Software Evolution Prediction Using Seasonal Time Analysis: a Comparative Study

Miguel Goulão, Nelson Fonte
CITI, Dep. Informática
FCT/Universidade Nova de Lisboa
Caparica, Portugal
miguel.goulao@di.fct.unl.pt, n.fonte@gmail.com

Michel Wermelinger
Centre for Research in Computing
The Open University
Milton Keynes, UK
m.a.wermelinger@open.ac.uk

Fernando Brito e Abreu
CITI, DCTI
ETA/ISCTE-IUL
Lisboa, Portugal
fba@iscte.pt

Abstract – **Prediction models of software change requests are useful for supporting rational and timely resource allocation to the evolution process. In this paper we use a time series forecasting model to predict software maintenance and evolution requests in an open source software project (Eclipse), as an example of projects with seasonal release cycles. We build an ARIMA model based on data collected from Eclipse's change request tracking system since the project's start. A change request may refer to defects found in the software, but also to suggested improvements in the system under scrutiny. Our model includes the identification of seasonal patterns and tendencies, and is validated through the forecast of the change requests evolution for the next 12 months. The usage of seasonal information significantly improves the estimation ability of this model, when compared to other ARIMA models found in the literature, and does so for a much longer estimation period. Being able to accurately forecast the change requests' evolution over a fairly long time period is an important ability for enabling adequate process control in maintenance activities, and facilitates effort estimation and timely resources allocation. The approach presented in this paper is suitable for projects with a relatively long history, as the model building process relies on historic data.**

Keywords
Software Evolution, Eclipse, Bugzilla, Change requests prediction model, ARIMA.

I. MOTIVATION

Software process managers need to be able to predict the evolution of change requests over time, as this helps them planning ahead the resources required for evolution. The evolution process is not necessarily a stationary one. There may be periods where a large number of change requests, which may refer to defects found in the software or to suggested improvements, are received, interleaved with other periods where the number of requests is lower. A model for predicting the number of change requests and the time required to address them must take into account this variability, otherwise its accuracy will decrease.

In long lived software development processes we can expect the profile of change requests to reflect the pace of the releases of the project. While in some projects the timings of the releases may be very flexible and, in that sense, harder to predict beforehand, in Eclipse and many other projects there is a seasonal pattern for releasing new versions and milestones of the project. If we build this seasonal information into prediction models, this is likely to improve their accuracy, compared to models without that information. While the evolution of change requests is likely to exhibit trends, the seasonal patterns resulting from the release policy followed in the project are expected to affect those properties. We propose a prediction model that allows to accurately forecast the distribution of change requests for a large period (12 months, in this paper).

A time series is a sequence of data points measured at successive, uniformly spaced, data points. Adjacent observations are not independent. Time series analysis provides methods for analyzing the nature of those dependencies. Often, time series can be described in terms of two components: trend and seasonality. A trend is a systematic linear or non-linear component that does not repeat within the time range captured by our data. In contrast, seasonality is similar in form, but repeats itself in systematic intervals over time[1]. In this paper, we build a forecast model for the Eclipse project, given the number of previous and current change requests. The model is an **a**uto**r**egressive **i**ntegrated **m**oving **a**verage (**ARIMA**) model, fitted to time series data of change requests, to allow a deeper understanding of the evolution of the number of change requests and to predict future data points in the change requests time series.

We aim at addressing three research questions:

(*RQ1*) Can we detect a seasonal pattern in the number of change requests submitted to the change requests tracking system? If so, what is causing that pattern?

(*RQ2*) Is there a statistically relevant trend in the distribution of change requests over time?

(*RQ3*) Is the usage of ARIMA models with seasonal information a valid and accurate approach to forecasting change requests in a long-lived, open-source, software system for which a seasonal releases pattern occurs?

The goal of this study is to analyze the Eclipse change request system, for the purpose of forecasting its evolution with respect to the number of change requests. The perspective is that of researchers who intend to investigate how different approaches can be used by project managers to accurately forecast the evolution of the change requests. The context is an observational study on the Eclipse change requests available in

The authors would like to acknowledge CITI - PEst - OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) - for the financial support for this work.

Eclipse's Bugzilla repository. The included change requests range from November 2001 to December 2009, which corresponds to over 270000 change requests. As such, the resulting model is expected to be valid in the context of Eclipse. The approach is generic and applicable to other software systems with a seasonal release policy, such as Eclipse's.

II. Related Work

Predicting change requests and defects for the next release of a software project has been a challenge for software project managers. Mining repositories for creating prediction models is a common approach and has been used with several software systems, including Eclipse. Several alternative techniques have been tried.

Zimmerman *et al.* used linear regression to predict the number of defects in Eclipse [2]. They combined defect tracking system information with complexity metrics to predict defects and their location in the source code, using linear regression models.

Bernstein *et al.* argued that using temporal features is central to prediction performance and combined them with non-linear models for defect prediction [3]. They combined CVS and Bugzilla information on 6 plugins of the Eclipse project to extract temporal features such as the number of revisions and reported issues in the previous three months, to predict the location of defects and number of bugs to be found in the next month of the development project. The usage of temporal features significantly improved the accuracy of their prediction models.

Kläs *et al.* combined the use of time series with expert opinion to create prediction models for defects [4], to mitigate for the lack of historical information early in the development process. They reduced the mean magnitude of relative error from 76,5% (with a data-based model) to 29,6% (with their hybrid model). The hybrid approach proved particularly more powerful than the data-based alternative early in the project's life cycle.

All the above mentioned works used other sources of information than just the change requests tracking system, as we do in this paper, and were targeted to predicting the number of defects, while we are aiming at the number of change requests.

Raja *et al.* successfully used the *ARIMA(0,1,1)* model to predict the number of defect reports in eight open source software products from different organizations, which are developed, evolved, and managed independently [5]. They used monthly defect reports collected through periods of 5 or more years and applied their model to predict the evolution of the number of reported defects in the next 4 months. Their approach sacrifices the benefits of fine tuning the prediction models to each of the products in favor of the simplicity of always applying the same model. They compared the performance of their model to that of an *ARIMA(1,1,0)* model. In this paper, we will use both models as comparisons for our own prediction model.

Kenmey *et al.* also use *ARIMA* models for predicting change requests, but build a different model for each of the four open-source systems (including Eclipse) they study [6]. They adopted a sampling policy of aggregating data every two weeks, so that they would have longer time series than if using monthly data. They used their models to predict the number of new defects in the next two, four, and six weeks. Their model for predicting defects in Eclipse is also used in this paper for comparison, but using monthly data. This adaptation aims at facilitating the comparability of the results, as it allows us to use exactly the same time series with all models.

Raja *et al.* and Kenmey *et al.*'s models are used in this paper, with the necessary adaptations. We make forecasts for a whole year, rather than just 4 months, or 6 weeks, respectively. We use a monthly granularity, unlike Kenmey *et al.*'s work. We use the same data set for all the tested prediction models. In that sense, our work can also be regarded as a differentiated replication of those two works.

None of the studies mentioned so far in this section used seasonal information. In this paper, we argue that seasonal information can be useful for improving prediction models (in this paper, change requests prediction models). Hindle *et al.* use Fourier analysis to discover periodicities in software change events [7] (i.e. changes recorded in source control systems). Their preliminary results suggest that recurrent activities are indeed observable, suggesting the existence of seasonal phenomena in projects such as Mozilla, MySQL, Evolution, MaxDB, and Xerces. In particular, seasonal phenomena such as a tendency for making more error-prone changes to software on Fridays than on other days of the week, or making more commits to the version control system on a particular day of the week (e.g. Fridays for Eclipse, Saturdays for Mozilla) have been observed [8]. In a different context (IT service management), Caldeira used time series models with seasonal information (daily and weekly) for predicting the evolution of IT service management trouble tickets in the context of commercial software products from a large software vendor [9], and concluded that the seasonal information allowed significant improvements in his prediction models. These three works focused on relatively short-termed seasonal effects (daily, or weekly), which turned out to be observable and, in Caldeira's work, useful for prediction models. In this paper we are looking for seasonal effects on a yearly basis, to enhance the prediction ability of our model.

Herraiz *et al.* used *ARIMA* models to predict changes in the CVS repository [10]. They created an *ARIMA* model for each of the plugins in the Eclipse CVS repository and used those models for predicting changes in a 3 months' time span. In practice, each of the plugins' series was modeled with one of two models: *ARIMA(2,1,0)* or *ARIMA(3,1,0).* As these models were built for predicting changes in the CVS, rather than change requests in the change requests tracking system, we do not use them for comparison purposes in this paper.

Time series analysis is also often used to understand the past of software projects, as a stepping stone for predicting the future. Mens *et al.* used time series and a set of software metrics to study the evolution of Eclipse [11] with respect to continuing change, increasing complexity and continuing growth, three of the *laws of software evolution* [12]. Wermelinger *et al.* also used time series for understanding the evolution of Eclipse [13] with respect to several design principles and guidelines, assessed at the architectural level. The evolution of software clones is another example of the successful usage of time series analysis in software evolution research [14].

III. EXPERIMENTAL DESIGN

Our experimental goals, labeled as *EGi*, match the research questions *RQi*:

(*EG1*) Our goal is to analyze Eclipse change requests with the purpose of characterizing their evolution over time, with respect to seasonal patterns, so that we can build prediction models to be used by process managers.

(*EG2*) Our goal is to analyze Eclipse change requests with the purpose of characterizing their evolution over time, with respect to trends, so that we can build prediction models to be used by process managers.

(*EG3*) Our goal is to define an *ARIMA* model for the time series of the number of change requests in Eclipse, and check the extent to which the predictions provided by the *ARIMA* model to be built are accurate.

The *theoretical population* of this observational study is the set of Eclipse's change requests, which are stored in a Bugzilla repository. We consider the *sampling frame*, with all change requests from November 2001 to the end of December 2009, as representative of Eclipse's change requests. Our goal is to forecast the number of change requests in a given future timeframe. The *independent variable* is the timeframe in which we want to know the number of change requests. The *dependent variable* is the number of change requests. The hypotheses tested in this observational study are directly derived from the research questions (table I).

TABLE I. TESTED HYPOTHESES

H1	***$H1_0$***: The time series of the number of change requests has significant seasonal patterns.
	$H1_1$: The time series of the number of change requests has no significant seasonal patterns.
H2	***$H2_0$***: The time series of the number of change requests has a significant trend.
	$H2_1$: The time series of the number of change requests has no significant trend.
H3	***$H3_0$***: Predicting the number of change requests using an ARIMA model is a valid and accurate approach.
	$H3_1$: Predicting the number of change requests using an ARIMA model is not a valid and accurate approach.

We use a *longitudinal study design*, where the *observations* are the counting of change requests submitted in a given month (no duplicates detection is employed). In this study, we regard the introduction of a new version, or a milestone, of Eclipse as a *treatment*. We represent each observation with an O, each new version with X, and each new milestone with x, and consider that new versions are typically introduced between by the end of May, or beginning of June, and new milestones are introduced between February and March, and between October and November, a typical year would be presented in Figure 1, where the first line represents each month by its first letter, and the second indicates a series of observations and treatments:

```
Month:      J F   M A M   J J A S O   N D
Obs/Treat:  O O x O O O X O O O O O x O O
```

Figure 1. A typical year in Eclipse

We have a total of 86 consecutive months, for training the prediction model, followed by 12 observations for testing the model (which are not used while training the model).

Our goal is to build an *ARIMA* model, fitted to time series data, to allow a deeper understanding of that data and to predict future data points in the series. The first step is to produce and analyze a correlogram, which is a plot of the sample autocorrelations versus the time lags. This plot shows the correlation among successive values of change requests in our sample. In this first stage, we look for possible seasonal patterns and trends. Correlograms are also helpful to identify the parameters used in *ARIMA* models.

An *ARIMA* model includes an auto-regressive (*AR*) parameter *p*, as well as a moving average (*MA*) parameter *q*. It also includes a parameter *d* specifying how many transformations were made (in our case, differentiations). In order to successfully apply an *ARIMA* model for prediction, the time series must be stationary. In other words, it has to have a stable mean, variance, and auto-correlation throughout the series. As such, we often have to differentiate the series until it becomes stationary. Sometimes, it is also necessary to perform a logarithmic transformation, to stabilize the variance, although this is not the case in this study. The number of differentiations corresponds to the *d* parameter in our model.

We also have to define the auto-regressive (AR) and moving average (MA) parameters (*p* and *q*) so that we have a model which is simultaneously effective and parsimonious. This selection is made through the analysis of auto-correlation function (ACF) correlograms, and partial auto-correlation function (PACF) correlograms. The selection process of parameters is not trivial. However, in most situations, one of the following basic models can be identified through the visual inspection of the ACF and PACF correlograms, following the practical recommendations for selecting parameters *p* and *q*, described in [19, 20]:

- One AR parameter *p*: ACF decreases exponentially; PACF has a peak in lag 1, with no correlation to the remainin lags.
- Two AR parameters *p*: ACF decreases exponentially; PACF has peaks in lags 1 and 2, with no correlation to the remaining lags.
- One MA parameter *q*: ACF has one peak in lag 1 with no correlation to the remaining lags; PACF falls exponentially.
- Two MA parameters *q*: ACF has peaks in lags 1 and 2, with no correlation with the remaining lags; PACF has a sine wave shape, or a set of exponential decreases.

Finally, after selecting the model's parameters, we compare the proposed model with several other models described in the literature for evaluation purposes.

IV. ANALYSIS

Eclipse is a plug-in based architecture. Its architecture contains a core base and a set of plug-in components that build up an Eclipse standard distribution. Figure 2 represents the change requests in the Eclipse project, from November 2001 to December 2009. The blue line, labeled *All*, represents all the change requests stored in Bugzilla. The green line, labeled *Core*, only includes change requests related to the core components, categorized in Bugzilla as “Eclipse”. We can observe an apparent decreasing trend in the number of change requests related to the core, particularly after 2004. The majority of change requests are, nowadays, related to plug-ins within the Eclipse project that are not part of the core.

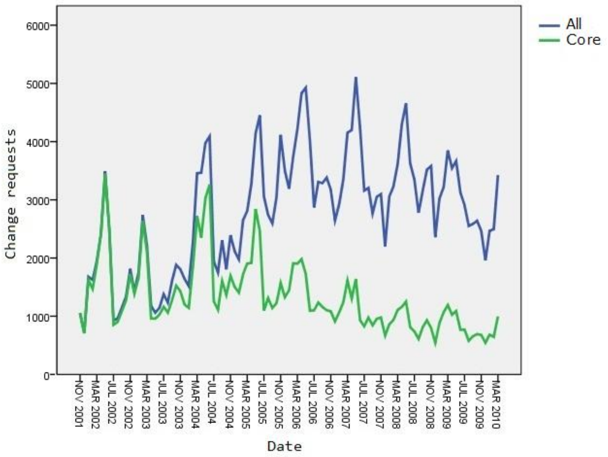

Figure 2. Eclipse change requests evolution

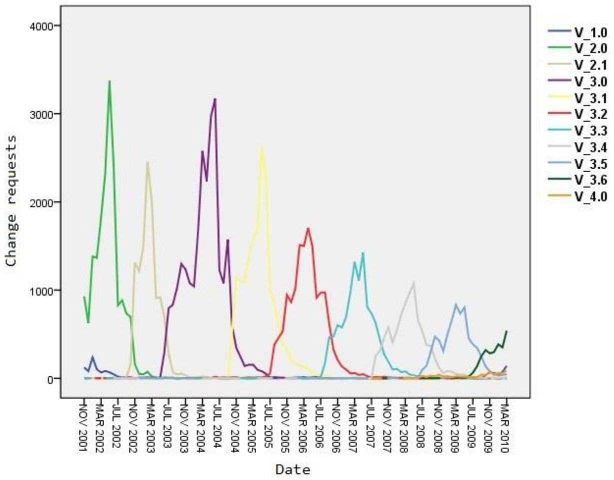

Figure 3. Change requests associated to new versions

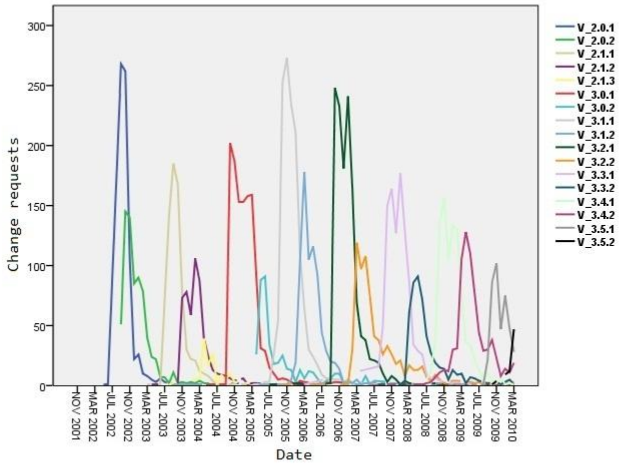

Figure 4. Change requests associated to new milestones

In general, the months from March to June are very active, with a relatively high number of change requests. This seems to be related with the launch of a new version, which usually happens by the end of June (Figure 3).

For each milestone within a new version, there is usually a relatively large number of change requests, which is considerably lower in subsequent milestones of the same version. This suggests that the version is becoming more stable, from one milestone to the next (Figure 4).

A. *Hypotheses testing*

Hypothesis H1: We looked for seasonality patterns in the number of new change requests per month. We started by analyzing the Auto Correlation Function (ACF), which represents the serial correlation coefficients for consecutive lags in a specified range. When analyzing the ACF we noted an overall decreasing correlation factor, with local stronger correlation factors occurring every 12 months (referred to as Lags). This suggests the presence of a seasonal pattern occurring every 12 months (Figure 5). We used the Partial Auto Correlation Function (PACF) to confirm the existence of a seasonal pattern. PACF is an extension of ACF, where the dependence on the intermediate elements (those within the lag) is removed (Figure 6).

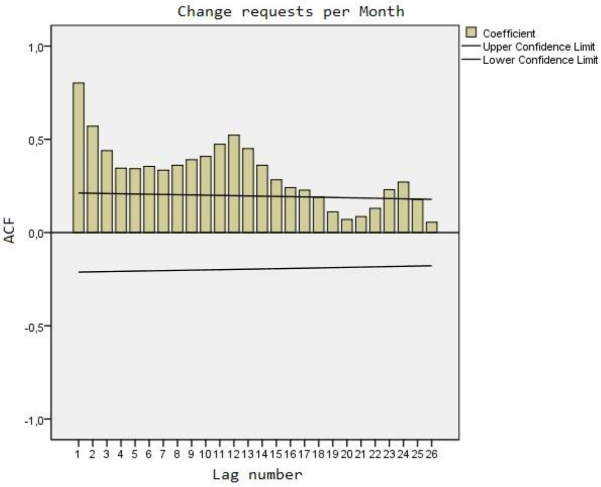

Figure 5. ACF for the change requests time series

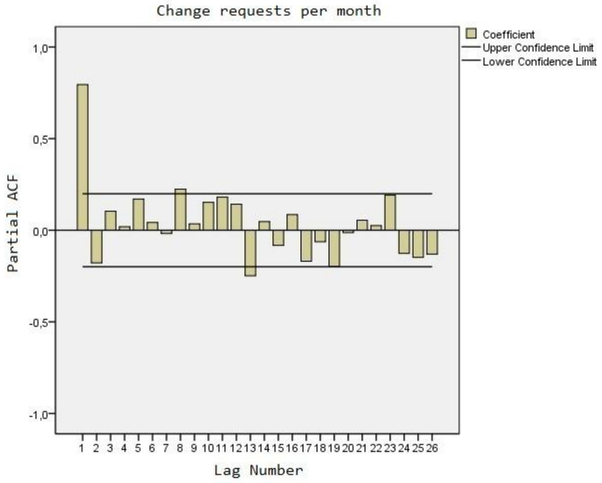

Figure 6. PACF for the change requests time series

As the PACF evaluation was inconclusive, we then observed the distribution of change requests over the months (Figure 7), where we see that some months appear to consistently have more change requests than others. From March to June, the number of requests seems to be consistently larger, indicating a seasonal pattern. These months usually precede the launch of major releases in Eclipse. In this analysis, May 2003 and December 2005 are identified with an unusually low and high number of change requests, respectively, but we can regard these as exceptional observations, overall.

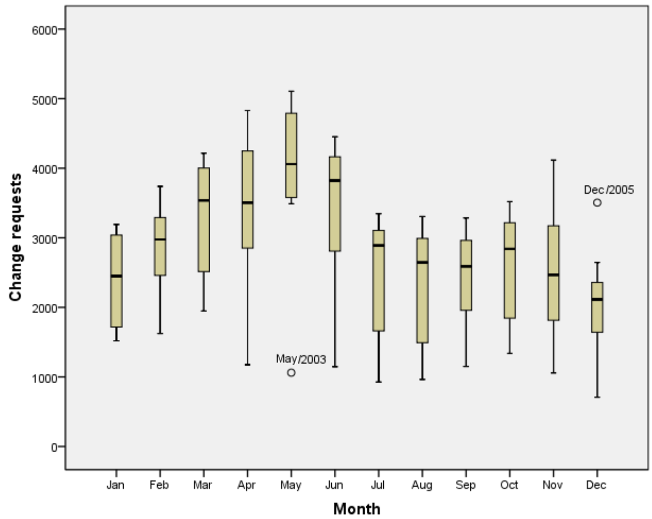

Figure 7. Boxplot of change requests distribution per month

To further explore this matter, we conducted the Kruskal-Wallis [15] and Jonckheere-Terpstra [16] tests. Both are non-parametric tests used to assess whether several samples could originate from the same population. We divided change requests per month, thus obtaining 12 samples (most of them with 8 countings each). In both tests, the null hypothesis is that the distributions of the 12 samples are the same. The alternative is that at least one of them is significantly different, which would suggest seasonality. We obtained a *p-value* of 0.003 for the Kruskal-Wallis test, and of 0.013 for the Jonckheere-Terpstra test and we can safely reject the null hypothesis. In other words, some months consistently have a significantly different number of change requests over the years. Overall, we conclude that the seasonal pattern exists, occurring every 12 months.

Hypothesis H2: There is no exact way of determining whether or not a trend exists in a time series. In the time series representing all the change requests (Figure 2), there seems to be a growing trend, particularly in the first years, which then seems to stabilize by the beginning of 2006, but the seasonal variations have a confounding effect in this analysis. To confirm this, we use seasonal decomposition, to decompose the time series into (i) a seasonal component which is, in turn, decomposed into two series (Seasonal Adjustment Factor – SAF – which indicates the effect of each period in the time series, and Seasonal Adjustment Series – SAS – which is the series obtained once the seasonal variation of the original series is removed), (ii) a component combining the trend and cycle of the series (STC), and (iii) an error component (ERR) representing the residuals of the de-trended series.

We then focus our attention on the STC series (Figure 8), where we can confirm that there is an overall growth tendency from the beginning of the series up until early 2006, in spite of the decrease in the first semester of 2003. The largest increases in this period led to the launch of versions 3.0 and 3.1, in June 2004 and June 2005, respectively. From then on, the series becomes fairly stationary, although with a slight decrease which was not so perceptible in Figure 2.

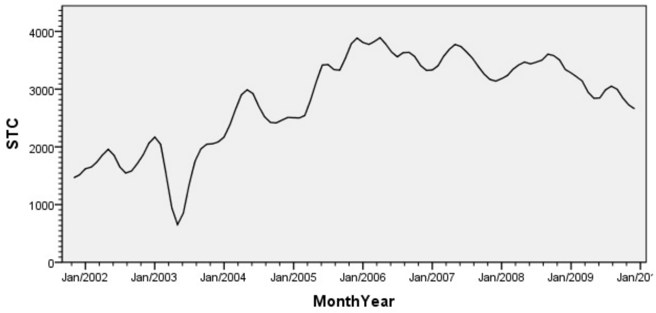

Figure 8. STC series

To validate this tendency, we need to analyze the residuals of the de-trended series (ERR). Figure 9 presents a histogram of the residuals, which have a normal distribution. This is confirmed using the Kolmogorov-Smirnov with Lilliefors correction [17] and the Shapiro-Wilk [18] tests (see table II). All the pre-requisites for applying *ARIMA* models were met.

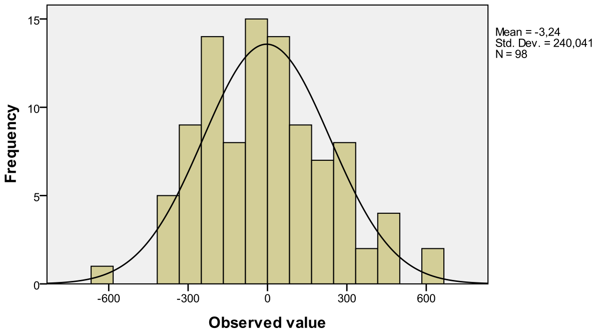

Figure 9. STC series

TABLE II. NORMALITY TESTS FOR ERR

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Statistic	**df**	**Sig.**	**Statistic**	**df**	**Sig.**
ERR	0.607	98	0.200	0.982	98	0.210

Overall, we conclude that there is a growing trend until around January 2006, followed by a slight decrease since then. The seasonal decomposition also shows that the requisites for using *ARIMA* models are met.

Hypothesis H3: An *ARIMA* model includes an **a**uto-**r**egressive (AR) parameter *p*, the number of differentiation steps used *d* (the differentiation step corresponds to the "**i**ntegrated" part of the model) and the **m**oving **a**verage (MA) parameter *q*. An *ARIMA* model can then be specified as *ARIMA(p, d, q)*. *ARIMA* models may require three non-seasonal parameters, *p*, *d*, *q*, and three seasonal parameters, *ps*, *ds*, *qs*: the auto-regressive seasonal parameter *ps*, the seasonal differentiation *ds*, and the seasonal moving average parameter *qs*. The series does not have to be stationary to begin with, but the correct application of *ARIMA(p,d,q)(ps,ds,qs)* models requires the series to be stationary. A series is considered stationary if there are no systematic changes in its variation and if the strictly periodic variations have been removed. If the series is not stationary, it is common practice to transform the series through differentiation. The number of required transformations until the series becomes stationary corresponds to the *d* parameter, in *ARIMA* models. In this study the original series is not stationary, with a growth trend until 2006 and a slight decrease since then, and strictly periodic variations, but the series obtained after the first differentiation is stationary, as shown in Figure 10.

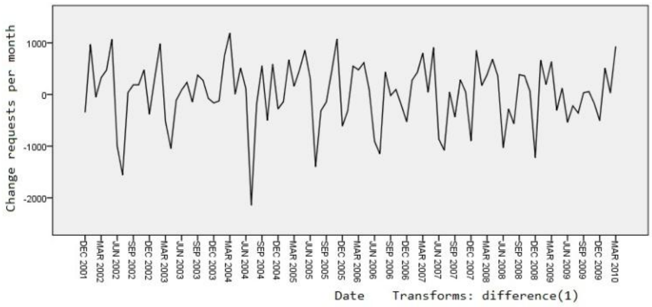

Figure 10. Time series with differentiation (1)

The PACF presented in Figure 11 further supports the observation that the differentiated series is stationary, as there is no apparent correlation among the lags. Therefore, the *d* parameter should be set to 1, in our ARIMA model.

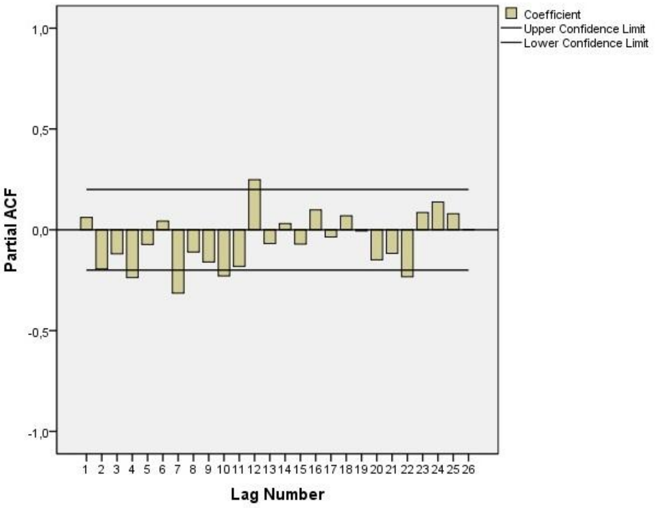

Figure 11. PACF of Time series with differentiation (1)

The non-seasonal parameters were chosen through the analysis of the ACF and the PACF, following the guidelines available in [19, 20]. We chose the *AR* parameter *p*=1 and the *MA* parameter *q*=0.

As we identified the existence of an annual seasonal pattern, we used one seasonal differentiation. After this, the ACF did not present a well defined pattern, as shown in Figure 12, which supports the idea that one seasonal differentiation was adequate.

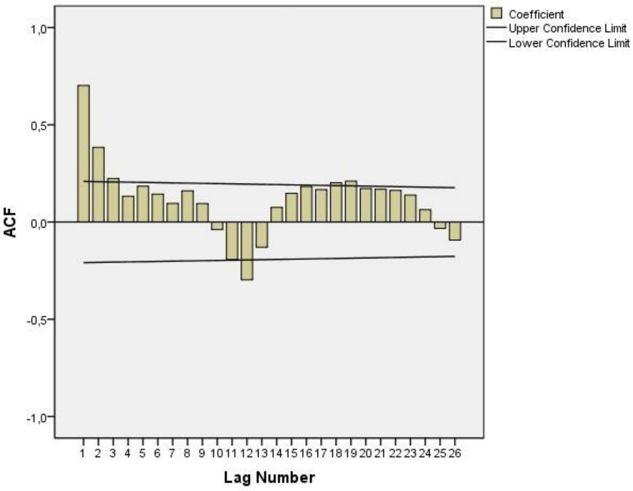

Figure 12. ACF after seasonal differentiation (1)

The PACF still had a peek in lag 1, but no apparent correlation with the remaining lags (Figure 13), also suggesting that one seasonal differentiation is adequate. Therefore, the *ds* parameter should be set to 1.

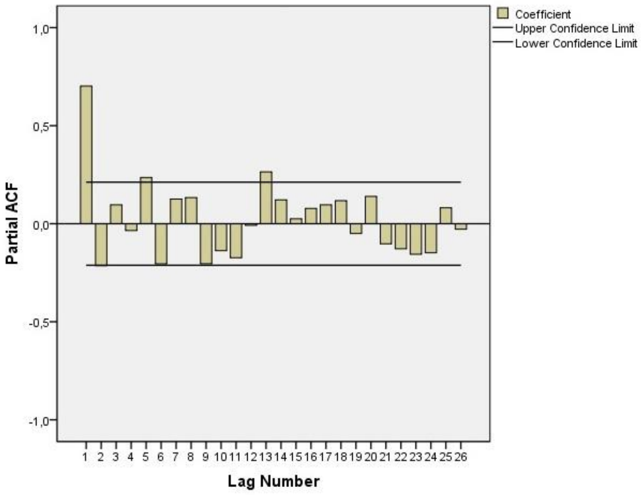

Figure 13. PACF after seasonal differentiation (1)

Overall, following the guidelines available in [19, 20], we choose the *AR* parameter *ps*=1 and *MA* parameter *qs*=0. The chosen model is the *ARIMA(1,1,0)(1,1,0)*, with a training period from November 2001 to December 2008 and a forecast period from January 2009 to December 2009. The resulting model is illustrated in figure 14, where we can see the observed time series, the fit time series, the forecast time series (which continues from the end of the fit time series, starting at January 2009), the upper confidence limit (UCL) and lower confidence limit (LCL), with a 95% confidence interval. The model seems to make reasonably accurate estimates, although in general it overestimates the number of change requests, in the forecasted period.

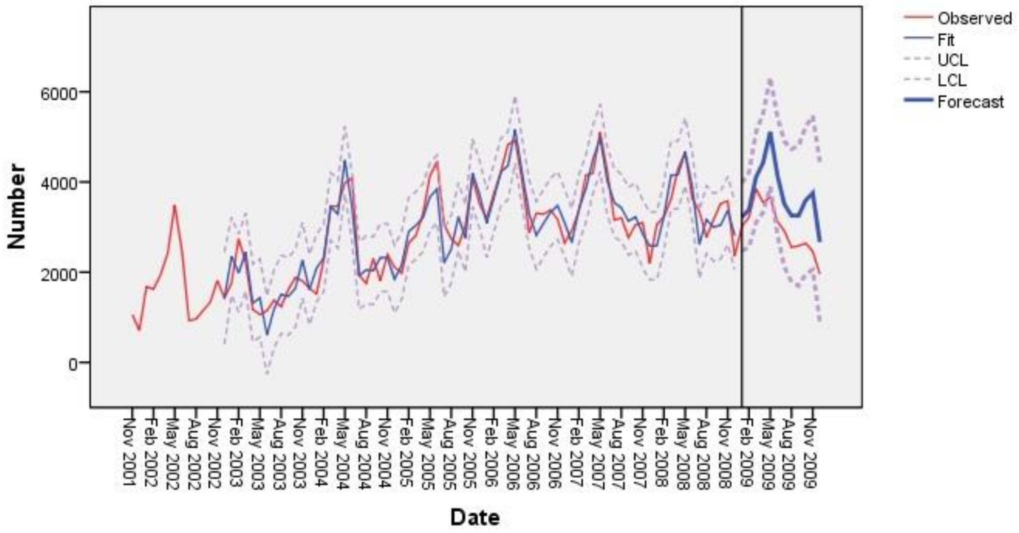

Figure 14. ARIMA(1,1,0), (1,1,0)

TABLE III. TESTED HYPOTHESES

Model	**Model Fit statistics**							**Ljung-Box Q(18)**			
	Stationary R-squared	**R-squared**	**RMSE**	**MAPE**	**MAE**	**MaxAPE**	**MaxAE**	**Statistics**	**DF**	**Sig.**	**Outliers**
ARIMA(110)(110)	***0.743***	***0.872***	***384.806***	***11.464***	***293.043***	***47.106***	***848.347***	***15.850***	***16***	***0.463***	***6***
ARIMA(110)(000)	0.149	0.681	611.421	19.579	469.599	170.267	1578.374	28.649	17	0.038	1
ARIMA(010)(010)	0.519	0.760	510.102	13.729	375.977	56.549	1282.146	29.103	18	0.047	4
ARIMA(011)(000)	0.155	0.683	609.299	19.521	468.542	160.486	1487.703	27.976	17	0.045	1
ARIMA(505)(000)	0.793	0.793	526.691	16.844	389.908	100.130	1288.319	11.002	8	0.202	1

Table III summarizes the model adjustment statistics for all the models analyzed in this paper, starting with our proposed model, the ARIMA(1,1,0)(1,1,0). For easier identification, the statistics of the model proposed in this paper are written in bold italic font. The presented model adjustment statistics include goodness-of-fit measures, namely stationary R-squared, R-squared, root mean square error (RMSE), mean absolute percentage error (MAPE), mean absolute error (MAE), maximum absolute percentage error (MaxAPE) and maximum absolute error (MaxAE). They also include information concerning residuals, namely autocorrelation function (ACF), partial autocorrelation function (PACF), the Ljung-Box Q test, and the number of outliers found in the residuals.

The Stationary R-Squared measure compares the stationary part of a model with a simple averages model. It is preferable to the R-Squared when there is a trend, or a seasonal pattern (as it happens with our time series). The R-Squared value estimates the proportion of total variation explained by the model. Both the stationary R-Squared and the R-Squared measures range from negative infinity to 1. Positive values indicate that the model is better than the baseline model, and the closer they are to 1, the better. RMSE, MAPE, MAE, MaxAPE, and MaxAE are different measures of distance between the predicted and the actual values of the models and should, as such, be as close to 0 as possible. While the first three concern error margins means, MaxAE and MaxAPE are interesting for assessing worst-case scenarios (in this context, how far the prediction values can get from the actual values). The Ljung-Box Q [21] tests that the residuals of a fitted ARIMA model have no autocorrelation. Finally, the outliers' column presents a count of the observations that are highly inconsistent with the remainder of the data. The same criteria for outliers detection was used with the 5 models.

We include 5 models in our analysis: our proposed model, our proposed model without the seasonal information, a random walk model and models borrowed from [5] and [6]:

- **The proposed *ARIMA(1,1,0)(1,1,0)* model**. This model has the second highest value in the stationary R-Squared, with a value which is close to the best among the models in this study, and the best R-Squared value. For all the error statistics, it is the model leading to the smallest errors. The Ljung-Box Q test indicates, through its high significance value, that the model is suitable and well-adjusted to the time series. The residual values generated by our model are not auto-correlated and have a normal distribution. This model has more outliers than the remaining ones. The detected outliers occurred from April to June, 2003, in March 2004, November 2005 and June 2006. These months were exceptional deviations from the overall pattern (e.g. the decrease in the number of change requests in the first semester of 2003 is unparalleled in Eclipse). In conclusion, this ARIMA model can be considered appropriate for estimating future change requests.

- **Our ARIMA model, without the seasonal information**. This ***ARIMA(1,1,0),(0,0,0)*** model excludes the seasonal information from the proposed *ARIMA(1,1,0)(1,1,0)* model. Our rationale for including this model in this comparison is to assess the impact of removing seasonal information. All the model adjustment statistics are worse than for its seasonality-informed counterpart, and the Ljung-box Q test significance is lower than 0.05 suggesting that this model is not robust and, therefore, not appropriate for this time series.

- A **random walk model**. In time series analysis, it is common to include a random walk model in the study for the purpose of comparing it with the model being proposed. The random walk model excludes the auto-regressive (AR) and moving average (MA) parameters. In other words, it can be described as *ARIMA(0,d,0)(0,ds,0)*. Since we have differentiated our series once non-seasonally, and also once seasonally, the random walk model to be used is ***ARIMA(0,1,0)(0,1,0)***. In the random walk model the fitness statistics are worse than in our prediction model. This is less noticeable in MAPE and MaxAPE, but more evident in Stationary R-Squared, R-Squared, RMSE, MAE and MaxAE. The Ljung-Box test confirms that the random walk model is significantly less robust than our prediction model and, therefore, not appropriate.

- **Raja *et al.*'s model**. This model, proposed in [5], is an ***ARIMA(0,1,1)(0,0,0)***. This model has lower stationary R-Squared and R-Squared values, higher error values and considerably higher error measures than the model proposed in this paper. The Ljung-Box test confirms that this model is not robust for our time series.

- **Kenmey *et al.*'s model**. This model, proposed in [6], is an ***ARIMA(5,0,5)(0,0,0)***. This is clearly the strongest alternative to the model proposed in this paper among those used in this comparison. It outperforms our model with respect to the stationary R-Squared measure and is also a robust prediction model for our time series, as the Ljung-Box test confirms. However, it leads to significantly higher error rates, with all the five different metrics used, so, it is overall outperformed by our proposed model. Figure 15 presents this model, and we can observe that the lack of seasonal information prevents the model from accompanying the variations throughout the tested year, thus leading to the higher error rates. It performs well in the first few months, but fails to predict the usual decrease of requests after June.

V. Interpretation

A. *Evaluation of results and implications*

Not all the Eclipse versions followed the same pattern with respect to releases. Version 2.1 had a different number of milestones launched in different moments. These variations are likely responsible for making this seasonal pattern harder to identify. There is usually a burst of change requests preceding the launch of a new version, which usually occurs around June, although for version 2.1 it occurred before March, when version 2.1 was launched.

The time series starts with relatively small variations, and then has an important decrease, which coincides with the 3 milestones of version 2.1. From then on, there is a steady increase of change requests, corresponding to the introduction of the versions 3.0 and 3.1 of Eclipse. The series becomes stationary, from then on. The 3.0 version of Eclipse was a "revolutionary" one, with respect to the architecture of Eclipse's core, with the introduction of the OSGi Service platform for supporting the runtime architecture of Eclipse. This change helped removing a large amount of external dependencies, and led to a far less decoupled architecture, in the subsequent versions of Eclipse [13].

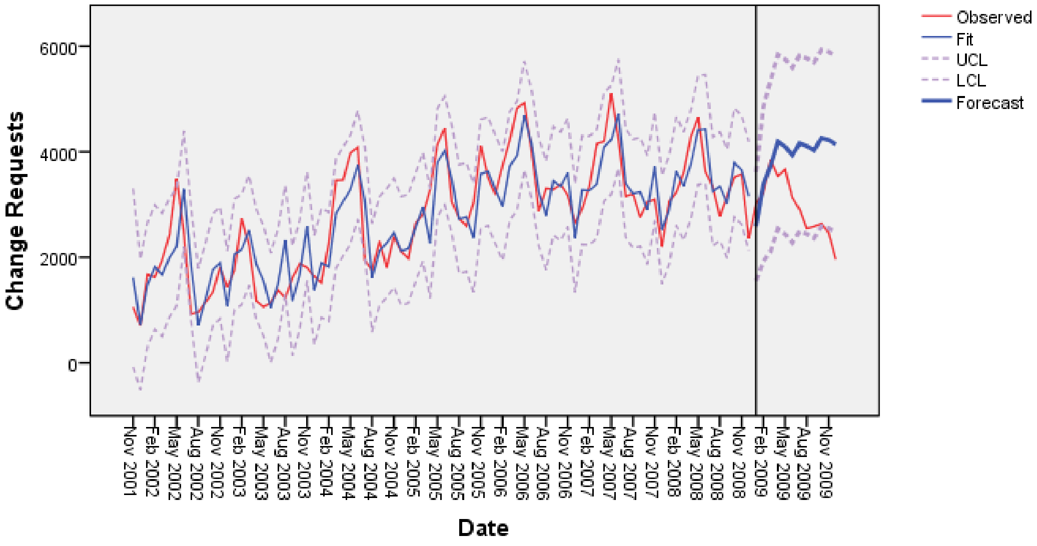

Figure 15. ARIMA(5,0,5)(0,0,0)

We confirmed that the *ARIMA(1,1,0)(1,1,0)* model is valid and more accurate for predicting the evolution of change requests in Eclipse than the other models in this comparative study. All the distance measures between the predicted and the actual values of the models (RMSE, MAPE, MAE, MaxAPE, and MaxAE, in table III) are smaller in our model, when compared to the alternatives. For example, the mean number of change requests per month in this time series is around 2795. The MAPE (mean absolute percentage error) value of around 11.5% is significantly better than those obtained with the other models, which range from around 13.7% to 19.6%. The same general conclusion can be drawn with all the distance measures used in this comparative study.

More generally, we conclude that time series analysis is a useful technique for analyzing the evolution of the change requests over a relatively long period, in this case of 12 months. It allows illustrating that evolution through a visual representation, such as the one presented in figure 14, in the series marked as *Observed*, confirming its usefulness to study the past evolution of software systems [11, 22]. Time series analysis is also an approach for predicting the evolution (in this case of change requests) that is particularly well suited for timely planning: we were able to make reliable forecasts with up to one year in advance, concerning the number of future change requests. The inclusion of seasonal information proved to have a significant impact on the suitability of the prediction model. The model with seasonal information outperformed the other four non-seasonal models in this comparative study.

B. *Validity threats*

One of the assumptions of this work is that all change requests are made through the official tracking system. In other words, we assume that there is no informal communication of change requests. If present, the informal communication of change requests would have a confounding effect. However, informal requests have been shown to have a small weight among developers who are geographically separated [23], as is the case with the Eclipse project, so this potential threat is mitigated. A related threat concerns the potential existence of duplicate change requests in the system (i.e., different requests for the same change). We use the change requests as available in the repository. With over 270000 change requests, we consider this threat to be mitigated, as existing duplicates would not compromise the overall results of this work. This observational study is performed on a single open source project, which is a threat to the external validity of the results obtained here. Nevertheless, Eclipse is often studied precisely for being a good representative of a long-term open source software project. As discussed in section II, the results obtained with our model are consistent with those obtained in other contexts [9]. This increases our confidence on their external validity, but also paves the way for differentiated replication studies. To facilitate such replication studies, we provide an experimental package containing both the change requests per month and the SPSS script for building the ARIMA(1,1,0)(1,1,0) model described in this paper. This material can be found at: http://ctp.di.fct.unl.pt/QUASAR/Resources/Papers/2011/CSMR2012Data.zip

C. *Inferences*

We were able to characterize and forecast the evolution of change requests using a time-series analysis approach. The model presented here is expected to be valid for future releases of Eclipse, as long as no dramatic changes are made into its evolution process. Extrapolating this model to other software systems evolution requires further analysis, but the approach itself is usable in different contexts. For instance, it has been successfully used in the context of IT service management of a large commercial software vendor [9]. The details of the model may vary, depending on the life cycle of each software product, which induces different trends and seasonal patterns.

Concerning trends, after the initial growing trend until 2006, Eclipse reached a stable stage, in part due to its plug-in based nature. As new functionalities are often added through plug-ins, this keeps the main project's number of requests relatively stable. Other projects with a different architectural structure may exhibit different trends, e.g. increasing requests. With respect to seasonality, in Eclipse we found an annual pattern that reflects the release schedule of new Eclipse versions. Different release schedules are likely to lead to different seasonal effects.

The approach discussed in this paper is only successful in the presence of reliable historic data. It is generic in the sense that it subsumes the case where no seasonal pattern is found: in that case, *ps*, *ds* and *qs* are set to 0, leading to an ARIMA model without seasonal information.

VI. Conclusions and Future Work

The prediction of the number of post-release bug reports for one given release has been well addressed, but the more challenging long-term prediction of the overall number of change requests – which is a more comprehensive indicator for planning the necessary human and technical resources – has been largely neglected. In addition, most existing research ignores the seasonal work patterns that many software projects follow, due to their regular releases. Past research shows that ARIMA models are adequate for short-term predictions, without considering seasonality. In this paper we investigate whether an ARIMA model could be constructed to adequately predict the long-term fluctuation of all change requests for a project having seasonal patterns.

We use Eclipse as a representative case study of projects with regular releases and with sufficient historical data. We first checked that the time series analysis approach we adopted could indeed detect the seasonal patterns and any evolution trends in the project's history. Those intermediate results informed our choice of the non-seasonal and seasonal ARIMA parameters. We trained the resulting model on 7 years of Eclipse's history and applied it to predict a full year of evolution. We did the same with 4 other non-seasonal ARIMA models. A variety of tests provided evidence that our model is statistically significant and outperforms the non-seasonal models.

The actual ARIMA parameters have to be chosen for each project at hand. We have explained all the necessary modeling and analysis steps and provided the statistical scripts so that managers and other researchers can more easily adopt this approach for their own projects.

This research provided a further step towards more comprehensive and more useful evolution predictions, by showing that an ARIMA model can provide an adequate long-term prediction of the overall change requests, which can be substantially improved by taking seasonal patterns into account.

REFERENCES

[1] T. Hill and P. Lewicki, *STATISTICS: Methods and Applications*. Tulsa, OK: StatSoft, Inc., ISBN: 978-1884233593, 2007.

[2] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), ICSE Workshops 2007,* Minneapolis, Minnesota, USA, 2007, doi: 10.1109/PROMISE.2007.10.

[3] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," Proc. Ninth international workshop on Principles of software evolution IWPSE'07, in conjunction with the 6th ESEC/FSE joint meeting, ACM, Dubrovnik, Croatia, 2007, pp. 11-18, doi: 10.1145/1294948.1294953.

[4] M. Kläs, H. Nakao, F. Elberzhager, and J. Münch, "Support planning and controlling of early quality assurance by combining expert judgment and defect data - a case study," *Empirical Software Engineering,* vol. 15, No. 4, pp. 423-454, 2010, doi: 10.1007/s10664-009-9112-1.

[5] U. Raja, D. P. Hale, and J. E. Hale, "Modeling software evolution defects: a time series approach," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 21, No. 1, pp. 49-71, 2009, doi: 10.1002/smr.398.

[6] B. Kenmey, G. Antoniol, and M. Di Penta, "Trend Analysis and Issue Prediction in Large-Scale Open Source Systems," in *12th European Conference on Software Maintenance and Reengineering (CSMR 2008),* 2008, pp. 73-82, doi: 10.1109/CSMR.2008.4493302

[7] A. Hindle, M. W. Godfrey, and R. C. Holt, "Mining Recurrent Activities: Fourier Analysis of Change Events," Proc. 31st International Conference on Software Engineering - Companion Volume (ICSE'2009), IEEE Computer Society, Vancouver, Canada, May, 2009, pp. 295-298, doi: 10.1109/ICSE-COMPANION.2009.5071005

[8] J. Śliwerski, T. Zimmermann, and A. Zeller, "When Do Changes Induce Fixes?," *ACM SIGSOFT Software Engineering Notes,* vol. 30, No. 4, pp. 1-5, May, 2005, doi: 10.1145/1083142.1083147.

[9] J. Caldeira, "Informational Technology Service Management: an Experimental Approach towards IT Service Prediction," MSc., Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2009

[10] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Forecasting the number of changes in Eclipse using time series analysis," Proc. Fourth International Workshop on Mining Software Repositories (MSR'07), IEEE Computer Society, Minneapolis, USA, May, 2007, pp. 32-33, doi: 10.1109/MSR.2007.10.

[11] T. Mens, J. Fernández-Ramil, and S. Degrandsart, "The Evolution of Eclipse," Proc. 24th IEEE International Conference on Software Maintenance, ICSM 2008, IEEE Computer Society, Beijing, China, September/October, 2008, pp. 386-395, doi: 10.1109/ICSM.2008.465808.

[12] M. M. Lehman and L. A. Belady, *Program Evolution: Processes of Software Change*: Academic Press Professional, ISBN: 0-12-442440-6, 1985.

[13] M. Wermelinger, Y. Yu, A. Lozano, and A. Capiluppi, "Assessing architectural evolution: a case study," *Empirical Software Engineering,* vol. 16, No. 5, pp. 623-666, June, 2011, doi: 10.1007/s10664-011-9164-x.

[14] G. Antoniol, G. Casazza, M. Di Penta, and E. Merlo, "Modeling clones evolution through time series," Proc. IEEE International Conference on Software Maintenance (ICSM'2001) IEEE Computer Society, Florence, Italy, November, 2001, pp. 273-280, doi: 10.1109/ICSM.2001.972740.

[15] W. H. Kruskal and W. W. Allen, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association,* vol. 47, No. 260, pp. 583-621, December, 1952.

[16] A. R. Jonckheere, "A test of significance for the relation between m rankings and k ranked categories," *British Journal of Statistical Psychology,* vol. 7, No. 2, pp. 93-100, 1954, doi: 10.1111/j.2044-8317.1954.tb00148.x.

[17] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association,* vol. 62, No. 318, pp. 399-402, June, 1967.

[18] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika,* vol. 52, No. 3/4, pp. 591-611, December, 1965, doi: 10.1093/biomet/52.3-4.591.

[19] S. Anderson, A. Auquier, W. W. Hauck, D. Oakes, W. Vandaele, and H. Weisberg, *Statistical methods for comparative studies: techniques for bias reduction*: John Wiley and Sons, ISBN, 1980.

[20] R. McCleary and R. Hay, *Applied time series analysis for the social sciences*: SAGE Publications, ISBN, 1980.

[21] G. M. Ljung and G. E. P. Box, "On a measure of lack of fit in time series models," *Biometrika,* vol. 65, No. 2, pp. 297-303, January, 1978, doi: 10.1093/biomet/65.2.297.

[22] M. Wermelinger, Y. Yu, and A. Lozano, "Design Principles in Architectural Evolution: a Case Study," Proc. International Conference on Software Maintenance (ICSM'2008), IEEE Computer Society, Beijing, China, September 28 - October 4, 2008, pp. 396-405, doi: 10.1109/ICSM.2008.4658088

[23] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An empirical study of global software development: distance and speed," in *23rd International Conference on Software Engineering (ICSE'01)*, Toronto, Ontario, Canada, 2001, pp. 81-90.