-medium) and Wilcoxon singed-rank tests result for comparison between ARIMA, ETS, MS, and the Naïve method (T – null hypothesis rejected with $\alpha = 0.05$).

	$\frac{AE_{ARIMA}}{AE_{Naive}}$		$\frac{AE_{ETS}}{AE_{Naive}}$		$\frac{AE_{MS}}{AE_{Naive}}$	
Product	δ	**diff. test**	δ	**diff. test**	δ	**diff. test**
Kernel Networking	n	F	n	F	n	T
Eclipse Platform	n	F	n	T	n	F
Eclipse Data Tools	n	F	n	F	n	F
Eclipse Birt	n	F	n	T	n	F
Eclipse Jdt	n	F	n	T	n	F
Eclipse Pde	n	F	n	F	n	T
Mozilla Calendar	s	T	s	T	s	T
Mozilla Firefox	s	T	s	T	s	T
Mozilla Core	s	T	s	T	s	T
Mozilla Thunderbird	s	T	n	T	s	T
Kernel IO Storage	n	F	n	F	n	F
Kernel File System	n	F	n	F	n	F
Open Office Writer	m	T	m	T	s	T
Open Office Calc	m	T	m	T	m	T
Open Office Draw	n	T	n	T	n	F
Apache Ant	s	T	s	T	s	T
Apache Apache 2	s	T	n	T	n	F
Libre Office Writer	m	T	m	T	m	T
Libre Office Calc	m	T	m	T	m	T
Libre Office Draw	n	T	n	F	s	F

In the next step, we compared the accuracy of the state-of-the-art MS method and two methods proposed in this paper that are based on ARIMA and ETS. As it follows from Table V the lowest mean MAE was observed for ETS (16.36) and ARIMA (16.70), however, the mean MAE for MS was only ca. 5% higher (17.28) than the one observed for ETS. It is also visible that the methods perform consistently for all projects, i.e., there are no methods that would visibly outperform other methods on a single project. Also, as it follows from Table VIII, only for 3-4 projects the observed difference in the central tendency for AE (intra-project level) could be considered statistically significant. Also, the effect size could be interpreted as "negligible" for comparing AE for all the projects that translated to NNT at the level of 68 for ARIMA vs. MS and 32 ETS vs. MS. However, statistical inference at the dataset level regarding central tendency in MAE resulted in rejecting the null hypothesis for comparison between ETS and MS methods. **Therefore, we conclude that**

all the considered methods are good candidates to be used for predicting defect backlog size for OSS projects—with a slight preference towards ETS (RQ2). Taking into account NNT, statistically, one shall see improvement in defect prediction for at least one week after applying ETS for 32 weeks instead of MS.

TABLE VIII: Effect size (n-negligible, s-small, m-medium) and Wilcoxon singed-rank tests result for comparison between ARIMA, ETS and MS (T – null hypothesis rejected with $\alpha = 0.05$).

	$\frac{AE_{ARIMA}}{AE_{MS}}$		$\frac{AE_{ETS}}{AE_{MS}}$	
Product	δ	**diff. test**	δ	**diff. test**
Kernel Networking	n	T	n	F
Eclipse Platform	n	F	n	F
Eclipse Data Tools	n	F	n	F
Eclipse Birt	n	F	n	F
Eclipse Jdt	n	T	n	T
Eclipse Pde	n	T	n	T
Mozilla Calendar	n	F	n	F
Mozilla Firefox	n	F	n	F
Mozilla Core	n	F	n	F
Mozilla Thunderbird	n	F	n	F
Kernel IO Storage	n	F	n	F
Kernel File System	n	F	n	F
Open Office Writer	n	F	n	F
Open Office Calc	n	F	n	F
Open Office Draw	n	F	n	F
Apache Ant	n	F	n	F
Apache Apache 2	n	T	n	T
Libre Office Writer	n	F	n	F
Libre Office Calc	n	F	n	F
Libre Office Draw	n	F	n	F

A. Threats to Validity

We address the threats to validity in the manner as described by Wohlin et al. [17] and de França et al. [14].

a) Construct validity: The main construct validity threat concerns the process of cleaning the data. Each backlog was divided into fragments in places of sudden falls in the number of defects. The rule of determining the sudden falls was the same for all backlogs. It assumes a division point between weeks for which the difference in the number of defects in the backlog was more than 99.5 percentile of the differences between successive weeks from the backlog. It resulted that all backlogs being divided, even those in which the aggressive declines have not really taken place.

b) Internal validity: There exists a threat to the internal validity of this study regarding the validity of reported defects. For OSS projects, all users can report defects to Bugzilla. The new reports may be duplicates or not be real defects. We cannot control who makes reports and what they are. Because of that, the size of some defect backlogs is extremely large.

c) External validity: The main threat to the external validity of our results is the fact that we applied the defect backlog prediction methods only to a selected sample of well-established OSS projects that maintain public Bugzilla instances. We do not know whether our results would also apply to smaller OSS projects, however, there is a question of whether such projects would benefit from defect backlog predictions. Also, we limited our study to OSS projects only, therefore, we would be careful in generalizing the findings to industrial projects since the ways of working differ visibly between the OSS and industrial settings. Even when it comes to OSS projects themselves, we have to be aware that the process could be less stable in time than it is for the industry (e.g., the number of contributors involved, the number of commits they produce, or the number of defects they fix can vary in time). Also, we used the 1-week prediction horizon after the previous studies in Ericsson AB, however, we cannot claim based on our results that the methods will behave the same way if a longer prediction horizon is needed by a given OSS community.

d) Conclusion validity: The main threat to conclusion validity regards performing multiple statistical inference tests while drawing some of the conclusions (statistical inference tests at intra-project and dataset levels). We set the local significance level α to 0.05, however, the true, global significance level would be much higher. Still, the outcomes of the statistical inference tests were only one of a few sources of information that we used to draw the conclusions, therefore, the impact of rejecting a true null hypothesis would have a minor impact on the final conclusions.

VI. Conclusions and Future direction

In this paper, we evaluated three defect backlog prediction methods in the context of open-source projects, i.e., the state-of-the-art Meding-Staron model (MS) and two new models based on Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing (ETS) time-series forecasting methods.

We compared the accuracy of these methods on the dataset consisting of defect backlog histories of 20 large open-source projects (ranging from 8 to 22 years). In the first step, we performed a sanity check by comparing the accuracy of these methods with the accuracy of the so-called Naïve method, which predicts a defect backlog size in the following week to be the same as the one observed in the current week. All three methods outperformed the Naïve method by ca. 10% to 17.7% (the largest improvement was observed for the ETS method). The observed differences in mean absolute errors (MAE) were statistically significant for all the methods. With respect to effect size, one would have to use these methods instead the Naïve one for 8 to 36 weeks to, statistically, notice improvement in defect backlog predictions for at least one week.

Exponential Smoothing (ETS) provided slightly more accurate defect backlog size predictions than the state-of-the-art MS method (by ca. 5%) and the prediction method based on ARIMA (by ca. 2%). If one decides to use the ETS-based defect backlog prediction method instead of the MS method should statistically, notice an improvement in defect backlog predictions for at least one week after using it for 32 weeks.

Therefore, the main contributions of our study to the body of knowledge are as follows:

- we provided some new observations regarding the state-of-the-art Meding-Staron model (MS) by evaluating its accuracy in the open source context (in addition to the previous studies in Ericsson). Based on the results of this and the previous studies on that method, we can conclude that it is suitable for defect backlog size prediction in both industrial and open-source settings.
- we proposed two new defect backlog size prediction methods based on the state-of-the-art time series forecasting methods ARIMA and ETS, which perform slightly better (especially ETS) than the MS method in the open source context.
- all the considered defect backlog prediction models based on autoregression (ARIMA, ETS, and MS) are very accurate when estimating defect backlog levels in OSS. Taking into account that the mean size of the defect backlogs for the considered OSS project ranged from ca. 112 to 29,000 defect reports and the mean absolute error for the ETS model ranged from ca. 2 to 85 defects, the relative error for that model was at the level of 0.13% to 2.28% (with respect to the average backlog size of a project).

As future research directions, we plan to investigate the accuracy of artificial neural network-based time-series prediction models (e.g., LSTMs, GRU) for defect backlog predictions. We would also like to validate the proposed models based on ARIMA and ETS based on historical data from industrial projects.

REFERENCES

[1] P. Sielicka, "Defect backlog size prediction with the autoregressive moving average and exponential smoothing models," Master's thesis, Poznan University of Technology, Poland, 2019.

[2] W. Meding, "Effective monitoring of progress of agile software development teams in modern software companies: an industrial case study," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, 2017, pp. 23–32.

[3] M. Staron and W. Meding, "A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation," *Information and Software Technology*, vol. 52, no. 10, pp. 1069–1079, 2010.

[4] ——, "Defect inflow prediction in large software project," *e-Informatica Software Engineering Journal*, vol. 4, no. 1, pp. 89–107, 2010.

[5] ——, "Predicting weekly defect inflow in large software projects based on project planning and test status," *Information and Software Technology*, vol. 50, no. 7, pp. 782–796, 2008.

[6] R. Rana, M. Staron, C. Berger, and J. Hansson, "Analysing defect inflow distribution of automotive software projects." PROMISE '14' - 10th International Conference on Predictive Models in Software Engineering, 2013.

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

[8] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International journal of forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[9] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[10] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice, 2nd edition*. OTexts: Melbourne, Australia, 2013, oTexts.com/fpp2. Accessed on 03.06.2019.

[11] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: the forecast package for R," *Journal of Statistical Software*, vol. 26, no. 3, pp. 1–22, 2008.

[12] R. Rana, "Software defect prediction techniques in automotive domain: Evaluation, selection and adoption," Ph.D. dissertation, University of Gothenburg, 2015.

[13] H. D. T.J. Yu, V.Y. Shen, "An analysis of several software defect models," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1261 – 1270, 1998.

[14] B. Bernard Nicolau de França and G. Horta Travassos, "Simulation based studies in software engineering: A matter of validity," *CLEI electronic journal*, vol. 18, no. 1, pp. 5–5, 2015.

[15] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[16] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Engineering*, vol. 22, no. 2, 2017.

[17] C. Wohlin and et al, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publisher, Boston, MA, 2000.