# Analysis and Prediction of Design Model Evolution Using Time Series

Hamed Shariat Yazdi[1], Mahnaz Mirbolouki[2], Pit Pietsch[1], Timo Kehrer[1], and Udo Kelter[1]

[1] Software Engineering Group, University of Siegen, Germany
{shariatyazdi,pietsch,kehrer,kelter}@informatik.uni-siegen.de
[2] Department of Mathematics, IA University of Shahre-Rey, Iran
m.mirbolouki@srbiau.ac.ir

**Abstract.** Tools which support Model-Driven Engineering have to be evaluated and tested. In the domain of model differencing and model versioning, sequences of software models (model histories), in which a model is obtained from its immediate predecessor by some modification, are of special interest. Unfortunately, in this application domain adequate real test models are scarcely available and must be artificially created. To this end, model generators were proposed in recent years. Generally, such model generators should be configured in a way that the generated sequences of models are as realistic as possible, i.e. they should mimic the changes that happen in real software models. Hence, it is a necessary prerequisite to analyze and to stochastically model the evolution (changes) of real software systems at the abstraction level of models. In this paper, we present a new approach to statistically analyze the evolution of models. Our approach uses time series as a statistical method to capture the dynamics of the evolution. We applied this approach to several typical projects and we successfully modeled their evolutions. The time series models could predict the future changes of the next revisions of the systems with good accuracies. The obtained time series models are used to create more realistic model histories for model versioning and model differencing tools.

## 1 Introduction

Tools, algorithms and methods in the context of Model Driven Engineering (MDE) have to be evaluated and tested. In this regard test models, which are concrete artifacts, should be used in order to test various desirable aspects such as correctness, quality, efficiency and scalability. The problem is that real test models are scarce and the available ones usually lack the properties which are required by the tests. The only solution is to employ artificial test models and to this end, test model generators have been proposed recently. To deliver a test model, a base model (which can be an empty or a non-empty model) is usually manipulated using a set of defined edit operations [13]. The manipulation step

L. Iliadis, M. Papazoglou, and K. Pohl (Eds.): CAiSE 2014 Workshops, LNBIP 178, pp. 1–15, 2014.

and the definition of the edit operations are done in a way that the resulting model(s) meets the requirements of the given testing scenario.

In the domain of model differencing [9] and model versioning systems [1], sequences of test models are of great importance and interest. By sequences, we mean revisions $r_1, r_2, \cdots, r_n$ of test models, where each revision is obtained from its immediate predecessor by applying some changes in terms of edit operations. The generated sequences should be "realistic" i.e. they should mimic the "real evolution" that we observe in real software models, otherwise the generated test models are of little value. By evolution, we mean the amount of changes that happen between subsequent revisions of software models. To achieve the goal of creating realistic sequences of test models, we need to (i) properly capture the "evolution" at the abstraction level of design models (class diagrams) in real software systems (ii) stochastically model the evolution (changes) and (iii) finally reproduce the real evolution in the generated sequences.

This paper particularly addresses the steps (i) and (ii). In this regard, we have to answer questions like: Is there time dependency in changes at the abstraction level of design models or the amount of changes between revisions of models are not correlated? If correlation exists, how can we mathematically model it? We prove that there is time dependency in observed changes between design models and the amount of change in one revision is dependent to the previous amount of changes. We show that ARMA time series models are capable of stochastically model such evolution. The results of this work are directly used in our test model generator called the SiDiff Model Generator [12] for producing realistic sequences of test models for MDE tools. Additionally, we will show that the presented time series models can be used for the prediction of future changes in design models of software systems with quite good accuracies.

We believe that the prediction power provides an effective method for planning the evolution of software systems as well as for maintenance of resources; when a large number of changes is predicted, more time and budget can be allocated to a project and testing efforts can be adjusted accordingly. Thus, the proposed time series models can be effectively used in this regard.

The rest of the paper is structured as follows. Section 2 presents how we capture the evolution of design models, while Section 3 explains how the projects that were used for our analysis were selected. Section 4 introduces the time series methodology to stochastically model the evolution of design models in software systems. Section 5 explains how we transform our raw data to make them appropriate for our time series analysis. We also discuss the detail of our analysis there and we show that ARMA models are quite successful to capture the evolution as well as to predict the future changes in design models of software systems. Threats to the validity of our work are discussed in Section 6 and related work is reviewed in Section 7. The conclusion and the summary of our work is provided in Section 8.

## 2 Usage of Design Models to Capture the Evolution of Java Systems

State-of-the-art approaches to understand the evolution of software systems are based on software metrics and similar static attributes. For example, research which is focused on the growth behavior of software systems is often based on software metrics reflecting the size of a system, e.g. SLOC (Source Lines of Code). In such cases the extent of the changes between revisions (or versions, depending on the granularity of the research) of a software system is expressed as a difference between values of metrics (e.g. see [19]). All further analyses are based on these differences. Unfortunately, such approaches reflect the dynamics of changes in a software system neither completely nor correctly. For instance, if we use the static metric *Number of Classes* (NOC) of a software system and if we observe an increase of 2 of this metric between two subsequent revisions, the actual amount of change might be much larger e.g. 4 existing classes were deleted, 6 new classes were added and 3 classes were moved to other packages. Thus, differences between values of metrics often underestimate the actual amount of change.

This error can be avoided by computing a precise specification of all changes between two revisions, i.e. a **difference**, and by computing metrics of this difference [23]. In our above example we would use the difference metrics *NOC-Deleted*, *NOC-Added* and *NOC-Moved* in which we get 13=(4+6+3) changes in total rather than an increase by 2 of the NOC metric. In other words, to fully understand the evolution of a system in detail, one has to count the occurrences of edit operations that have been applied between subsequent revisions.

To successfully implement the above approach, the differences should be appropriately derived. Obviously, textual differences consisting of insertions and deletions of lines of source code will not be a basis for computing meaningful difference metrics and we have to consider the semantic of the system into account. Thus, we reverse-engineered a set of carefully selected open-source Java systems into their design models.

The meta model which represents this design level information is similar to the class diagrams. The core of the meta model is depicted in Fig. 1. A revision is represented by the root element of type *Project*. Each project can contain several *Packages*. Packages can contain other packages, *Classes* and *Interfaces*. Both interfaces and classes, can contain *Methods* and *Constants*, while a class can furthermore contain *Fields*. Finally, methods can contain *Parameters*. For the sake of more readability, seven element types were omitted from Fig. 1. These elements represent specific constructs used by the Java language. The detailed description of the full meta model is presented in [16].

Having design models at hand, we can compare them using a model differencing tool. A survey of the existing approaches and tools for the problem of model comparison and difference derivation is given in [9]. Many model comparison approaches rely on persistent identifiers to trace model elements in subsequent revisions. However, reverse engineered models do not have persistent identifiers, and there are no other known information about the applied edit operations.

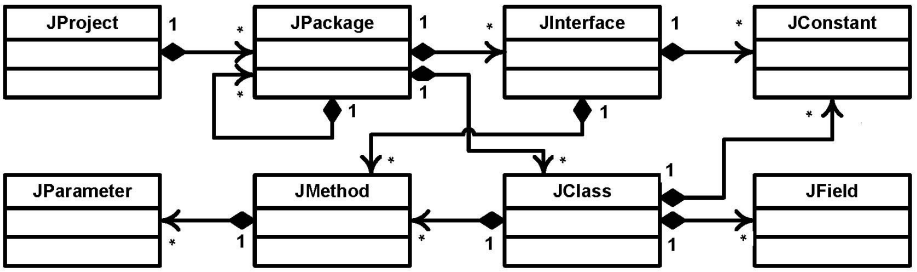


**Fig. 1.** (Simplified) Meta Model for Design Models of Java Source Code

Thus, we used a similarity-based model comparison engine known as the SiDiff model differencing framework [7] to compute differences. This generic matching engine was carefully adapted to handle our design models. The matching engine basically produces a set of correspondences between elements, which is called a *matching*. Model elements not reported in the matching are considered to be created or deleted between the compared revisions. SiDiff defines five different kinds of edit operations (changes) that might have been applied between two subsequent revisions of $r_i$ and $r_{i+1}$: **Additions:** An element is inserted in $r_{i+1}$. **Deletions:** An element is removed from $r_i$. **Moves:** An element is moved to a different position, i.e. the parent element is changed in $r_{i+1}$. **Attribute Changes:** An element is updated in $r_{i+1}$, e.g. the name or visibility of a class is changed. **Reference Changes:** A reference of an element is changed, e.g. a method has a different return type in $r_{i+1}$.

For each element type $ET$ (see Fig. 1) in our design model and for each edit operation $OP$ defined as above, the corresponding **difference metric**, $DM$, is the count of occurrences of $OP$ on all model elements of the type $ET$. The difference metrics represent the structural changes between two models. Since our model representation consists of 15 element types ($ET$s) and we distinguish 5 edit operations ($OP$s), a total of 75=(15*5) different difference metrics ($DM$s) were counted. Thus, between two revisions of design models, the changes are reflected in terms of 75 difference metrics. For design models of a software system, we compute the total number of changes between each two subsequent revisions, as the sum of all difference metrics between them. In this way we obtain a sequence of total numbers of changes. Such sequences are our data sets and our future statistical analyses of evolution will be based on them.

## 3 Sample Project Selection

For comprehension of evolution of software models and for generalizing the results of our statistical analysis, it is required that we appropriately select our sample space. Therefore, we defined three properties for selection of a project: (P1) The project must be an open-source Java system which is actively used by

end users. (P2) The project must be or have been actively developed over a long enough period of time. (P3) For each revision, its design model representation must consist of at least 100 elements, preferably more. These requirements exclude small and trivial projects. We found out that the Helix Software Evolution Data Set (HDS) [20] has 55 projects which meet our requirements.

Eight projects were randomly selected from the HDS. All revisions of each project were checked out from the respective repositories. Then, a parser was used to generate the design model representation of each revision according to the meta model in Fig. 1. If the design models of two consecutive revisions were equal, the second model was ignored. Such cases occurred if only documentation was changed or if the changes affected only the code within a method; i.e. parts of the system which are not represented in the design models. In total 3809 different design models were created. Columns 1-3 of Table 1 contain basic information about the selected projects, respectively the name of the project, mean of the number of elements in models and the number of observations (revisions).

## 4 Time Series

In this section we provide the theoretical background which is required by our time series analysis. *Time Series* (TS) are sequential measurements or observations of a phenomenon of interest through time [3]. The time dependency of the data gives time series a natural ordering which is used to study them. Having $N$ successive observations at times $t_1, \ldots, t_N$, the TS is denoted by $x = x_{t_1}, \ldots, x_{t_N}$.

In order to extract useful information and statistics out of the data and to discover its underlying dynamics, TS methods are employed. These methods are also used in order to forecast the future of the system. In TS methods, it is usually assumed that the data obey some underlying parametric stochastic processes (i.e. a sequence of random variables) and the parameters are estimated in order to describe the dynamics of the system. Principally it is assumed that an observed time series of $x_1, x_2, \ldots, x_N$ is a sample realization from an infinite population of such time series that could have been generated by such stochastic process $\{X_1, X_2, \ldots, X_k, \ldots\}$ [3][1].

In the study of TS, it is generally assumed that the underlying stochastic process is weakly stationary. Roughly speaking, a process is weakly stationary when there is no change in the trend of the data and also there is no change in the variance of the data through time, i.e. in the TS plot, the data are fluctuating around a fixed level with constant variation. Formally, a stochastic process is *Weakly Stationary* when it is mean and covariance stationary [4]. The variance stationary property is resulted from the covariance stationary as its special case.

There are some methods for transforming non-stationary processes to stationary ones in order to stabilize mean and variance, e.g. trend elimination methods, differencing, seasonal differencing, logarithm and power transformations, etc.

[1] When the underlying stochastic process is identified, this property is used by our test model generator [12] in order to create many number of model histories all with the same stochastic behavior.

### 4.1 ARMA Models

*Autoregressive Moving Average* models (ARMA) have been successfully used to analyze and stochastically model weakly stationary TS [3]. There, the behavior of the system is assumed to be dependent on: (i) previous observations and (ii) random disturbances in the past. Let $\{x_t\}$ be a weakly stationary TS with mean of $\mu$ and let $\tilde{x}_t = x_t - \mu$. Using $ARMA(p, q)$, with $p$ degree of dependency to the past observations and $q$ degree of dependency to the past disturbances, the current state of the model, i.e. $\tilde{x}_t$, is defined by:

$$\tilde{x}_t = \sum_{i=1}^{p} \phi_i \, \tilde{x}_{t-i} + a_t + \sum_{i=1}^{q} \theta_i \, a_{t-i} \tag{1}$$

where $\tilde{x}_{t-i};\ i = 1, \ldots, p$ are the past observations, $a_{t-i};\ i = 1, \ldots, q$ are the past disturbances and $a_t$ is the current disturbance. $\{a_t\}$ is a *White Noise* process i.e. a weakly stationary process with mean 0 and constant variance $\sigma_a^2$ and no correlation [4]; additionally when $\{a_t\}$ is normally distributed it is called *Gaussian White Noise*. By definition, we have $\phi_p \neq 0$ and $\theta_q \neq 0$. Equation (1) has $p + q + 2$ unknowns that should be estimated from data: $\phi_i$'s, $\theta_i$'s, $\mu$ and $\sigma_a^2$ [3].

### 4.2 Methodology for Time Series Modeling

The methodology for TS modeling can be described by the following steps [11]:

**Phase I - Model Identification**

M1) *Data Preparation:* In this step, the original data should be investigated for adequacy of use in TS analysis. Data should be transformed into weakly stationary, by possibly removing trend for mean stabilization and also possibly some transformation for stabilizing the variance.

M2) *Model Selection:* The degrees of the $ARMA(p, q)$, i.e. $p$ and $q$, should be estimated by investigating the *Autocorrelation* (ACF) and the *Partial Autocorrelation* (PACF) functions of the transformed data. In this step some candidates for $p$ and $q$ are suggested by examining the plots of the functions. Let $S_{AR}$ and $S_{MA}$ be the sets of values suggested for $p$ and $q$ respectively. The set of candidate models are created as [11]:

$$M = \{ARMA(p, q) \mid (p, q) \in S_{AR} \times S_{MA} - \{(0, 0)\}\}.$$

A more accurate approach is considering $S_{AR} = \{0, \ldots, p_{max}\}$ and $S_{MA} = \{0, \ldots, q_{max}\}$. Values of $p_{max}$ and $q_{max}$ can be specified by the user or can be estimated using the ACF and the PACF.

**Phase 2 - Model Estimation and Testing**

M3) *Estimation:* Getting $M$ as the set of candidate models, $p+q+2$ parameters for each model should be estimated (Sec. 4.1) e.g. by using the *Maximum Likelihood Estimation* (MLE) method [3].

M4) *Diagnostics:* The residuals of candidate models should be examined to decide if they are appropriate candidates or not. Ideal residuals are Gaussian white noise with almost no correlation [4] (Sec. 4.1).

*Anderson-Darling Test* can be used to check the normality of residuals. The ACF and the PACF of the residuals as well as *Ljung-Box Test* are used to test, if there are correlation between the residuals or not [18].

If a model is adequate in this respect, it will remain in $M$. If $M = \{\}$ we can go to Step M2 and try to find new candidates or stop and concluding that there is no TS model capable of describing the dynamics of the data.

M5) *Best Model Selection:* If there is more than one candidate in $M$, we will have some suitable models to describe the dynamics of the data. The superior model in $M$ is the one with lower *Akaike Information Criterion* (AIC). AIC rewards the goodness of fit and accuracy and penalize the overfitting or complexity of models [18,3].

**Phase 3 - Applications**

M6) *Forecasting and Simulation:* The best model can be used to predict the future of the system or it can be used to simulate the dynamics of the system and produce infinite similar time series all obeying the same characteristics. More information about forecasting and their calculation as well as their errors rates and bands can be found in [3,4].

### 4.3 Accuracy of Forecasts

Due to high volatility in our observations (see e.g. Fig. 2 in Sec. 5), the relative error is not suitable for assessing the accuracy of our TS forecasts. The same problem is also reported in [26] when predicting highly volatile Internet traffic. A remedy similar to ours is proposed there as well. In this section we describe the appropriate way of assessing the forecasts' accuracy.

Let $x$ be an observation and $\hat{x}$ be the forecast of $x$. The relative error, in percent, is defined by $\delta_x = |x - \hat{x}|/|x| \times 100$ where $x \neq 0$. To clarify the problem, now suppose that $x = 1, \hat{x} = 2$ and $y = 100, \hat{y} = 101$ then $\delta_x = 100\%$ and $\delta_y = 1\%$ although for both the absolute error is 1. Therefore, in our case, the Mean Relative Error (MRE) [11] which is defined as $MRE = \frac{1}{N}\sum_{i=1}^{N} \delta_{x_i}$ is not a good interpretation for error rates of forecasts.

To solve this, *Normalized Relative Error* (NRE) is defined. Suppose that $X$ is the set of possible outcomes for $x$ and $x$ is approximated by $\hat{x}$, then the NRE, in percent, is defined as:

$$\eta_x = \frac{|x - \hat{x}|}{max\,(X) - min\,(X)} \times 100 \tag{2}$$

Similarly, let $X$ be the set of all possible outcomes for observations $x_1, \ldots, x_N$ and $x_i$ is approximated by $\hat{x}_i$. The *Normalized Mean Squared Error* (NMSE) is defined by [10]:

$$NMSE = \frac{1}{N}\sum_{i=1}^{N} \frac{(x_i - \hat{x}_i)^2}{(max\,(X) - min\,(X))^2} \tag{3}$$

Equations (2),(3) are used to assess the quality of TS forecasts in Sec. 5.

## 5 Time Series Models of Changes

As discussed in Sec. 2, we distinguish 5 types of changes, namely additions, deletions, moves, reference changes and updates, for all 15 different model element types. In each difference between two revisions of a software system, the total number of changes reflects the extent of evolution of the system.

We use the sequence of the total numbers of changes in each revision as the basis of our analysis. Although in the theory of TS analysis, the steps M4 and M5 in Sec. 4.2, i.e. the "diagnostics" and the "best model selection", are enough to use a TS model in practice, we additionally demonstrate the validity of our approach and prove the goodness of the proposed TS models, by partitioning our measured data, $\{x_1, \ldots, x_N\}$, into two disjoint subsets. The first set, called the "base-set", consists of the observations of $x_1$ up to $x_{N-3}$. The second one, which is called the "hold-out set", consists of the last three observations. We try to estimate our TS models on the base-set. The fitness of the proposed models is then checked by comparing their three step forecasts with the actual observations in the hold-out set. The more the predictions and the actual observations are close together, the better the model is for simulation and forecasting.

For our analyses, we used the Wolfram Mathematica® 9.0.1 computational engine and in what follows, we will take the ASM project as the exemplary project for describing the details of our approach. In total, we had 259 observations for the ASM project. The base-set of this project consists of the first 256 observations, the hold-out set contains the observations 257, 258 and 259.

Figure 2 shows the total number of changes for ASM, observed in its base-set. The data show high degree of volatility and both big and small changes are observable. For the sake of comprehension in Fig. 2, we limited the top of the y-axis to 130, but we do have many bigger changes which are not visible there e.g. the two biggest peaks with 1973 and 956 changes, respectively at 75th and 189th observations. Additionally, we observe time periods in which more changes occur in comparison to other time periods. Such a behavior is a significant obstacle for applying some simpler statistical methods such as regression analysis. Since the variance of the data is not stable and the process is not stationary, ARMA models can not be applied and we first have to make the data weakly stationary by suitable transformations (Step M1 in Sec. 4.2).

We used the *Box-Cox Transformation* (BCT) for stabilizing the variance[2] of the data [22,14]. BCT has a transformation parameter ($\lambda$) which is obtained by maximizing the logarithm of the likelihood function of the data. For stabilizing the mean of data and for removing its trend, *differencing* can be used in which instead of the original series, $\{x_i\}$, the series $\{x_i - x_{i-1}\}$ is used (Sec. 4).

All of our sample projects were successfully transformed into weakly stationary by subsequently applying the following three transformations: (T1) A Box-Cox transformation using the optimum $\lambda$ [3]. (T2) A Difference transformation of degree "one". (T3) The series mean of the previous step is subtracted.

[2] Alternative, but in our work less suitable methods is the logarithmic transformation.
[3] For each project, the optimum value of $\lambda$ is given in the website of the paper at [16].

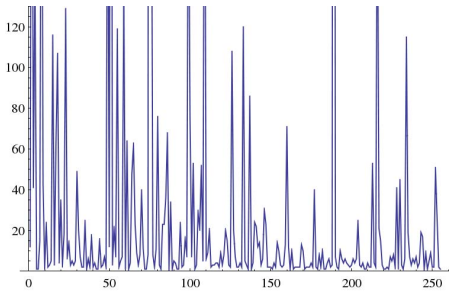


**Fig. 2.** ASM project - Total number of changes in the hold-out set. For better comprehension, the y-axis is limited to 130 and bigger values are not visible.

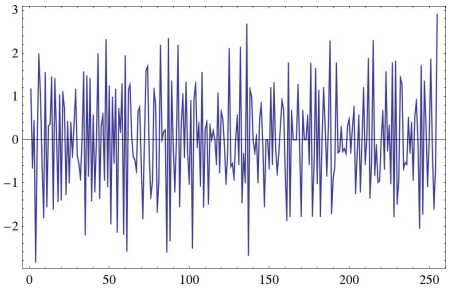


**Fig. 3.** ASM project - Fully transformed series using T1 to T3 transformations

In our exemplary project, Fig. 3 shows the transformed series after application of the three transformations. Apparently, the variance is more stable and the data are fluctuating around the mean of zero. Thus, the transformed series is stationary and meets the requirements of the ARMA process.

As mentioned earlier in Sec. 4.2 (Step M2), investigating the ACF and the PACF of a TS suggests some possible degrees for $p$ and $q$ in $ARMA(p, q)$. But here, in order to prove the applicability of our approach we deliberately used the more accurate way and we chose $p_{max} = q_{max} = 9$. The choice of the value 9 is due to the fact that for all projects, there is no significant correlation after the 9th lag in their ACFs and PACFs. Thus, $S_{AR} = S_{MA} = \{0, \ldots, 9\}$ and the set of our candidate model, $M$, consists of 99 ARMA models.

For all 99 ARMA models considered for each project, first their parameters were estimated using the MLE method and later their residuals were calculated[4]. If the residuals of a model is Gaussian white noise with almost no correlation, it stays in the candidate set $M$ (Step M4 in Sec. 4.2). For normality of the residuals we used the Anderson-Darling test with the significance level of 0.01.

At first, we disregarded the models with non-normal residuals. Then we discarded the models with significant correlations in residuals. We used the ACF and the PACF considering the first 40 lags and the 0.95 confidence band. In the end we came up with 10 possible candidate models[5] for the ASM project.

To select the best model out of our candidate set, we additionally used the Ljung-Box test. The test is designed to check if the residuals are correlated or not. $\mathcal{H}_0$ states that data are independently distributed i.e. there is no correlation between them and $\mathcal{H}_1$ declares that they are not independently distributed. The number of lags which should be examined affects the power performance of the test. The minimum number of lags is usually approximated by $ln(N)$ where $N$

[4] For each project, the website of the paper [16] contains estimated parameters of all ARMA models, their residual and error bands.

[5] ARMA(1,4), ARMA(2,1), ARMA(2,3), ARMA(2,4), ARMA(3,2), ARMA(3,5), ARMA(3,7), ARMA(6,1), ARMA(6,2) and ARMA(6,4).

is the length of observations [18][6]. For each model, we therefore considered the first 40 lags and calculated the p-value of the test for each of the lags. We then counted the number of lags with p-value less than our significance level of 0.05, i.e. those for which the null hypothesis is rejected.

ARMA(3,5) and ARMA(6,4) had 4 and 9 lags with p-values less than our significance level respectively, so they were less competent and both were no longer considered. Other candidate models are equally good in this respect since none of their lags has a p-value less than our significance level. For the rest of 8 candidates, we checked their AICs and the model with smallest AIC was our final best model (Sec. 4.2, Step M5). We therefore selected ARMA(2,3)[7] as our best model for the ASM project.

In order to additionally prove the usability of our best model, we tried to compare the actual observations in the hold-out set with the three step forecasts of the model. The forecasts can be transformed back to their original form by reversing the order of three steps transformations (T1 to T3) and employing the inverse of the Box-Cox transformation. This respectively results in 5.96, 5.56 and 5.17 as forecasts of the hold-out set of the ASM project.

Since our aim is to forecast "changes" and we assume that there should be some changes to predict, we can round up our real valued forecasts to integers using the ceiling function i.e. $\lceil x \rceil$. Thus, we have 6, 6 and 6 as our final forecasts which is quite close to real observations of 5, 4 and 1 (see Table 1 for the NREs). Therefore, the suitability of our ARMA(2,3) model for capturing the dynamics of design model evolution of the ASM project is approved (see Step M5 in Sec. 4.2).

Generally it is shown that the errors are accumulated for k-step forecasts of ARMA models lose their prediction power as well as their accuracy quickly for farther predictions [3]. Table 1 shows the best chosen model for each project with their three step forecasts as well as the actual number of changes observed in their hold-out sets. The last column of the table (**#LB Lags**) contains the number lags out of 40 considered lags[8], for which their p-value of the Ljung-Box are less than our significance level.

In Table 1, the error rates of forecasts, i.e. NRE and NMSE, are given. They are calculated based on (2) and (3) in which they require the true max of observed changes for each project. For each of NRE and NMSE, first we calculated the errors based on the true max of observed changes as suggested by the formulas. Additionally since very big changes are not very frequent, we tried to be more conservative and used the 0.98-quantile (Q98) of observations instead of the true max in (2) and (3). The reason for choosing Q98 is the fact that it can properly disregard, for all projects, the less frequent big changes. Alternatively, one can exclude big changes using clustering techniques and calculate the max on the rest. Therefore, in Table 1 we report two sets of error rates for NRE and NMSE. The first sub-row shows the error rates calculated based on the true

[6] For the CheckStyle project which has the most number of observations, we have $N = 1010$ and $ln(N) \approx 6.918$.

[7] Parameters: $\phi_1 = 0.8207$, $\phi_2 = -0.6621$ & $\theta_1 = 1.7559$, $\theta_2 = 1.3910$, $\theta_3 = -0.5903$.

[8] For DataVision, we had 28 observations. So instead of 40 lags, we considered 28.

max of observations and the second sub-row delivers the results of our more conservative approach using Q98. For example in the ASM project, using true max, NRE and NMSE are $0.05\%, 0.10\%, 0.25\%$ and $2.5715 \times 10^{-6}$ respectively. When we use Q98 we get $0.37\%, 0.74\%, 1.84\%$ and $1.3616 \times 10^{-4}$. Error rates in either of two approaches are quite good typically below 3%.

Generally the error rates of forecasts for the hold-out sets are quite small which proves the appropriateness of our approach in order to capture the true dynamics of design model evolution. The only exception is the Struts project in which its last prediction is not very good. This has two reasons: first, it is the third and the farthest forecast which is the least accurate and second, 9 lags out of 40 did not pass the Ljung-Box test and the residuals are little twisted from the ideal Gaussian white noise (Sec. 4.2, Step M4). These two factors degrade the prediction power of the proposed model for its farthest forecast.

**Table 1.** All projects - Forecasts vs observations

| Project | Mean #Elem | #Obs. | Max of Obs. | Q98 of Obs. | (p,q) of Best ARMA | Obs./ Pred. | NRE: Max/Q98 | NMSE: Max/Q98 | AIC | #LB Lags |
|---|---|---|---|---|---|---|---|---|---|---|
| ASM | 3635 | 259 | 1973 | 272 | (2,3) | 5, 4, 1 | 0.05%, 0.10%, 0.25% | $2.571 \times 10^{-6}$ | −0.304 | 0 |
| | | | | | | 6, 6, 6 | 0.37%, 0.74%, 1.84% | $1.362 \times 10^{-4}$ | | |
| CheckStyle | 3181 | 1010 | 1366 | 99 | (9,5) | 9, 9, 16 | 0.22%, 0.29%, 0.073% | $4.651 \times 10^{-6}$ | −0.307 | 0 |
| | | | | | | 12, 5, 15 | 3.06%, 4.08%, 1.02% | $9.024 \times 10^{-4}$ | | |
| DataVision | 5520 | 28 | 105 | 105 | (2,1) | 2, 3, 1 | 0.96%, 0.96%, 7.7% | $2.034 \times 10^{-3}$ | −0.691 | 0 |
| | | | | | | 3, 4, 9 | 0.96%, 0.96%, 7.7% | $2.034 \times 10^{-3}$ | | |
| FreeMarker | 6895 | 339 | 851 | 189 | (4,7) | 2, 2, 2 | 0.23%, 0.0%, 0.0% | $1.845 \times 10^{-6}$ | 0.130 | 0 |
| | | | | | | 4, 2, 2 | 1.06%, 0.0%, 0.0% | $3.772 \times 10^{-5}$ | | |
| Jameleon | 3139 | 285 | 1107 | 85 | (7,5) | 12, 1, 5 | 0.54%, 1.99%, 1.36% | $2.030 \times 10^{-4}$ | 0.531 | 0 |
| | | | | | | 18, 23, 20 | 7.14%, 26.19%, 17.86% | $3.519 \times 10^{-2}$ | | |
| JFreeChart | 21583 | 366 | 599 | 244 | (9,0) | 4, 2, 32 | 0.50%, 1.34%, 3.51% | $4.791 \times 10^{-4}$ | −0.094 | 0 |
| | | | | | | 7, 10, 11 | 1.23%, 3.29%, 8.64% | $2.901 \times 10^{-3}$ | | |
| Maven | 3786 | 781 | 2735 | 346 | (8,8) | 1, 10, 1 | 0.0%, 0.15%, 0.0% | $7.135 \times 10^{-7}$ | −0.203 | 0 |
| | | | | | | 1, 6, 1 | 0.0%, 1.16%, 0.0% | $4.481 \times 10^{-5}$ | | |
| Struts | 4323 | 737 | 715 | 188 | (8,9) | 39, 5, 10 | 3.22%, 0.14%, 30.11% | $3.057 \times 10^{-2}$ | −0.134 | 9 |
| | | | | | | 16, 6, 225 | 12.30%, 0.53%, 114.97% | $4.457 \times 10^{-1}$ | | |

Due to space limitation, we have put the detailed material of our analysis on the accompanied website of our paper [16]. Estimated parameters for all of the 99 models of each project, residuals of each model, error rates, confidence bands and error bands of the models are all available there.

## 6 Evaluation and Threat to Validity

In this section we discuss the evaluation of our work and the threats to validity to this research.

**Evaluation:** The evaluation of our work boils down to the appropriateness of our approach in order to successfully capture the evolution of design models. From the TS theory perspective, correctly performing steps M1 to M5 proves the appropriateness of a TS model to capture the dynamics of the data. In order

to additionally prove the usability of the proposed TS models, as we discussed in Sec. 5, we divided our observations into the base and the hold-out sets. The usability of the TS models then additionally approved by comparing their forecasts with the hold-out sets. The forecast error rates were quite small (see Table 1) which approves the use of the proposed TS models in practice.

For the run-time evaluation, one might argue that estimating 99 ARMA models is time-expensive. As we mentioned in Sec. 4.2, the order of promising models can be estimated by investigating the ACF and PACF of the data. This will significantly reduce the number of possible candidate models to just a few. In this paper, we deliberately used 99 ARMA models to generally prove our proposed approach, but in practice much fewer models have to be investigated.

**Threat to Validity:** The first threat to validity is the accuracy in capturing the evolution of design models. As discussed in Sec. 2, difference metrics have advantages compared to static software metrics. Therefore, the evolution of design models is more accurately described by difference metrics rather than variation in values of static metrics.

The second threat is the accuracy of the calculation of difference metrics. Due to the heuristic nature of the model comparison algorithms, there is no guarantee that the established correspondences between elements are always correct. Error rates for class diagrams were calculated for the SiDiff algorithm which is used in this research [23]. These error rates were generally less than 2%. Because the model type used to represent our design models is similar to a class diagram, we expect similar error rates in our correspondence calculation. They do not have any significant effect on our results and the findings are not distorted.

The third threat is the selection of hold-out sets of length "3". As we mentioned earlier, our primary focus was to suitably capture the evolution dynamics of design models. We used the prediction power as an extra evaluation (see Sec. 4.2) for the suitability of our TS models. Therefore, to this end, a hold-out set with the length of 3 is sufficient. Additionally, it is known that for ARMA models the errors are accumulated for k-step forecasts [3] and the models quickly loose their prediction power and their accuracy. In the case that we have periodic or seasonal effect, the predictions are usually considered with a length equal to $k$ times (usually $k$=1,2) of the length of the period. Since our data does not have any periodic effect, again, hold-out sets with length of 3 are justifiable.

The fourth threat is the external validity, i.e. how our results can be generalized. Our sample set of open source Java projects are quite diverse with respect to application domains, number of revisions, number of elements etc. The process described in this paper is independent of a specific programming language, though. It can be easily applied and used to analyze the evolution of projects which are coded in a different programming language. The only two constraints are: 1) a suitable design model representation for the respective language exists and 2) there is a model differencing tool capable of comparing these models.

It is not clear if comparable results will be observed considering closed source systems, especially where specific programming discipline is practiced. Hence, closed source systems as well as other object-oriented programming languages such as C++ can be the subjects of further research.

## 7 Related Work

In our previous work [17], we studied how observed frequencies of edit operations can be statistically modeled by statistical distributions. We showed that the Waring and the Beta-Negative Binomial distributions were able to statistically model the frequencies. In that work, we did not considered the "evolution" in real model histories and no time series analysis were considered either.

The following papers used time series analysis to answer research questions in the context of software maintenance and evolution. None of these paper addresses the topic of this paper, i.e. how (design) models of software systems evolve over a long period of time.

Fuentetaja et al. [5] applied time series analysis on growth data of released versions of software systems. They were able to validate the third and eighth law of software evolution, i.e. *Self Regulation* and *Feedback Systems*. Caution is advised when considering the presented results though, because for most systems less than 30 releases were considered, the base data is therefore limited.

Herraiz et al. [6] used time series to analyze the growth of open source software system based on simple size metrics. They concluded that time series analysis is the best method to forecast the evolution of software projects. Their findings showed that design decisions and changes in the history of a project have more influence on its future development than external events.

Kenmei et al. [8] analyzed the connection between change request in large-scale open source software and lines of code to identify trends in the change patterns of large software systems. Based on the trends they were able to predict the need for the number of developers for the next 2 to 6 weeks.

Antoniol et al. [2] presented a time series based method for monitoring and predicting the evolution of clones in a software systems. At first clones were detected within a system based on software metrics, then the identified clones were modeled as time series in a second step. They were able to predict the number of clones in the next release very accurately with this method.

Wu et al. [24] used ARIMA time series models to predict the number of bugs in large software systems on a monthly basis. The results of their prediction were acceptable. Furthermore they could show in their data, that software bug number series are to a certain extent dependent on seasonal and cyclical factors.

Raja et al. [15] used time series approach to predict defects in software evolution. They used monthly defect reports for eight open source projects and built up time series models to predict and analyze software defects.

The following papers address research questions in the area of system evolution which are similar to the one addressed in this paper. However, these works did not use time series or any other advanced mathematical concepts.

Vasa et al. [21] studied the evolution of classes and interfaces in open-source Java software. They analyzed the evolution based on static metrics of released versions of the systems. Their findings suggests that the average metric values are almost stable over histories, i.e. the average size of classes does not change much between released versions. This work also analyzes which types of classes change more than average and concludes that large, complex classes change more. This work does not use time series and makes no attempt to model time-dependency. In [19] Vasa presented an extended and more detailed version of his research.

Xing et al. [25] count the occurrences of language specific edit operations and categorize change levels according to 5 specific profiles, i.e. intense evolution, rapidly developing, restructuring, slowly developing and steady-state. To identify the edit operations they use a model differencing tool known as UMLDiff. Their findings suggest that different change categories are related to different phases of the software development process. Their findings are consistent with ours in that phases of low development activity alternate with phases with high development activity (see Fig. 2). Time series analysis was not used in their research.

## 8 Summary and Conclusion

In this paper we presented a new approach to statistically analyze the time-dependency of changes in design models of open source Java software systems. This allows us to precisely describe the past evolution of a system as well as to forecast the amount of changes which is to be expected in the next revisions. We applied this approach to several software systems. The evaluation showed that our models could very accurately predict the actual amount of changes in the last 3 revisions of each system; usually with error rates as low as 3% and less. This information can be used to control the time and budget allocated for a project in the near future or to plan regression testing activities. In our own work, we use these stochastic models to configure a test model generator which generates realistic histories of models; these histories are used to test and evaluate various tools which support model driven development.

**Acknowledgment.** The authors of this paper would like to thank Prof. Lefteris Angelis, head of Statistics and Information System Group at the Aristotle University of Thessaloniki, for his valuable comments and feedback.

The work of the first author, Hamed Shariat Yazdi, is supported by the German Research Foundation (DFG) under grant KE 499/5-1.

## References

1. Altmanninger, Seidl, Wimmer: A survey on model versioning approaches. Inter. Journal of Web Information Systems 5(3) (2009)
2. Antoniol, Casazza, Penta, Merlo: Modeling clones evolution through time series. In: Proc. Inter. Conf. Software Maintenance, ICSM (2001)

3. Box, Jenkins, Reinsel: Time series analysis, forecasting and control, 4th edn. Wiley (2008)
4. Brockwell, Davis: Introduction to time series and forecasting, 2nd edn. Springer (2002)
5. Fuentetaja, Bagert: Software evolution from a time series perspective. In: Proc. Inter. Conf. Software Maintenance, ICSM (2002)
6. Herraiz: A statistical examination of the evolution and properties of libre software. In: Proc. Inter. Conf. Software Maintenance ICSM (2009)
7. Kehrer, Kelter, Pietsch, Schmidt: Adaptability of model comparison tools. In: Proc. 27th Inter. Conf. Automated Software Engineering, ASE (2012)
8. Kenmei, Antoniol, Di Penta: Trend analysis and issue prediction in large-scale open source systems. In: Software Maintenance and Reengineering (2008)
9. Kolovos, Ruscio, Paige, Pierantonio: Different models for model matching: An analysis of approaches to support model differencing. In: Proc. ICSE Workshop Comparison & Versioning of Software Models CVSM (2009)
10. Lughofer: Evolving Fuzzy Systems - Methodologies. Advanced Concepts and Applications. Springer (2011)
11. Makridakis, Wheelwright, Hyndman: Forecasting methods and applications, 3rd edn. Wiley (1998)
12. Pietsch, Yazdi, S., Kelter: Generating realistic test models for model processing tools. In: Proc. 26th Inter. Conf. Automated Software Engineering (2011)
13. Pietsch, Yazdi, S., Kelter: Controlled generation of models with defined properties. In: Proceedings of Software Engineering Conference (SE 2012), Berlin, Germany (2012)
14. Porunov: Box-Cox transforamtion and the illusion of the Normality of macroeconomics series. Business Informatics 2 (2010)
15. Raja, Hale, Hale: Modeling software evolution defects: a time series approach. Journal of Software Maintenance and Evolution 21(1) (2009)
16. Yazdi, S., Pietsch: Accompanied material and data for the APSIS 2014 paper (2014), `http://pi.informatik.uni-siegen.de/qudimo/smg/APSIS2014`
17. Yazdi, S., Pietsch, Kehrer, Kelter: Statistical analysis of changes for synthesizing realistic test models. In: Proc. of Multi-conf. Software Engineering 2013 (SE 2013), Aachen, Germany (2013)
18. Tsay: Analysis of financial time series, 2nd edn. Wiley (2005)
19. Vasa: Growth and change dynamics in open source software systems. Ph.D. dissertation, Swinburne University of Technology (2010)
20. Vasa, Lumpe, Jones: Helix - Software Evolution Data Set (2010), `http://www.ict.swin.edu.au/research/projects/helix`
21. Vasa, Schneider, Nierstrasz: The inevitable stability of software change. In: Inter. Conf. Software Maintenance, ICSM (2007)
22. Viktor: The Box-Cox transforamtion (March 2010), `http://www.mql5.com/en/articles/363`
23. Wenzel: Unique identification of elements in evolving models: Towards fine-grained traceability in MDE. Ph.D. dissertation, Uni. Siegen (2011)
24. Wu, Zhang, Yang, Wang: Time series analysis for bug number prediction. In: 2nd Inter. Conf. Software Engineering & Data Mining, SEDM (2010)
25. Xing, Stroulia: Analyzing the evolutionary history of the logical design of object-oriented software. IEEE Trans. Software Engineering 31(10) (2005)
26. Zhou, He, Sun: Traffic predictability based on ARIMA/GARCH model. In: 2nd Conf. Next Generation Internet Design and Engineering (2006)