CHAPTER I

INTRODUCTION

Two primary concerns in software release planning are: improving functionality and maintaining quality. Both objectives are constrained by limits on development time and cost. In order to respect these constraints and still pursue both objectives, the scope of planned work must be limited so that time is available to properly deal with the inevitable defects (bugs) that will arise. In this way, a software release can better ensure quality while also improving functionality.

A critical step in this planning process is to factor in a suitable amount of time for testing and bug-fixing. Otherwise, there is a risk of slip in the development schedule and/or software quality. As the time and effort required for testing and bug-fixing will likely be a function of the number of defects introduced during development, it is desirable to be able to predict how many bugs can be expected as development proceeds.

A potential application for defect prediction is to compare different release plans according to their estimated bug fallout and subsequent impact on testing and bug-fixing times. This would assist release planners in ensuring that the total development time does not exceed the project's time budget for a release. The comparison of different release plans is integral to release plan optimization, which is the focus of The Next Release Problem [2] (discussed in detail in the Motivation chapter).

Many approaches to defect prediction focus on either code analysis or historical defect information. To make the defect prediction model useful for comparing release plans, the model must be dependent in some way on the basic elements of the release

plan: planned features and improvements. The historical defect models discussed in the Literature Review chapter are limited in this respect, as they depend only on the past defects.

An approach to defect prediction is presented using a multivariate time series model. This model can be applied for a proposed release, because predictions can be made using only information about proposed features and improvements.

The paper is organized as follows. First, related work is presented in the Literature Review chapter. Then, further motivation for the use of a time series model for predicting defects is presented in the Motivation section. Next, an overview of time series modeling concepts is provided in the Background section. The methods used for data collection and preparation, and time series modeling are detailed in the Methods chapter. The results of applying these methods are then given in the Results chapter, and discussed in the Discussion chapter. After this, possible sources of invalidity are put forth in the Threats to Validity chapter, and potential avenues of future research are laid out in the Future Work chapter. The paper ends with the Conclusion chapter.

CHAPTER II

LITERATURE REVIEW

Software defect (bug) prediction typically involves a detailed analysis of code or proposed design changes. Some of these analytical methods are mentioned in the next section. These analytical approaches require more information in more detail than might be available during the software release planning stage. For this reason, alternative approaches were sought out, and several that depend on historical data and use statistical methods are discussed.

Analytical Approaches to Defect Prediction

Akiyama [1] predicted defect counts based on lines of code (LOC), number of decisions, and the number of subroutine calls. Gafney [7] likewise predicted defect count based on LOC. Rather than code itself, Henry and Kafura [10] define metrics that are based on information taken from design documents, to be used in defect prediction. Nagappan and Ball [14] use relative code churn (lines modified) as a metric for predicting the density of defects. Giger, Pinzger, and Gall [8] compare the use of code churn to a more fined-grained approach, capturing ". . . the exact code changes and their semantics down to statement level." (p. 83)

Statistical Approaches to Defect Prediction

Rather than requiring a detailed code analysis to predict defects, the approach proposed in this paper is to develop a mathematical model based on historical data on defect occurrences. Specifically, the proposed approach is to develop a defect prediction model using previous software features, improvements, and defects.

A related approach, used by Li, Shaw, Herbsleb, Ray, and Santhanam [12], is to study only the defect occurrences themselves, and attempt to develop a mathematical model for defect projection. In their work, functions were fitted to a time series of defect occurrences, and then the function parameters themselves were extrapolated for each new release. They found that the Weibull model fit best in 73% of the tested software releases. They attempted to extrapolate model parameters using naive methods, moving averages, and exponential smoothing, but found these techniques to be ". . . inadequate in extrapolating model parameters of the Weibull model for defect-occurrence projection." (p. 271) The reason given for this ineffectiveness is the changing nature of the software development system. For example, development practices, staffing levels, and usage patterns may all change between releases.

In another related approach, Graves, Karr, Marron, and Siy [9] developed several models that predict the future distribution of software faults in a given code module. The basis of their predictive models is a statistical analysis of change management data, which describes only the changes made to code files. The best model they found was a weighted time damping model, where every change in the module files contributed to defect prediction, with time-damping to account for age of changes. They achieved a performance nearly as good by basing a generalized linear model on just the modules age and the number of past changes. They also found factors that did not improve model performance: module length, number of developers making changes in the module, and how often a module is changed simultaneously with another module.

In the final approach discussed here, by Singh, Abbas, Ahmad, and Ramaswamy [15], the Box-Jenkins method is applied to datasets from the Eclipse and Mozilla software projects, which are represented as time series data, and defect count is predicted using an ARIMA model. Their modeling effort is focused at the component-level, and they conclude that ". . . current bug count of a component is linearly related to its previous bug count." (p. 6)

CHAPTER III

MOTIVATION

Release planners typically rely on both their experience and project conventions to generate a release plan by selecting planned features and improvements such that the estimated time to test for and fix defects will not cause a schedule slip.

However, if the defect estimation technique is only loosely based on past experience, as with a rule-of-thumb, then it may prove too coarse for comparing multiple release plans. Specifically, such a technique may not provide any quantitative difference between release plans that are similar (but not the same). For example, suppose two different release plans are being considered. Both include two features, but one has five improvements and the other has seven. A rule-of-thumb approach may provide the same estimate for each. Even for dissimilar release plans, such an approach still has the disadvantage of lacking confidence intervals to quantify prediction uncertainty.

An alternative approach is to develop a model that will take into account the differences in composition of features and improvements between the release plans. In this case, one would expect that the predicted number of defects would vary across the release plans and that prediction uncertainty can be quantified by confidence intervals. Such a model would assume some explanatory relationship, like that shown in Figure 1.

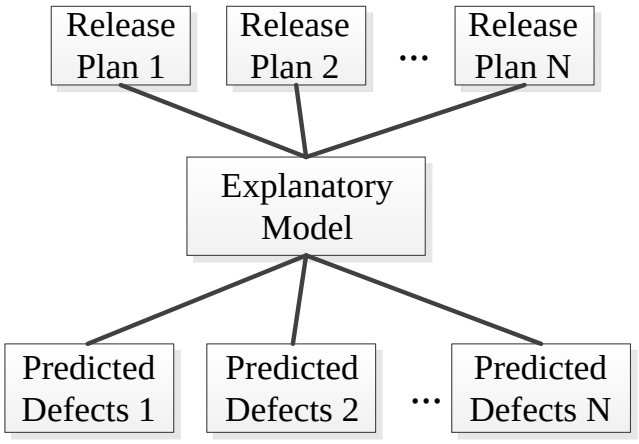

Figure 1 Using an explanatory model allows for the possibility of different defect predictions for each release plan.

A predictive model will have some inaccuracy, but confidence levels can be used to quantify the uncertainty of future prediction based on past accuracy. This will allow release planners to assess the risk of relying on the defect prediction. A higher confidence level results in less risk because it encompasses a larger window for the prediction. Conversely, a lower confidence level results in more risk and a more narrow prediction window.

The Next Release Problem

Release plan optimization is exactly the goal of The Next Release Problem [2] (NRP), but there is a gap between the abstract domain of the NRP and the detailed, messy data found in software projects. By applying an explanatory predictive model there is a path toward bridging this gap, opening up the potential for using NRP optimization techniques in real-world release planning. In this section, first the NRP is described, then the gap between it and practical planning is discussed, and finally it is shown how the explanatory model suggested earlier would be applied to help bridge this gap.

Defining the NRP

The Next Release Problem (NRP) was defined by Bagnall, Rayward-Smith, and Whittley [2], and was shown to be NP-Hard. Being abstract in its treatment of feature cost, a broad range of optimization techniques can be applied to the NRP, such as integer programming, hill climbing, simulated annealing, genetic algorithms, etc. The NRP is the subject of academic research in the area of Search-Based Software Engineering [11][17][19].

The NRP describes the situation where software project planners, who have multiple customers to satisfy, would like to maximize the revenue produced from completing the project. This is all described mathematically as follows.

A software project has a set R of all possible requirements (new features and enhancements) that might be included in the next software release. A customer i is satisfied by completing a subset $R_i \subseteq R$. The importance of a customer i is given by the weight, $w_i \in \mathbb{Z}^+$.

Requirements may have acyclic dependencies, or prerequisites, that must be completed first. A subset that includes all prerequisite requirements, recursively, is indicated by $\hat{R}_i$, and should be taken to mean

$$\hat{R}_i = R_i \cup ancestors(R_i)$$

For example, if $R_1 = \{r_2\}$, and r_1 is a prerequisite for r_2, then $\hat{R}_i = \{r_1, r_2\}$.

A requirement $r \in R$ has a cost $cost(r) \in \mathbb{Z}^+$, associated with its implementation, not considering the cost of any prerequisite requirements. Then, the cost for some subset $R' \subseteq R$ will be

$$cost(R') = \sum_{r \in \hat{R}'} cost(r)$$

Once customer i is satisfied, their weight w_i contributes to the total revenue from the project, as in

$$\sum_{i \in S} w_i$$

So, the NRP is posed as follows: for a group of n customers, select the subset $S \subseteq \{1,2, \dots, n\}$ that maximizes total revenue, while keeping the total cost within some budget constraint B. This is given by

$$maximize \sum_{i \in S} w_i$$

$$subject\ to\ cost\left(\bigcup_{i \in S} \hat{R}_i\right) \leq B$$

The Gap between Abstraction and Reality

As was discussed in the previous section, a planner would need several things to be able to implement a NRP-like optimization:

1. A set of requirements that could potentially be implemented.
2. A set of customers that are satisfied by some subset of the requirements, and have an associated weight.
3. A cost function, to quantify the cost of each requirement.
4. A cost budget that should not be exceeded.

Having all these in hand, a planner could proceed to optimize the subset of requirements planned for the next release. One difficulty with this that can be highlighted

is in the definition of a cost function. It might be suggested that the estimated time to implement a requirement alone might be used to determine cost, but there is a practical detail that prevents this: in order to maintain quality software, the total cost of any requirement should take into consideration both the cost of implementation *and* the cost of fixing associated defects. Otherwise, a release plan would appear to be within budget, when there is a risk that the budget will be exceeded when defect costs are also considered.

Bridging the Gap

We use the explanatory model to address the need to consider defect cost. Such a model, given some subset of proposed requirements, can be used to predict defects and to find additional cost which should be considered. This use of the predictive model is illustrated in Figure 2.

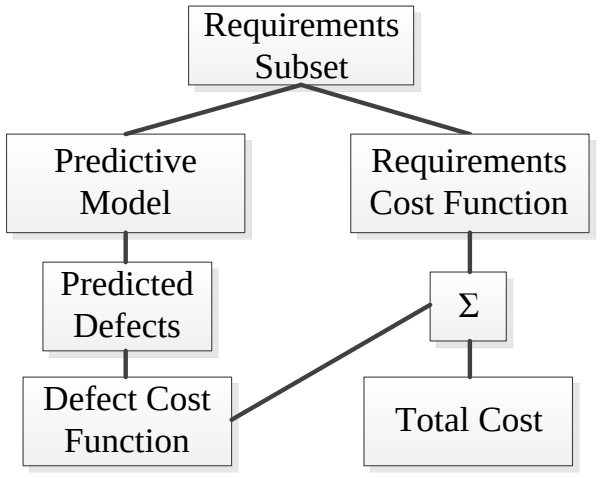

Figure 2 Defect prediction model being used to determine the overall cost of some requirements subset.

Since predictive models cannot be perfectly accurate, instead we would expect that any forecasting would include confidence levels. Taking into account the confidence of a prediction allows planners to account for risk in the use of the defect prediction. If more risk is acceptable, then planners will get a narrower prediction window, and in exchange take more of a chance that the prediction is inaccurate. A wider prediction window means, though, that when the defect prediction is used to determine requirements cost, that potential cost range will also be wider.

CHAPTER IV

BACKGROUND

In this section, time series models are introduced, and then further concepts related to modeling, exogeneity and stationarity, are discussed.

Time Series Models

A time series is a collection of observations that occur in order. The process underlying a time series is assumed to be stochastic, so the model must correspondingly be probabilistic. Critically, the sequence of observations cannot be re-arranged, as each observation is typically dependent on one or more previous observation. This dependence is termed autocorrelation and accounting for it is one of the primary functions of a time series model.

Autoregressive Models

A basic autoregressive (AR) model is formed as a linear combination of previous values, plus a white noise term that accounts for random variations (the stochastic portion). An $AR(p)$ model for predicting a value X at time t can be written as

$$X_t = c + \sum_{i=1}^{p} \varphi_t X_{t-1} + \varepsilon_t$$

where $\varphi_1, \varphi_2, \ldots, \varphi_p$ are the p parameters, c is a constant, and ε_t is the white noise term.

Multivariate Models

When the AR model is extended to the multivariate case (i.e. allowing for multiple time series), a Vector AR (VAR) model is formed. This model will support not

only a time series for defect count, but also time series for the two release plan variables: improvements and new features.

Endogeneity and Exogeneity

Under the VAR model, the behavior of each time series is explained by both its own past values and the past values of the other time series. This makes the variables endogenous.

The alternative is that a time series should not be explained by itself, and is only used to explain other time series. This type of explanatory variable is called exogenous, and could be considered an input.

By also considering exogenous variables, a VAR model would become a VARX model. This model meets the requirements of the explanatory model described in the Motivation section, since it would allow release plan variables to be kept exogenous and used only to explain defect count.

Trends and Stationarity

AR, VAR, and VARX models do not account for non-stationary data. If a time series is not stationary, differencing may produce a stationary series. Trending time series are challenging to analyze, because the summary statistics of mean, variance, and autocovariance vary over time, and are therefore not interpretable [6]. Two trend types are discussed here: deterministic and stochastic.

A deterministic trend will move upward or downward, meaning that the time series mean is non-constant. However, the time series will be constant according to a deterministic function and the time series movements will generally follow the

deterministic function, with non-permanent fluctuations above or below. Such a time series is said to be stationary around a deterministic trend.

In contrast, a stochastic trend shows permanent effects whenever random variations occur, and the series will not necessarily fluctuate only close to the area of a deterministic function. The application of differencing can be used to remove a stochastic trend.

Stationarity can be strict or weak (of some order). Strict stationarity occurs when statistical properties are invariant with respect to shifts of the time origin [13]. Alternatively, a weak stationarity (of second order) can be established, and from this strict stationarity can be established by then assuming normality [4].

For a multivariate time series, stationarity holds if all the component univariate time series are stationary [18], so the goal of stationarity testing will be to establish second-order stationarity for each univariate time series component, and then show that the assumption of normality is reasonable. This will establish the stationarity of the multivariate time series as a whole. Next, tests are discussed for assessing if a deterministic or stochastic trend is present.

Unit Root and Stationarity Testing

A time series that contains a stochastic trend is non-stationary. A pure auto-regressive (AR) model of such a time series contains a unit root [6]. Testing for the presence of a unit root can therefore be used to test for non-stationarity. A unit-root test poses as the null hypothesis that an AR model has a unit root. Then, a test statistic is measured. If the p-value is below some significance, the null hypothesis can be rejected,

and it is established that the time series does not have a stochastic trend. The Augmented Dickey Fuller (ADF) test is often used for unit root testing.

On the other hand, a stationarity test uses the null hypothesis that a time series is stationary around a deterministic trend. If the test statistic shows that this hypothesis can be rejected at some significance level then a stochastic trend should be considered by the unit root test. The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test can be applied for testing stationarity.

CHAPTER V

METHODS

In this chapter, we consider methods for both obtaining time series data (data methods) and for obtaining a model using that data (modeling methods).

Data Methods

In this section, the data sources and the rationale for their selection are discussed. Then the methods used for preparing data for modeling, by cleansing, sampling, stationarity testing, and windowing, are described. The procedure used is summarized in Figure 3.

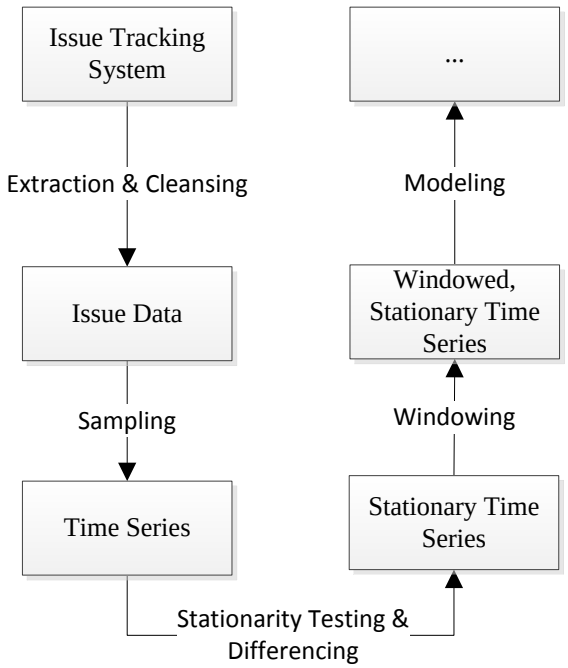

Figure 3 An overview of data methods.

Data Sources

The empirical datasets used to establish predictive models came from several software projects' historical data, and were taken from their issue tracking systems[1]. To be considered for selection, it was required that a project:

- Has been actively developed for at least several years
- Has openly available issue tracking system data
- Distinguishes between defects and other issue types

The projects selected by these criteria were:

- MongoDB[2]: *core server* product
- Hibernate[3]: *orm* product
- NetBeans[4]: *platform* and *java* products

The MongoDB software project has been actively developed since 2009. MongoDB uses JIRA[5] for issue tracking. Issue data for *core server* product was exported from the project's JIRA web interface[6] as XML data.

The Hibernate software project has been actively developed since 2003, and also uses JIRA for issue tracking. Issue data for the *orm* product was exported from the project's JIRA web interface[7] as XML.

---

[1] An issue tracking system can be used to track bugs, new features, improvements, etc.
[2] MongoDB is a scalable document-oriented database system (http://www.mongodb.org/).
[3] Hibernate is an object-relational mapping (ORM) framework for the Java language.
[4] NetBeans is a software development platform written in Java
[5] JIRA is an issue tracking and project management system made by Atlassian
[6] The project's JIRA web interface is at https://jira.mongodb.org/browse/SERVER
[7] The project's JIRA web interface is at https://hibernate.atlassian.net/projects/HHH

The Netbeans software project has been actively developed as an open source project since 2000. The project uses Bugzilla for issue tracking. Issue data for the *platform* and *java* products was obtained using a 2010 dump of the Bugzilla MySQL database. This database was made available as part of the mining challenge for the 2011 conference for Mining Software Repositories[8].

Data Preparation

The datasets need some preparation before a time series modeling procedure is run. Preparatory steps include: cleansing, sampling, stationarity testing and differencing, and windowing. These steps are now explained below.

Data Cleansing

Not all of the data was preserved for modeling. The modification or removal of data is discussed next.

First, only issues with resolutions such as *fixed, complete,* or *done* were kept. Issues with other resolutions, such as *unresolved, won't fix, duplicate,* etc. were counted as unfixed and were not kept. This was done because the proposed model structure assumes that bug creation is explained by software changes. Therefore, issues that do not result in any change were not included in the dataset.

Next, issues that are categorized as sub-tasks were converted to be the same issue type as the parent issue. Those sub-tasks whose parent issue is not in the dataset are considered orphans and discarded.

[8] The mining challenge data is available at http://2011.msrconf.org/msr-challenge.html

Data Sampling

Data was sampled at regular periods to measure the following: number of improvements resolved, number of features resolved, and number of bugs created. As an example, this sampling process is illustrated in Figure 4, with the outcome of sampling the example data shown in Table 1.

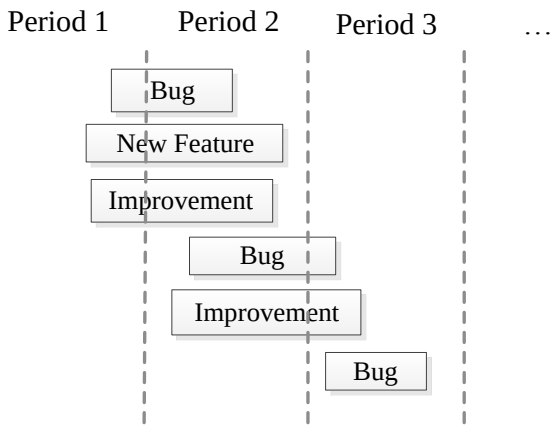

Figure 4 Sampling issue data by dividing time into equally-spaced periods.

Table 1 Results of sampling example issues shown in Figure 4.

Period	Improvements Resolved	New Features Resolved	Bugs Created
1	0	0	1
2	1	1	1
3	1	0	1

Stationarity Testing & Differencing

To establish stationarity, we first need to see if we can rule out the presence of a stochastic trend by applying the augmented Dickey-Fuller (ADF) test. If we can indeed rule out a stochastic trend, we should be able to confirm stationarity by applying the KPSS test. Or, if a stochastic trend cannot be ruled out, then KPSS test should be applied to check that trend stationarity is also rejected. If the data is found to have a stochastic trend, it should be differenced and then retested to confirm (trend) stationarity. In both tests, it will be assumed that the deterministic component is constant, with an intercept but no trend. The `ur.df` and `ur.kpss` functions from the *urca*[9] library were used to perform the ADF and KPSS tests, respectively.

Time Windowing

It is assumed that the software development process underlying a given project might change over time. Rather than developing a model that also changes over time, the data will be kept for modeling only if it occurs within a time window. This will limit the amount of process change the model is exposed to. Taking this approach means that the modeling methods will be executed for each time-windowed part of the data. See an illustration of a window in Figure 5.

[9] The *urca* library (http://cran.r-project.org/web/packages/urca) provides tests for time series data, and is freely available as a package for the *R* computing environment.

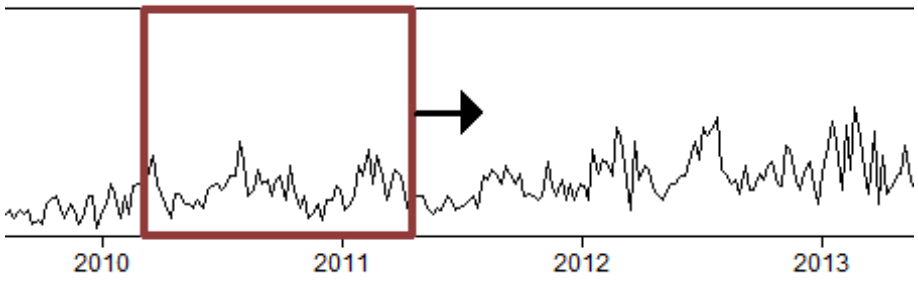

Figure 5 An illustration of time-windowing, where only data within the window is used for modeling.

It will be necessary to advance the time window after modeling the data within the window, so that the entire time series can take part in the modeling. This notion of applying modeling data within the window, advancing the window by one sample, and then repeating until the end of the time series is reached, is called herein a *sliding window*.

Modeling Methods

The typical method for building time series models involves specification, estimation, and diagnostics checking [4]. Once specified and estimated, the diagnostic checking step ensures that only valid models are considered for selection. The final step of modeling would be selection, where models are compared by some model selection criterion [4]. The next sections present the approach used to specify, estimate, check, and select a VARX model to be used for defect prediction.

Model Specification & Estimation

Specification of a $VARX(p)$ model is accomplished by choosing an order p, which is the number of autoregressive terms to include in the model. Once an order is

specified, the model parameters can be estimated by a procedure such as least squares regression.

The model order will directly affect the number of parameters included in the model. One goal of specification will be to avoid having too many parameters relative to the number of observations. The following derivation will lead to a simple rule for limiting the model order in this respect. First, let n be the number of time samples in a time series. When there are m time series, each sample contains m observations, so there are mn total observations for all time series. Next, for a $VARX(p)$ model of the m time series variables, there are m^2p unknown parameters to be estimated. Let the ratio of observations to parameters be denoted by

$$K = \frac{mn}{m^2p} = \frac{n}{mp}$$

To keep K at or above some minimum ratio K_{min}, so there are not too few observations per parameter, we form the inequality

$$K_{min} \leq K = \frac{n}{mp}$$

In terms of p this becomes

$$p \leq \frac{n}{mK_{min}}$$

Then, for a fixed value of K_{min}, an upper bound on the model order would be

$$p_{max} = \left\lfloor \frac{n}{mK_{min}} \right\rfloor$$

With this upper bound, model specification will include the generation of models having order 1, 2,..., p_{max}. These models, with their estimated parameters, will be candidates for final model selection after undergoing diagnostic checking.

The `estVARXar` function of the *dse*[10] library was used to estimate the parameters of a VARX model.

Diagnostics Checking

Diagnostic checking is performed to verify that a model can be accepted. This step includes testing for model stability, inadequacy, and normality.

Stability Test

For an autoregressive model to be stable, the roots of the process characteristic equation must lie outside the unit circle [4]. Equivalently, the inverse of the roots must lie inside the unit circle. The `stability` function from the *dse* library was used to perform this stability test.

Portmanteau Test

For an adequate ARMA model, it can be shown that "As the series length increases, the [model residuals] become close to the white noise . . ." [4, p. 338] For this reason, there are model inadequacy tests formed around a study of the residuals.

One of these tests, the Ljung-Box test, forms a statistic from the autocorrelation of the residuals (up to some lag). In this test, the null hypothesis is that residuals are independent, so their autocorrelation is not high enough to be distinguished from a white

[10] The *dse* library (http://cran.r-project.org/web/packages/dse) provides tools for time series models, and is freely available as a package for the *R* computing environment.

noise series. To support this hypothesis, the test p-value should be above some level of significance. The `Box.test` function from the *stats*[11] library was used for performing the Ljung-Box inadequacy test, with a 5% significance level.

Normality Test

To form a prediction interval for the model forecast, it is assumed that model residuals are normal. Therefore, models with non-normal residuals violate this assumption. Normality of model residuals are tested using the Jarque-Bera (JB) adjusted Lagrange multiplier (ALM) test, which is very precise for a wide range of sample sizes [5]. The JB test in general is testing that sample skewness and kurtosis matches that of a normal distribution. The `jbTest` function from the *fBasics*[12] library was used to perform the JB ALM normality test, with a 5% significance level.

Model Selection

Model selection criteria are used to compare models according to their fit, by penalizing for residual error and the number of parameters. There are a number of different selection criteria, including Akaike Information Criterion (AIC), AIC with correction (AICc), and Bayesian Information Criterion (BIC). Bisgaard and Kulahci noted that ". . . [t]he penalty for introducing unnecessary parameters is more severe for BIC and AICC than for AIC." [3] A less severe penalty for the number of parameters would be preferred in this case, since we are already limiting the number of parameters in

[11] The *stats* library (http://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html) provides core statistics functions, and is freely available as a package for the *R* computing environment.
[12] The *fBasics* library (http://cran.r-project.org/web/packages/fBasics/index.html) was prepared for teaching computational finance, and is freely available as a package for the *R* computing environment.

the model specification step, and because additional parameters may in fact be necessary to account for time series autocorrelations with higher lags. Therefore, AIC was chosen as the selection criterion. The `bestTSestModel` function from the *dse* library was used to perform model selection with the AIC criterion.

CHAPTER VI

RESULTS

The data and modeling methods described in the Methods chapter were applied to the four datasets: MongoDB *core server*, Hibernate *orm*, NetBeans *platform* and NetBeans *java*. The results of applying the methods are described in the following sections.

Data Results

Data was collected from project issue tracking systems, as described in the Data Sources section. Table 2 shows the range of dates over which data was collected for each project product, and the number of issues that were collected as a result, both before and after data cleansing. See the Data Cleansing section for an explanation of why certain issues were excluded.

Table 2 Date ranges of data collected, and the number issues that resulted.

Project Product Name	Date Range	Initial Issue Count	Final Issue Count
MongoDB *core server*	Apr, 2009 – Jan, 2015	7,007	6,971
Hibernate *orm*	Apr, 2003 – Apr, 2015	14,262	8,278
NetBeans *platform*	Jan, 2001 – Jun, 2010	24,745	11,335
NetBeans *java*	Jan, 2001 – Jun, 2010	18,313	8,699

It is worth noting that none of the datasets contained many orphaned subtasks. The highest number found was 80 in the Hibernate *orm* dataset.

Sampling Results

The collected datasets were then sampled to create time series. Not knowing which sampling period would work best, sampling was performed for each of the following sampling periods: 7 days, 14 days, and 30 days. The resulting time series are shown in Appendix A: Time Series Data Plots.

Stationarity Testing & Differencing Results

The resulting time series were then tested for stationarity. The time series were found to be non-stationary, with the exception of the Hibernate *orm* dataset, which was stationary when using a 30-day sampling period. Differencing was found to remove non-stationarity, but not knowing how differencing would affect model accuracy, data differencing of degrees of 0, 1, and 2 were made available for the modeling phase. The stationarity testing results for non-differenced and differenced time series data can be found in Appendix B: Stationarity Testing Results.

Windowing Results

Not knowing which window size would work best for the sliding window, a range of window sizes were selected for each sampling period, as shown in Table 3.

Table 3 Sliding windows sizes to be used for each sampling period

Sampling Period	Sliding Window Sizes
7 days	36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78
14 days	24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54
30 days	12, 15, 18, 21, 24, 27, 30, 33, 36

Modeling Results

The modeling methods were first applied to the datasets using the sliding window approach. This was done in an exploratory fashion in which the whole procedure was repeated using various values for the parameters. The hope was to find the parameter values which could provide the best results. The results of this exercise are discussed first in the next section. Then, with the results of the exploratory modeling to guide in selecting parameter values, the sliding window approach is applied once to each dataset, and these final results are presented.

Exploratory Sliding Window Results

The parameters for the sliding window approach are: sampling period, window sizes, and degree of differencing. These parameters were varied for each data set. Several metrics are used to evaluate the results:

- The none-valid proportion, which is the proportion of windows with no valid model (all models fail either the stability or inadequacy test).
- The non-normal proportion, which is the proportion of windows, having a valid model, where model residuals are non-normal (fail the normality test).
- The root-mean-square error (RMSE) of the forecast errors from all windows used for prediction. Each error value comes from a forecast made in one window. The RMSE of these errors is computed by

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

where Y and $\hat{Y}$ are n size vectors for the actual and predicted values, respectively. The RMSE value is the standard deviation of the error distribution.

- The in-interval proportion, which is the proportion of windows with forecasted values within the given prediction interval.

The first two metrics, the none-valid and non-normal proportions, measure the frequency of cases where the forecasting step is not reached. These metrics will be grouped together and called the *validity* metrics. The next two metrics, RMSE and the in-interval proportion, measure the model accuracy. These metrics form a basis for choosing sliding windows parameter values, and will be called together the *accuracy* metrics.

The results from running the sliding window with a range of parameters are listed in Appendix C: Exploratory Modeling Results. In these results, the data is separated first by dataset, then by sampling period, and finally by the degree of differencing. From there, the window size is varied and metrics are recorded for each.

The significance of these results is now discussed, first from the standpoint of validity and then accuracy. Following this, a procedure is outlined for the selection of sliding window parameter values.

Effects on Validity

The validity metrics indicate that there are trends as the window size increases. See the plot in Figure 6 below, for example. However, these trends are not consistent for different sampling periods and across datasets, so no attempt will be made to generalize them. But for a given dataset and sampling period they should provide empirical

justification for choosing one window size over another, to minimize the number of invalid cases encountered over the course of the sliding window.

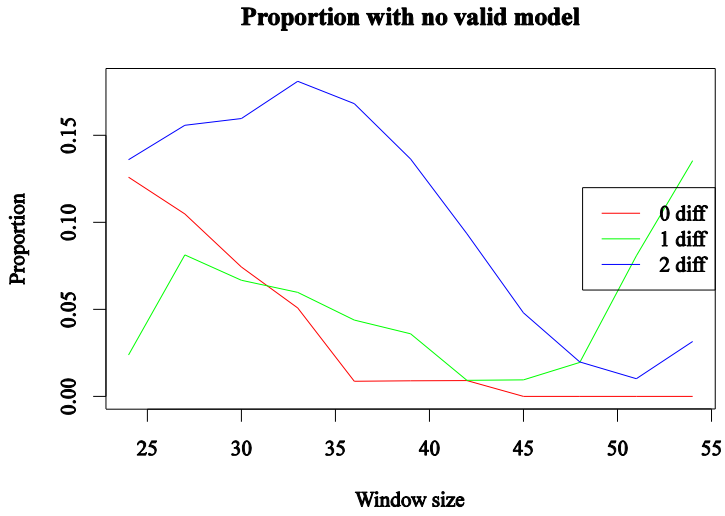

Figure 6 **Plot of the none-valid proportion, using the MongoDB *core server* dataset, with a 14-day sampling period.**

Effects on Accuracy

The accuracy metrics indicate that a higher degree of differencing results in lower model accuracy. See the plot in Figure 7 below, for example. The undifferenced data, unfortunately cannot be used because it is non-stationarity. It is not clear whether the window size has a consistent effect on accuracy that can be generalized, but again it may provide an empirical justification for choosing a window size to maximize accuracy, once a sampling period and degree of differencing are chosen.

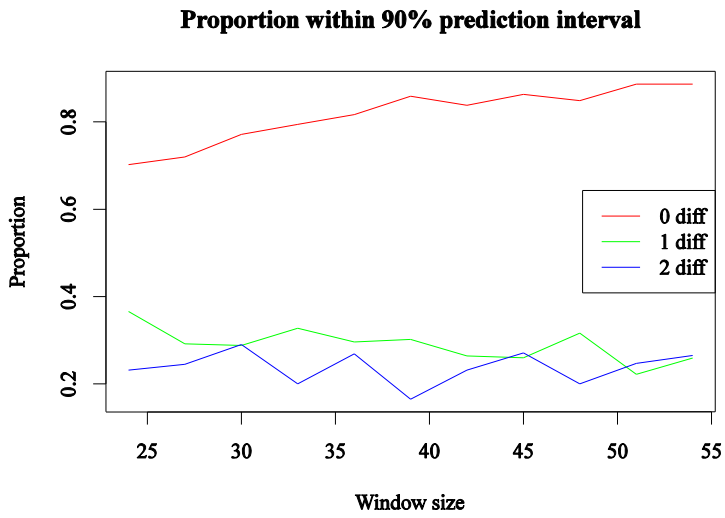

Figure 7 **Plot of the proportion of forecasts within a 90% prediction interval, using the MongoDB *core server* dataset, with a 14-day sampling period.**

The accuracy metrics also indicate that a smaller sampling period has a different effect on accuracy, depending on the degree of differencing. For an undifferenced time series, smaller sampling periods results in better accuracy. For time series that have one or two degrees of differencing, the effect of sampling period is inconsistent, and so should be checked empirically to obtain the best accuracy according to the choice in sample period.

Parameter Value Selection

Based on the observations made in the previous two sections, a procedure can be outlined to establish sliding window parameter values. First, the smallest degree of differencing is used, as stationarity allows. Next, if data is undifferenced then chose a 7-day (small) sampling period. Otherwise, try several sampling periods to see which results

in accuracy trend lines that are highest. Last, try several window sizes in order to maximize validity and accuracy.

This procedure is applied using the validity and accuracy results from Appendix C: Exploratory Modeling Results. First, since all of the time series require differencing, the degree of differencing chosen is 1 for all. Next, the sampling period and windows size are chosen to to try and maximize both validity and accuracy. The values chosen for these and the other parameters are shown in Table 4.

Table 4 The parameter values selected, based on results from exploratory modeling.

Dataset	Degree of Differencing	Period	Window
MongoDB *core server*	1	14	24
Hibernate *orm*	1	30	24
NetBeans *platform*	1	14	27
NetBeans *java*	1	14	30

Final Sliding Window Results

The sliding window approach was applied for each dataset using the parameters arrived at during exploratory modeling (see Table 4). The results from this final modeling step will be presented and discussed next. For each dataset, several aspects of the results will be discussed:

- The none-valid and non-normal proportions
- The distribution of actual compared to the distribution of predicted number of bugs

- The distribution of forecast errors, where each error is the difference between the predicted and actual number of bugs for one window.
- The in-interval proportion for a 75% or a 90% prediction interval

The comparison of actual and predicted number of bugs will be in the form of kernel density plots of the two distributions, shown together. The distribution of forecast mean errors will be presented in terms of shape, using a Q-Q plot, and also by scale, using the RMSE.

MongoDB core server Results

The MongoDB *core server* dataset was processed using a difference degree of 1, a sampling period of 14 days, and a window size of 24. Of the 126 windows used in the sliding window, no valid model could be found for 3 (2.38%) of them. And of the remaining 123 windows with valid models, none of the model residuals were non-normal, leaving 123 windows for making predictions.

The distributions of actual bugs and predicted bugs are quite similar in appearance, shown together in Figure 8. The distribution of errors between predicted and actual bug counts is shown in Figure 9. The scale of this distribution can be summarized by the RMSE value of 14.723.

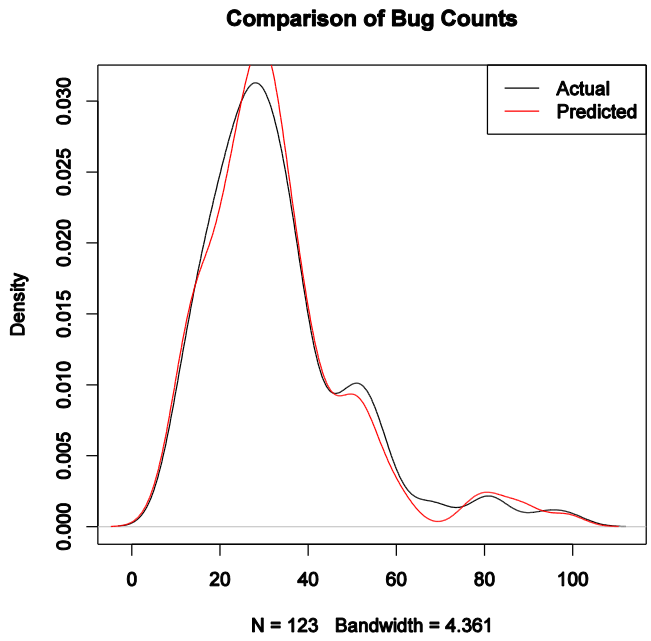

Figure 8 Comparison of the distributions for actual and predicted number of bugs.

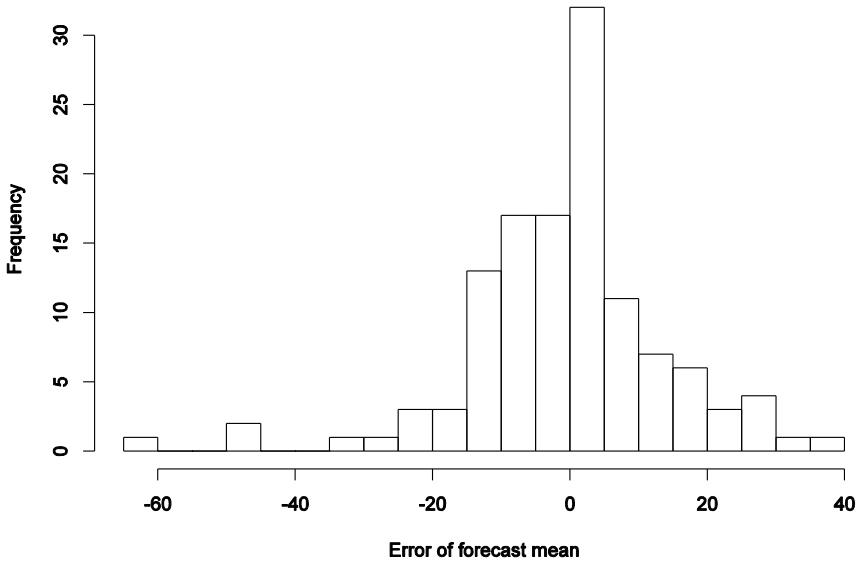

Figure 9 Histogram of forecast mean errors obtained using a 24-sample sliding window.

The shape of this distribution is visualized using the Q-Q plot in Figure 10. This plot shows that both the left- and right-tail portions of the distribution are non-normal. Of the 123 prediction windows, 45 (36.59%) were within a 90% prediction interval, and 34 (27.64%) were within a 75% prediction interval.

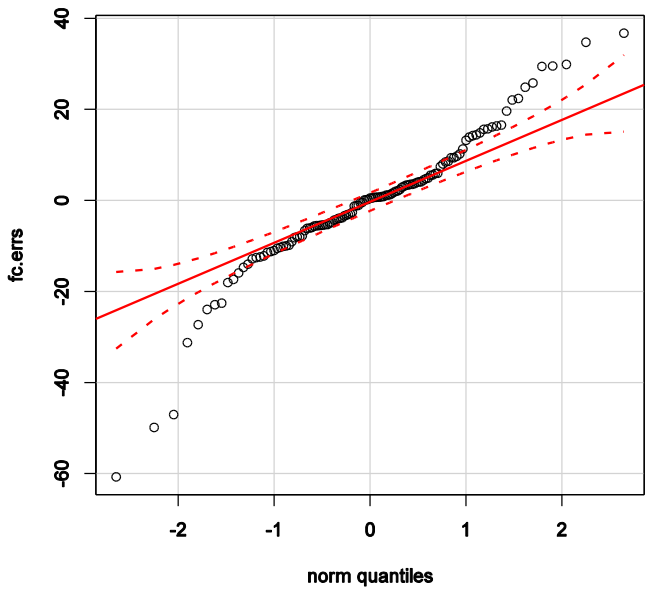

Figure 10 Q-Q plot of forecast mean errors.

Hibernate orm Results

The Hibernate *orm* dataset was processed using a difference degree of 1, a sampling period of 30 days, and a window size of 24. Of the 121 windows used in the sliding window, no valid model could be found for 5 (4.13%) of them. And of the remaining 116 windows with valid models, the model residuals were non-normal for 1 (0.86%) of them. This left 115 windows that were used to make predictions.

The distributions of actual bugs and predicted bugs are quite similar in appearance, shown together in Figure 11.

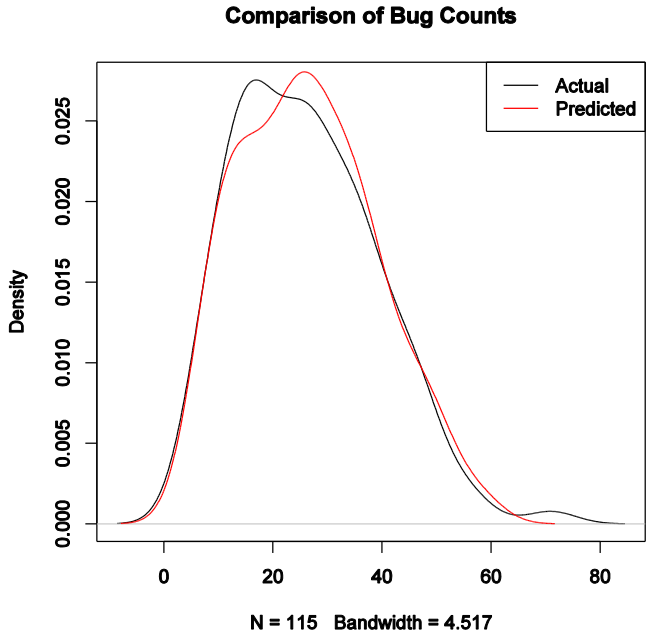

Figure 11 Comparison of the distributions for actual and predicted number of bugs.

The distribution of errors between predicted and actual bug counts is shown in Figure 12. The scale of this distribution can be summarized by the RMSE value of 10.27. The shape of this distribution is visualized using the Q-Q plot in Figure 13. This plot shows some right- and left-tail portions are non-normal. Of the 115 prediction windows, 62 (53.91%) were within a 90% prediction interval, and 52 (45.22%) were within a 75% prediction interval.

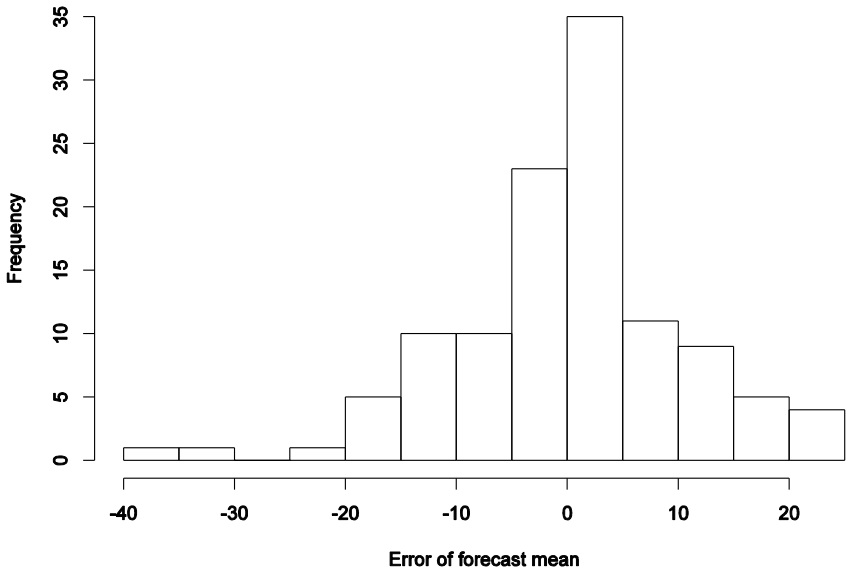

Figure 12 Histogram of forecast mean errors obtained using a 24-sample sliding window.

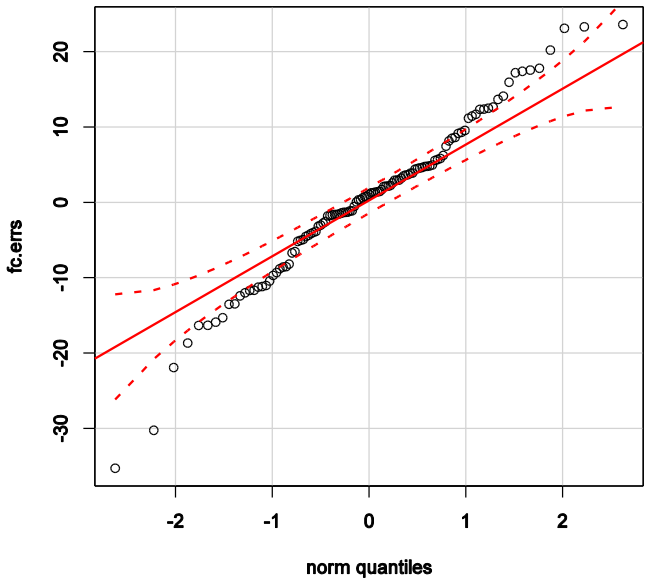

Figure 13 Q-Q plot of forecast mean errors.

NetBeans platform Results

The NetBeans *platform* dataset was processed using a difference degree of 1, a sampling period of 14 days, and a window size of 27. Of the 219 windows used in the sliding window, no valid model could be found for 21 (9.59%) of them. And of the remaining 198 windows with valid models, the model residuals were non-normal for 5 (2.53%) of them. This left 193 windows that were used to make predictions.

The distributions of actual bugs and predicted bugs are quite similar in appearance, shown together in Figure 14.

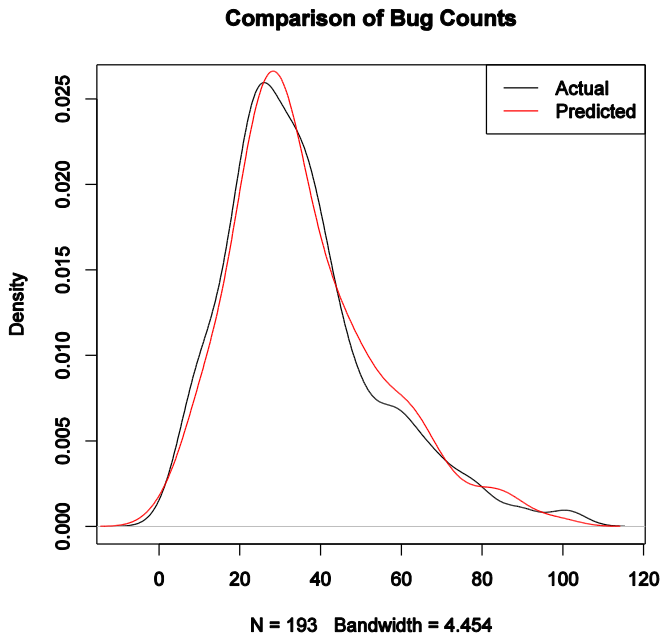

Figure 14 Comparison of the distributions for actual and predicted number of bugs.

The distribution of errors between predicted and actual bug counts is shown in Figure 15. The scale of this distribution can be summarized by the RMSE value of 15.2702. The shape of this distribution is visualized using the Q-Q plot in Figure 16. This plot shows that many of the tail values are outside of the confidence bands, especially on the left side.

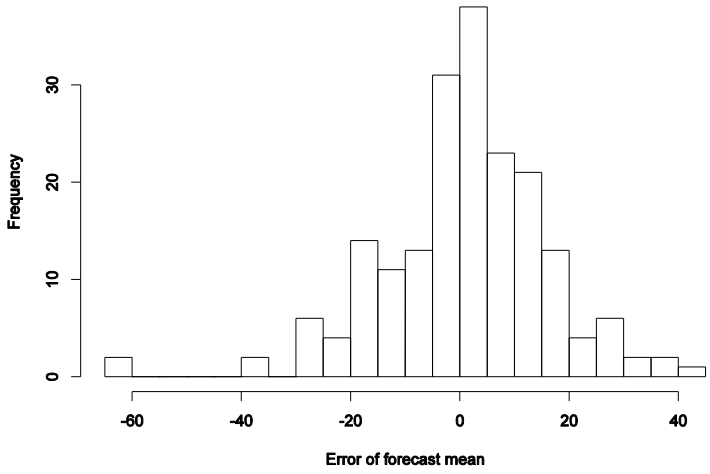

Figure 15 Histogram of forecast mean errors obtained using a 27-sample sliding window.

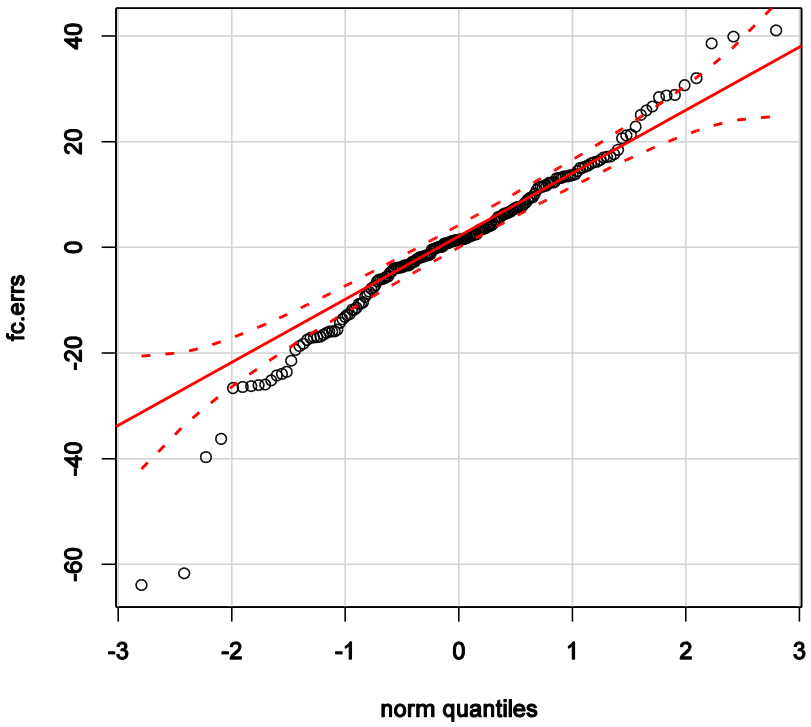

Figure 16 Q-Q plot of forecast mean errors.

Of the 193 prediction windows, 89 (46.11%) were within a 90% prediction interval, and 76 (39.38%) were within a 75% prediction interval.

NetBeans java Results

The NetBeans *java* dataset was processed using a difference degree of 1, a sampling period of 14 days, and a window size of 30. Of the 216 windows used in the sliding window, no valid model could be found for 28 (12.96%) of them. And of the remaining 188 windows with valid models, the model residuals were non-normal for 28 (14.89%) of them. This left 160 windows that were used to make predictions.

The distributions of actual bugs and predicted bugs are quite similar in appearance, shown together in Figure 17.

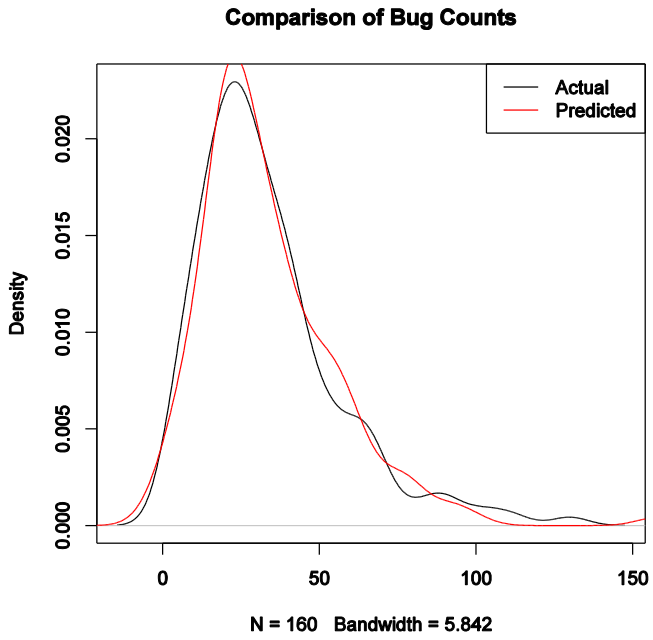

Figure 17 Comparison of the distributions for actual and predicted number of bugs.

The distribution of errors between predicted and actual bug counts is shown in Figure 18. The scale of this distribution can be summarized by the RMSE value of 18.0469. The shape of this distribution is visualized using the Q-Q plot in Figure 19. This plot shows strong non-normality at the tails, with almost all of the tail values outside of the confidence bands.

Of the 160 prediction windows, 69 (43.125%) were within a 90% prediction interval, and 49 (30.625%) were within a 75% prediction interval.

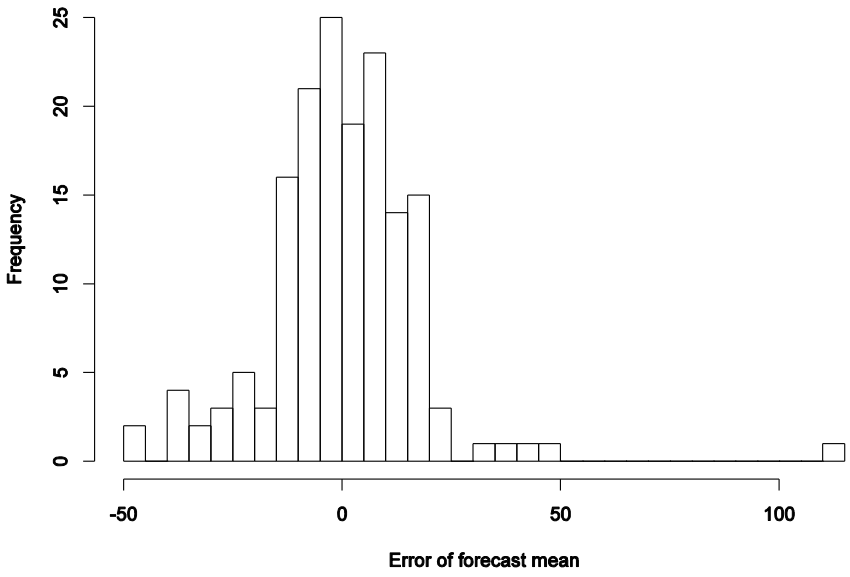

Figure 18 Histogram of forecast mean errors obtained using a 15-sample sliding window.

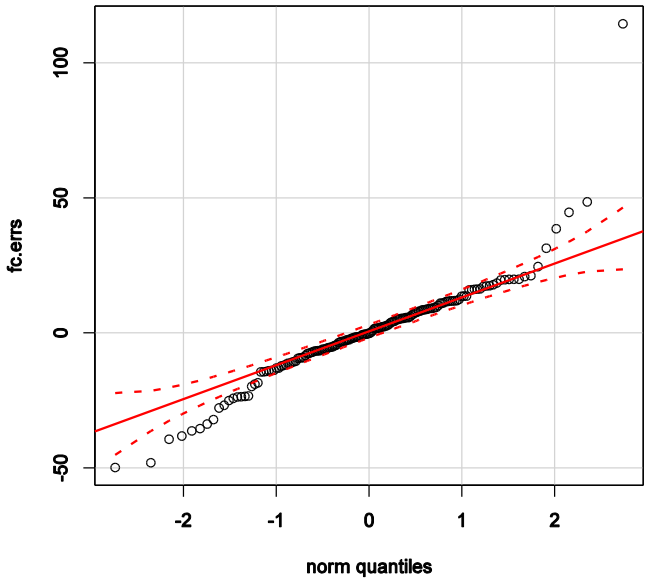

Figure 19 Q-Q plot of forecast mean errors.

A summary of all the final results is presented in Table 5, which is convenient for making a comparison.

Table 5 A comparison of the final modeling results across datasets.

Dataset	Window Count	None-valid Proportion	Non-normal Proporation	RMSE	In-interval Proportion	
					90% Conf.	75% Conf.
MongoDB *core server*	126	2.38%	0%	14.7230	36.59%	27.64%
Hibernate *orm*	121	4.13%	0.86%	10.2685	53.91%	45.22%
NetBeans *platform*	219	9.59%	2.53%	15.2702	46.11%	39.38%
NetBeans *java*	216	12.96%	14.89%	18.0469	43.13%	30.63%

CHAPTER VII

DISCUSSION

The validity of modeling results was evaluated by the none-valid and non-normal proportions. These measures both varied by window size, so windowing could be used to improve them. For the datasets and windowing parameters used, the none-valid proportions were between 2% and 13%, and the non-normal proportions were between 0% and 15%. Together, these proportions represent the risk that for any given sample window there will be no valid course for making a prediction.

The accuracy of model predictions was evaluated with RMSE and in-interval proportion. These measures both varied by window size, so windowing could be used to improve them. For the datasets and windowing parameters used, the in-interval proportions at a 90% prediction interval were between 36% and 54%, and the in-interval proportions at a 75% prediction interval were between 27% and 46%.

Evaluating a dataset with a sliding window does not only provide control over validity and accuracy, it also conveys a picture of how a model can generally be expected to perform for any given window in the future. In the cases where the none-valid and non-normal proportions were quite low, this would lead to an expectation that for any given window in the future, there will likely be a valid model available, having normal residuals. Since the in-interval proportions were often far below the level of their prediction intervals, this would lead to an expectation that in many cases a model prediction would not be within the prediction interval. Such an expectation might

discourage the model's use for defect prediction. On the other hand, if a low RMSE value is obtained, the model may still be considered useful for defect prediction.

CHAPTER VIII

THREATS TO VALIDITY

The historical data used to form models and make predictions is taken from actual software projects' issue tracking systems. Though the data exists independently from the work of this thesis, there are still potential threats to the validity of the thesis results which may be due to: the way data was originally recorded in the issue tracking system, the way data was treated before use in modeling, as well as the relationships that are assumed to exist between the data variables. In the following sections, potential threats to internal validity and external validity are identified and discussed.

Internal Validity

Threats to internal validity serve to undermine the causal relationships that are assumed. Throughout this paper, *bugs created* have been held as a dependent variable, with *improvements resolved* and *features resolved* being held as independent variables. It is also assumed that there exists some causality between the independent variables and the dependent variable. Several threats to this assumption are discussed next.

Ambiguous Temporal Precedence

The threat of ambiguous temporal precedence exists when it is not clear that one variable only occurs before another. Using the chosen model structure, the resolution of features and improvement s should occur before bugs are reported. But through visual inspection of the available time series data, it was such a temporal precedence was not clear. This confirms that internal validity is threatened by ambiguous temporal precedence.

Confounding

Confounding may arise due to the existence of an additional variable which affects the dependent variable, and whose behavior is related to that of an independent variable. The software development process is hugely complicated, both in the number of actors and in the ways that an actor can participate in the process. Because the thesis work relies only on data from an issue tracking system, there are likely other variables which may play into the creation of software defects. The existence of unmeasured or unconsidered variables makes confounding a definite, but also probably inevitable, threat to internal validity.

History

The effects of external events, outside of the scope of software development, may contribute to the behavior of the dependent variable. For example, team attrition, team reorganization, and negative quality reports may all affect current and future development activities. With such large changes, development teams may be forced to change focus in the areas of quality or functionality. Such changes may disrupt historical behavioral patterns and relationships between variables. This threat to validity is perhaps unavoidable in long-term consideration of historical data. The approach taken to counter this threat to validity is to window data such that models are less exposed to structural changes.

External Validity

The type of generalizability that is sought for in this thesis work is two-fold: generalizability across software projects and generalizability across time windows. Threats to these types are discussed in the next sections.

Generalizability across Software Projects

For results to be generalizable across software projects, they must be inferred from many datasets. So far only four datasets have been used. This small number of datasets limits how well the results can be generalized to other software projects. Also, by design the selected projects were all open source. This makes the datasets and project information available to all researchers, but might also threaten the generalizability to projects that are not open source.

Generalizability across Time Windows

For a particular project dataset, results will vary for each of the time windows. To provide a result that can be generalized across time windows within the data set, a sliding window is applied over the entire dataset and several measurements are obtained. These measurements are proportions that indicate how probable it is that any given time window will produce a valid, accurate model. Additionally, the distribution of forecast errors across time windows is presented for each dataset, to characterize the probability of obtaining any given range of forecast error.

CHAPTER IX

FUTURE WORK

To improve on the methods presented so far, a modification to the current methods is mentioned: excluding time windows that contain outliers. Additionally, two lines of potential future research are proposed: modeling with undifferenced data using birth-death process models and making use of change management data in a time series model.

Exclusion of Outliers

With each dataset, a distribution of the forecast errors was shown as a histogram. There appears to be one or more outliers present in each of these histograms. The presence of an outlier may indicate that a time window contains data whose behavior significantly deviates from the rest of the time series. Such deviations could be caused by unaccounted-for externalities, as is suggested in the History subsection from the Threats to Validity chapter. Because such externalities would not be accounted for by the model, it would be desirable to prevent their influence from confounding any time series model under consideration. The presence of outliers in the forecast error distribution can be established by statistical testing. Once a window is identified as containing an outlier, it may be necessary to exclude all samples in that window from the sliding window process. Or, a detailed inspection of the time series may reveal which portion of the data should be excluded.

For the datasets with the worst results, NetBeans *java*, a large outlier was present in the sliding window forecast errors. These errors are shown by window in Figure 20, revealing the location of the outlier at window 43, which includes samples 43 through 72.

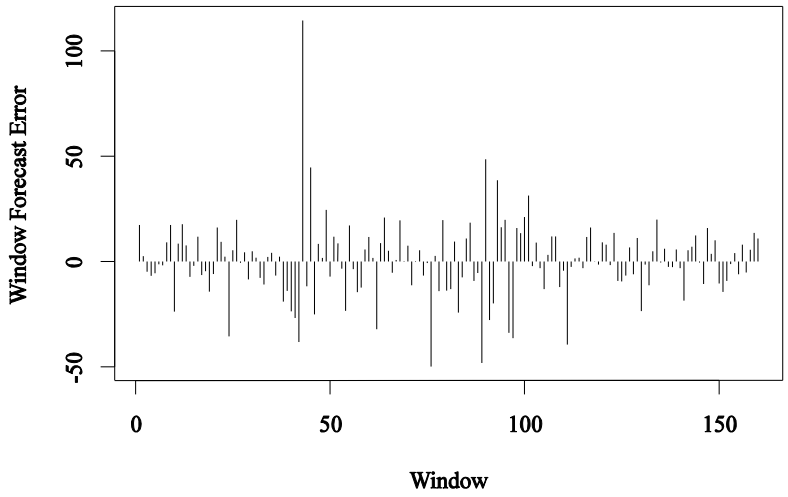

Figure 20 Forecast errors by window reveal the location of an outlier.

When the sliding window method is applied to samples after this window, starting at sample 73, the results improved over those obtained using the entire sample range, suggesting that outlier occurrence may provide good guidance for which portion of the sample range is usable for modeling. A comparison of the full and restricted results is shown in Table 6.

Table 6 Comparison of NetBeans java results from full sample range and restricted sample range (beginning after window where outlier occured).

Sample Range	Window Count	None-valid Proportion	Non-normal Proporation	RMSE	In-interval Proportion	
					90% Conf.	75% Conf.
Full	216	12.96%	14.89%	18.0469	43.13%	30.63%
Restricted	144	11.81%	7.87%	16.2761	49.57%	34.19%

Modeling with Birth-death Processes

The exploratory modeling results showed much better model accuracy when using the undifferenced time series data, with in-interval proportions near the level of the prediction interval. If a model could be used that operates on the undifferenced data without violating the model assumptions, then much better accuracy could be obtained. The model may need to take into account the special nature of the issue tracking system data. This data will always be non-negative, since it is count data. And due to the irregular flurries of software development activity, this means that the count data tends to spike and then return to a low, zero or near-zero value. The plot of undifferenced time series data in Figure 21 illustrates this tendency. Increasing the sampling period will smooth the sharp features somewhat, but not greatly, and at the loss of feature detail.

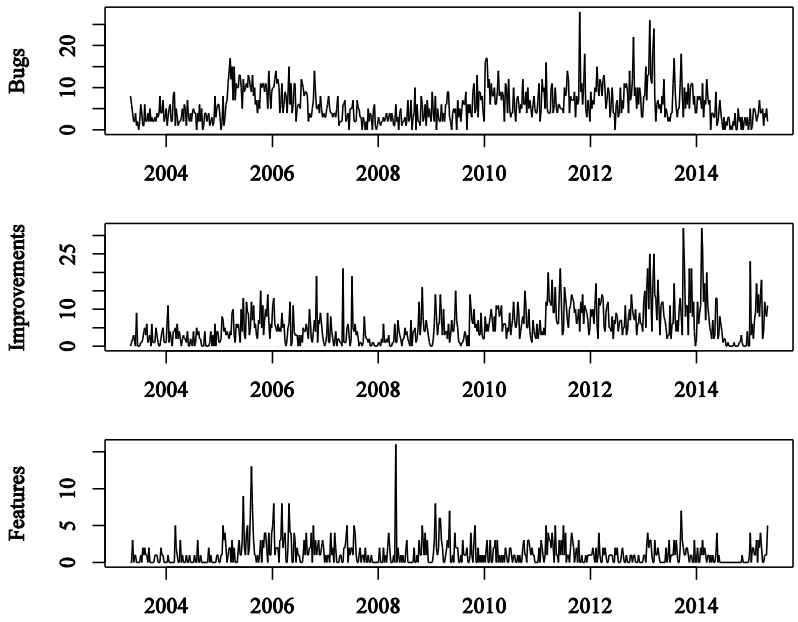

Figure 21 Undifferenced time series data from the Hibernate *orm* dataset, using a sampling period of 7 days.

Rather than smoothing or differencing the data to make it valid for a conventional time series model, another approach is to choose a model that is suitable for handling count data. It is proposed that a birth-death process be used as a model of this kind. In a birth-death process, the state transitions whenever a birth or death occurs, and count is incremented or decremented, respectively. The birth and death in this case would be the creation and resolution of a software issue.

Modeling with Change Management Data

A problem with the use of issue tracking system (ITS) data is that it is disconnected from the actual changes made to the software. This is a problem for two reasons. First, because there is a time lag between when a software change is committed and when the software change is reported in the ITS. Fortunately, if this lag time were characterized then a suitable sampling period can be chosen to minimize any negative effect. The other reason why a disconnect is problematic is that the issue tracking data does not contain direct information as to the magnitude of the software changes made, nor to which software subsystem the changes were made.

To overcome this lack of information, it is proposed that change management (CM) data be used as the exogenous input to a time series model, in place of the new feature and improvement data currently being used. CM data can provide information to both the time and magnitude of a change. Coupled with the existing bug report data from the ITS, such a model could capture the varying degree to which a software change might be likely to lead to bug reports, based on factors such as: the magnitude of the change, location in the codebase, and the author.

CHAPTER X

CONCLUSION

The data and modeling methods described allowed issue tracking system data to be used to form a time series model for defect prediction. These methods were applied to datasets from several open-source software projects.

The data methods that were employed helped to improve the modeling results. To begin with, non-stationarity was removed by differencing. This allowed the data to be used by the model, when non-stationary data could not be used. Then, validity and accuracy were improved by windowing. This was accomplished by choosing windows with: a low proportion of invalid models, a low RMSE, and a high proportion of forecasts values within a prediction interval. Without windowing, a model would need to account for an entire dataset, even where structural changes may occur.

The modeling methods were used to select model order and to estimate parameters. Additionally, the modeling methods allowed for diagnostic testing to identify invalid models or models with non-normal residuals. The proportion of windows with unusable models varies by window size, so being able to identify such unusable models and also to control the window size gives some control over this proportion.

REFERENCES

[1] F. Akiyama. An example of software system debugging. In IFIP Congress (1), volume 71, pages 353–359, 1971.

[2] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. Information and software technology, 43(14):883–890, 2001.

[3] S. Bisgaard and M. Kulahci. Time series analysis and forecasting by example. John Wiley & Sons, 2011.

[4] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. Time Series Analysis. John Wiley, 2008.

[5] W. Diethelm, and H. G. Helmut. Precise finite-sample quantiles of the Jarque-Bera adjusted Lagrange multiplier test. Swiss Federal Institute of Technology. Institute for Theoretical Physics, ETH Hönggerberg, C-8093 Zurich 26: 70-71, 2005.

[6] P. H. Franses. Time series models for business and economic forecasting. Cambridge university press, 1998.

[7] J. E. Gaffney. Estimating the number of faults in code. Software Engineering, IEEE Transactions on, SE-10(4):459–464, July 1984.

[8] E. Giger, M. Pinzger, and H. C. Gall. Comparing fine-grained source code changes and code churn for bug prediction. In Proceedings of the 8th Working Conference on Mining Software Repositories, pages 83–92. ACM, 2011.

[9] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. Software Engineering, IEEE Transactions on, 26(7):653–661, 2000.

[10] S. Henry and D. Kafura. The evaluation of software systems' structure using quantitative software metrics. Software: Practice and Experience, 14(6):561–573, 1984.

[11] H. Jiang, J. Zhang, J. Xuan, Z. Ren, and Y. Hu. A hybrid ACO algorithm for the next release problem. In Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on, pages 166–171. IEEE, 2010.

[12] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. SIGSOFT Softw. Eng. Notes, 29(6):263–272, Oct. 2004.

[13] T. K. Moon and W. C. Stirling. Mathematical methods and algorithms for signal processing, volume 1. Prentice Hall, New York, 2000.

[14] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pages 284–292. IEEE, 2005.

[15] W. Shadish, T. Cook, and D. Campbell. Experimental and Quasi-Experimental Designs for Generilized Causal Inference. Houghton Mifflin, Boston, 2002.

[16] L. L. Singh, A. M. Abbas, F. Ahmad, and S. Ramaswamy. Predicting software bugs using arima model. In Proceedings of the 48th Annual Southeast Regional Conference, page 27. ACM, 2010.

[17] J. Xuan, H. Jiang, Z. Ren, and Z. Luo. Solving the large scale next release problem with a backbone-based multilevel algorithm. Software Engineering, IEEE Transactions on, 38(5):1195–1212, 2012.

[18] K. Yang and C. Shahabi. On the stationarity of multivariate time series for correlation-based data analysis. In Data Mining, Fifth IEEE International Conference on, pages 4–pp. IEEE, 2005.

[19] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In Proceedings of the 9th annual conference on Genetic and evolutionary computation, pages 1129–1137. ACM, 2007.

APPENDIXES

Appendix A: Time Series Data Plots

Plots of the time series data obtained from sampling the software project datasets are illustrated in the figures below.

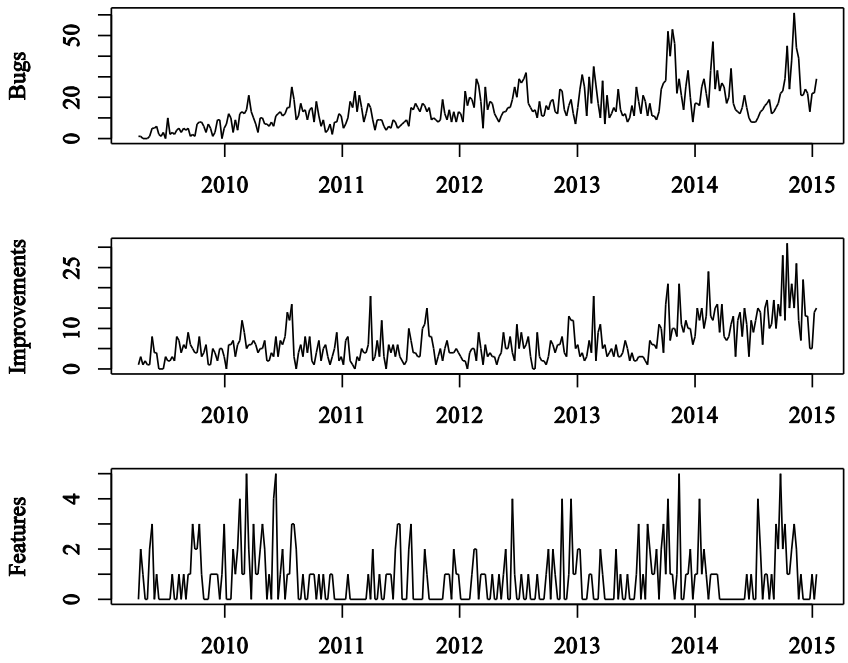

Figure 22 Time series resulting from sampling the MongoDB *core server* dataset with a 7-day sampling period.

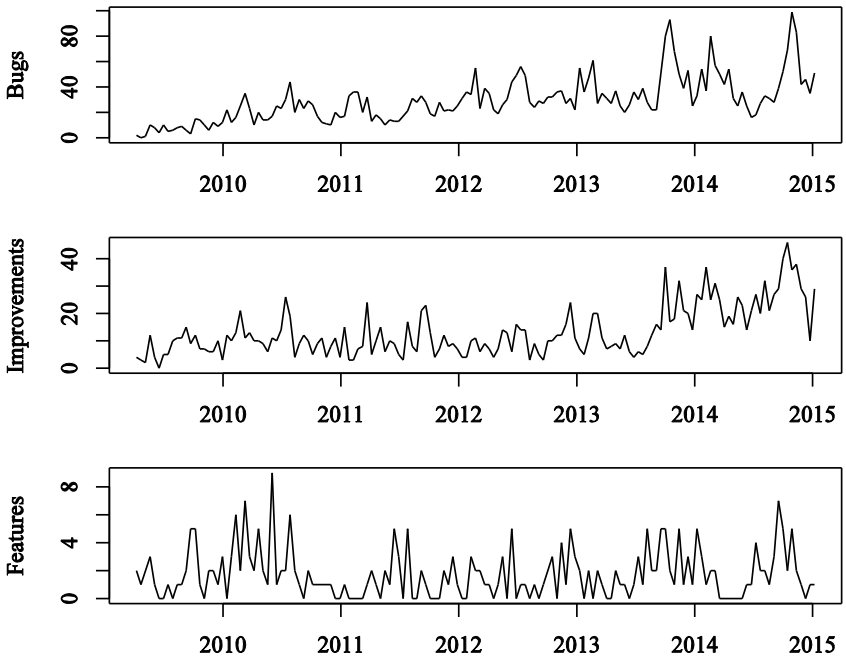

Figure 23 Time series resulting from sampling the MongoDB *core server* dataset with a 14-day sampling period.

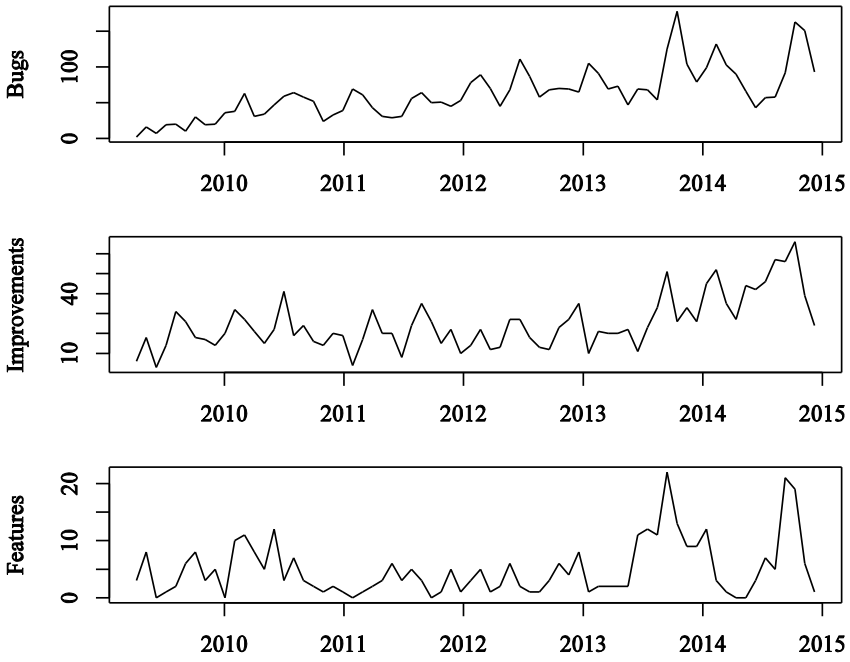

Figure 24 Time series resulting from sampling the MongoDB *core server* dataset with a 30-day sampling period.

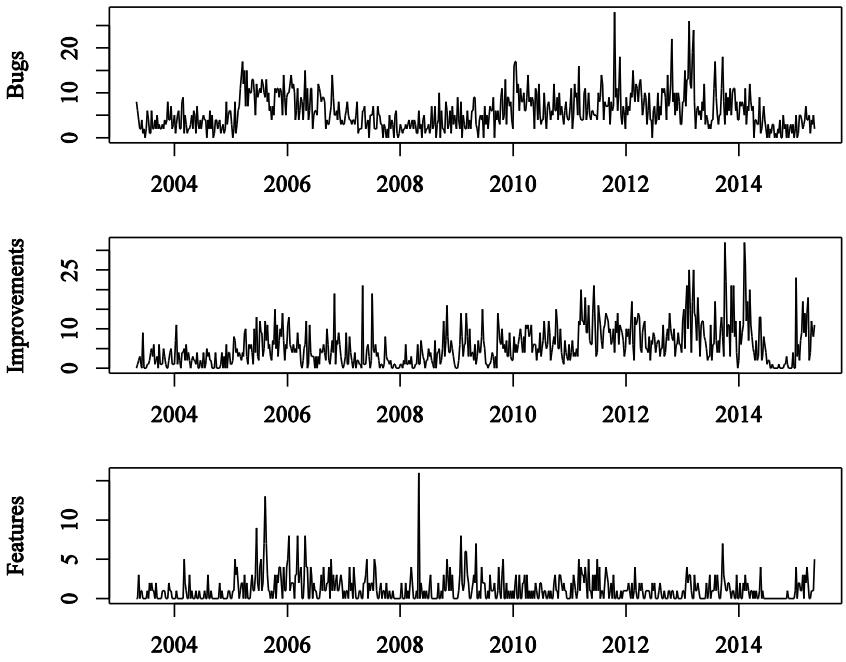

Figure 25 Time series resulting from sampling the Hibernate *orm* dataset with a 7-day sampling period.

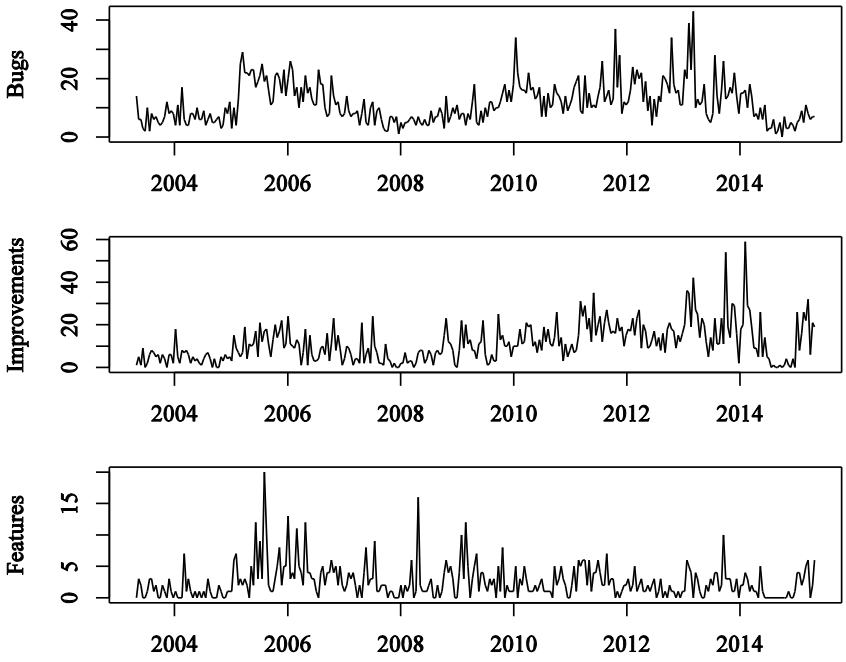

Figure 26 Time series resulting from sampling the Hibernate *orm* dataset with a 14-day sampling period.

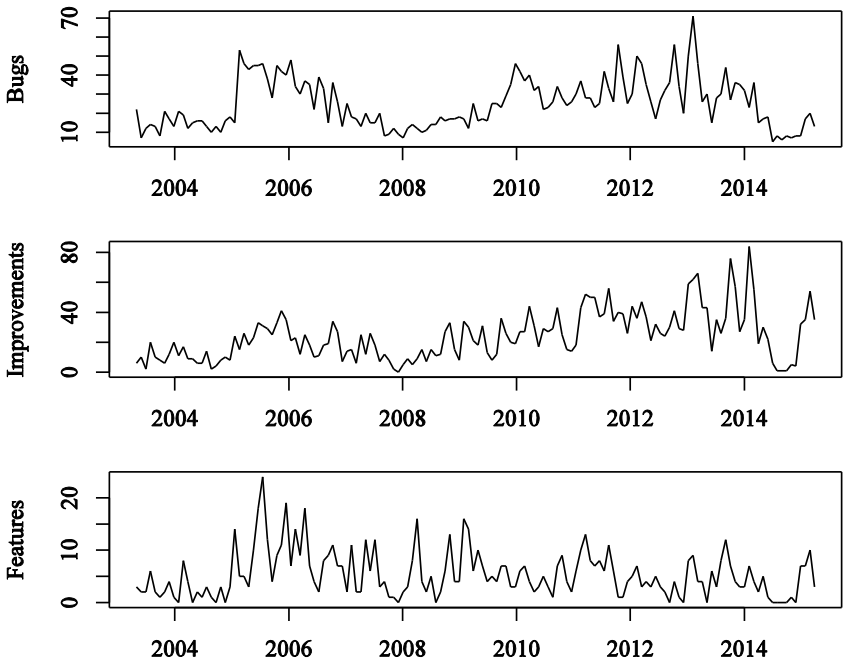

Figure 27 Time series resulting from sampling the Hibernate *orm* dataset with a 30-day sampling period.

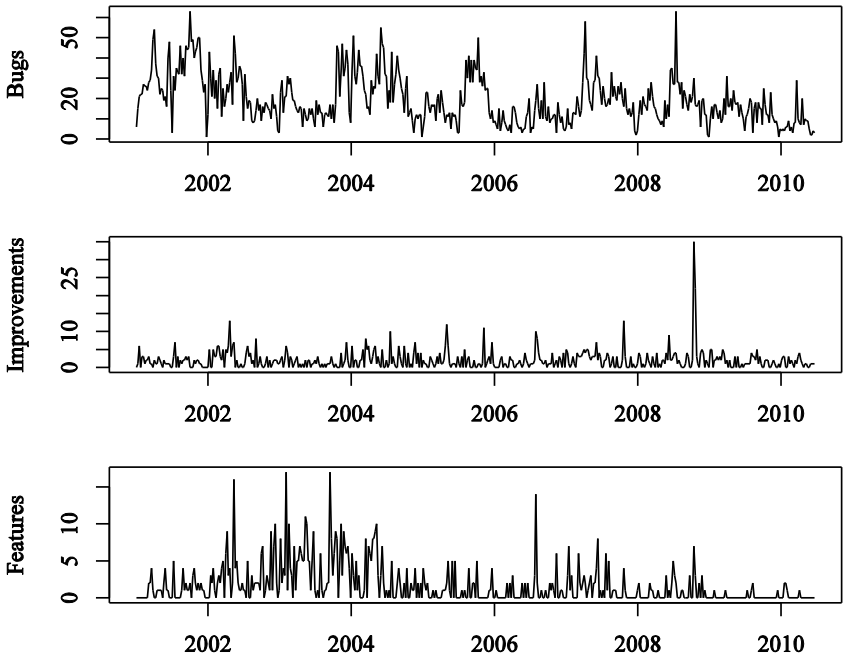

Figure 28 Time series resulting from sampling the NetBeans *platform* dataset with a 7-day sampling period.

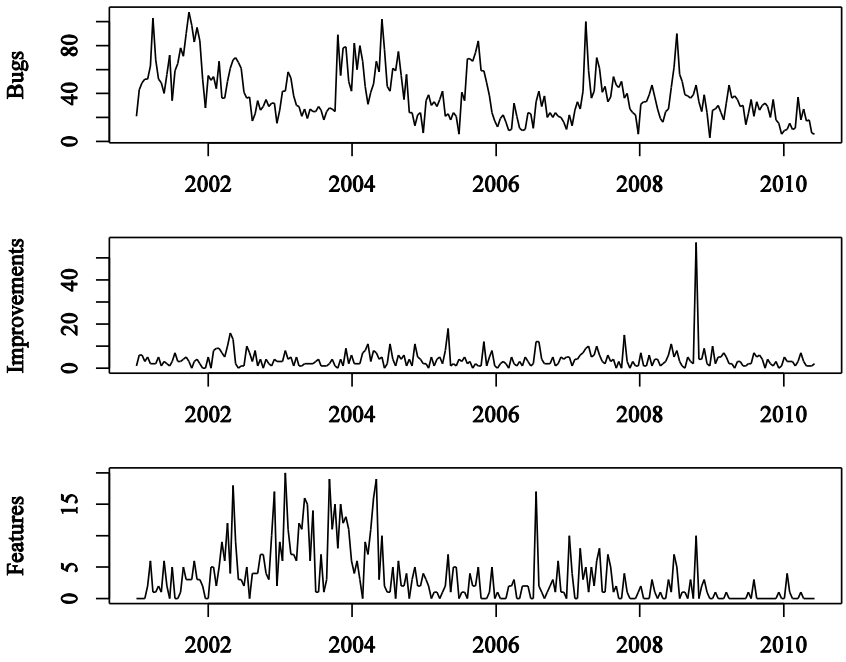

Figure 29 Time series resulting from sampling the NetBeans *platform* dataset with a 14-day sampling period.

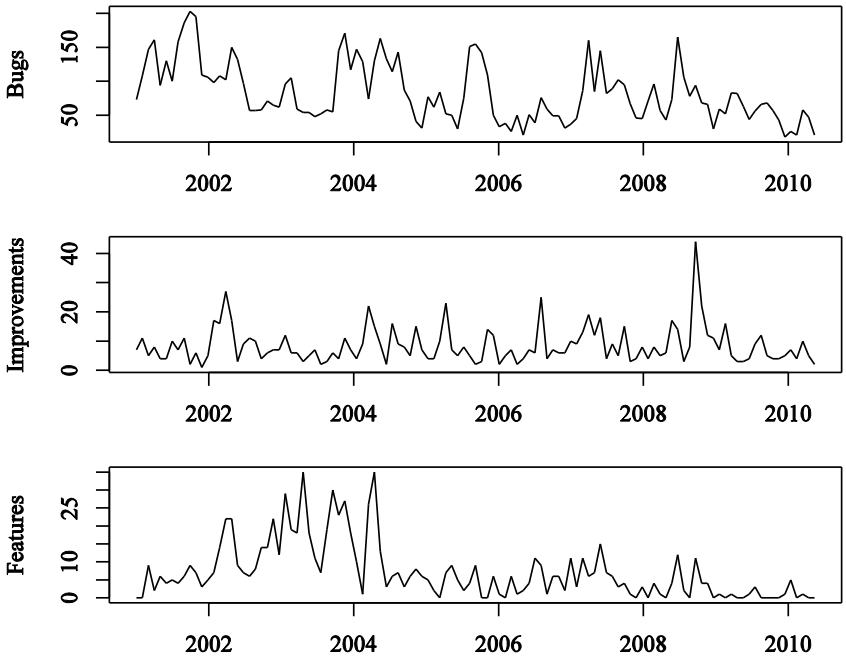

Figure 30 Time series resulting from sampling the NetBeans *platform* dataset with a 30-day sampling period.

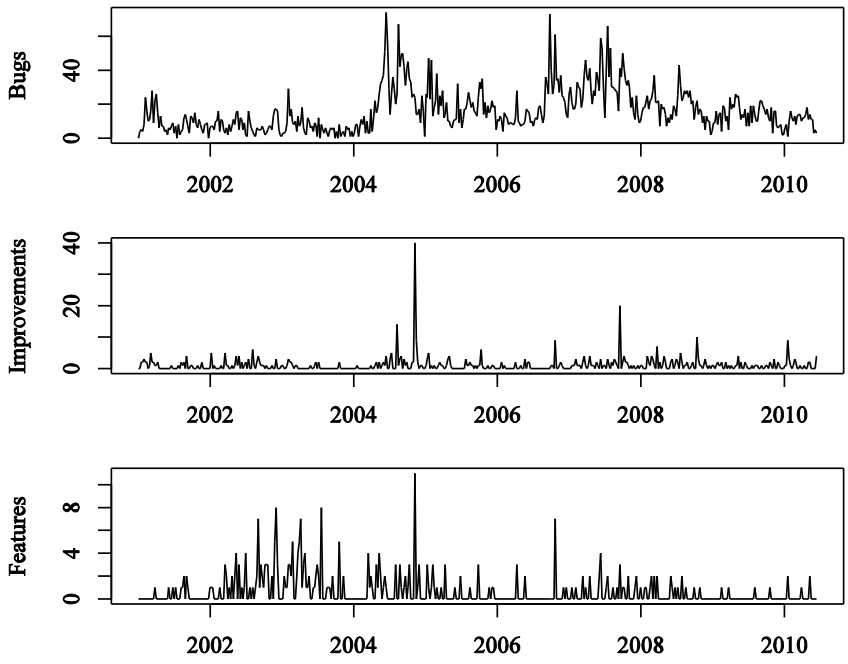

Figure 31 Time series resulting from sampling the NetBeans *java* dataset with a 7-day sampling period.

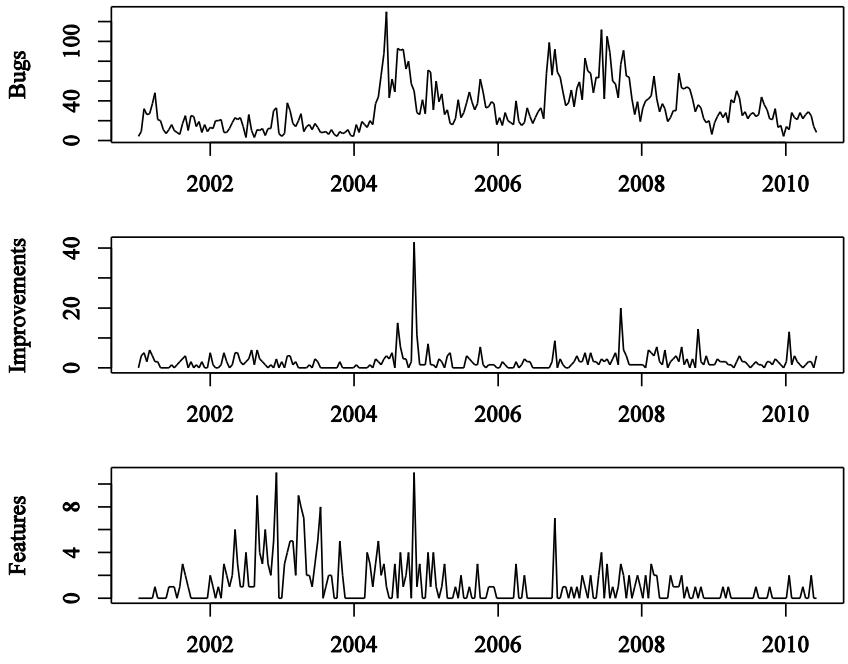

Figure 32 Time series resulting from sampling the NetBeans *java* dataset with a 14-day sampling period.

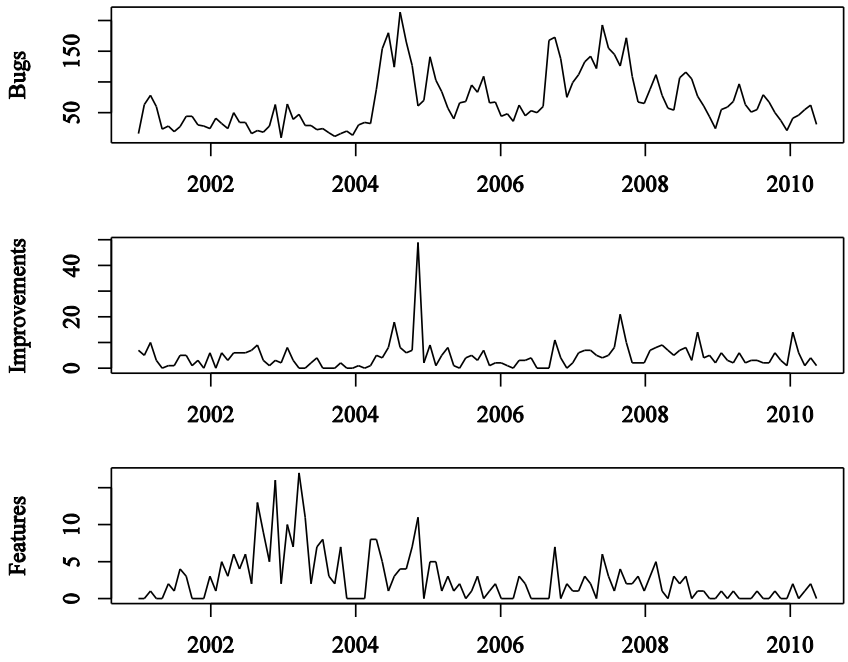

Figure 33 Time series resulting from sampling the NetBeans *java* dataset with a 30-day sampling period

Appendix B: Stationarity Testing Results

The results of stationarity testing are contained in the following tables, both for differenced and non-differenced data, and for each sampling period used (7-day, 14-day, and 30-day). The Augmented Dickey Fuller (ADF) and Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests were both run.

Table 7 Stationarity test results for the MongoDB *core server* time series data, with a sampling period of 7 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-5.009168 (< 1%)	12.59583 (< 1%)	2.810561 (< 1%)	-17.5075 (< 1%)	153.2578 (< 1%)	0.01125 (> 10%)
Improvements	-5.851094 (< 1%)	17.15104 (< 1%)	2.43306 (< 1%)	-20.2816 (< 1%)	205.6782 (< 1%)	0.01561 (> 10%)
New Features	-10.80503 (< 1%)	58.37575 (< 1%)	0.1376936 (> 10%)	-21.1322 (< 1%)	223.2843 (< 1%)	0.01278 (> 10%)

Table 8 Stationarity test results for the MongoDB *core server* time series data, with a sampling period of 14 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-3.954806 (< 1%)	7.903041 (< 1%)	1.977684 (< 1%)	-9.9046 (< 1%)	49.0530 (< 1%)	0.01552 (> 10%)
Improvements	-3.708167 (< 1%)	6.93959 (< 1%)	1.613534 (< 1%)	-12.8286 (< 1%)	82.2958 (< 1%)	0.02771 (> 10%)
New Features	-6.47668 (< 1%)	20.974 (< 1%)	0.10850 (> 10%)	-15.2122 (< 1%)	115.7057 (< 1%)	0.01891 (> 10%)

Table 9 Stationarity test results for the MongoDB *core server* time series data, with a sampling period of 30 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-3.858221 (< 1%)	7.500749 (< 1%)	1.413058 (< 1%)	-9.25577 (< 1%)	42.88765 (< 1%)	0.039097 (> 10%)
Improvements	-2.543267 (> 10%)	3.315932 (> 10%)	1.056792 (< 1%)	-7.90954 (< 1%)	31.31153 (< 1%)	0.037643 (> 10%)
New Features	-4.57232 (< 1%)	10.47271 (< 1%)	0.0928971 (> 10%)	-8.24411 (< 1%)	33.98363 (< 1%)	0.03578 (> 10%)

Table 10 Stationarity test results for the Hibernate *orm* time series data, with a sampling period of 7 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-9.891018 (< 1%)	48.91804 (< 1%)	0.5578149 (> 2.5%)	-28.932 (< 1%)	418.5303 (< 1%)	0.010707 (> 10%)
Improvements	-10.61357 (< 1%)	56.33118 (< 1%)	2.818589 (< 1%)	-28.7815 (< 1%)	414.1865 (< 1%)	0.007084 (> 10%)
New Features	-13.57442 (< 1%)	92.14123 (< 1%)	0.4729388 (> 2.5%)	-27.0919 (< 1%)	366.9867 (< 1%)	0.015379 (> 10%)

Table 11 Stationarity test results for the Hibernate *orm* time series data, with a sampling period of 14 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-5.601016 (< 1%)	15.68603 (< 1%)	0.3742806 (> 5%)	-17.2864 (< 1%)	149.4123 (< 1%)	0.020300 (> 10%)
Improvements	-6.266768 (< 1%)	19.65349 (< 1%)	1.913421 (< 1%)	-18.8948 (< 1%)	178.5101 (< 1%)	0.012008 (> 10%)
New Features	-9.058437 (< 1%)	41.03137 (< 1%)	0.3597925 (> 5%)	-19.4734 (< 1%)	189.6103 (< 1%)	0.016904 (> 10%)

Table 12 Stationarity test results for the Hibernate *orm* time series data, with a sampling period of 30 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-4.09404 (< 1%)	8.381815 (< 1%)	0.2431273 (> 10%)	-13.9911 (< 1%)	97.87568 (< 1%)	0.044111 (> 10%)
Improvements	-4.566302 (< 1%)	10.4551 (< 1%)	1.26875 (< 1%)	-12.6055 (< 1%)	79.4494 (< 1%)	0.020981 (> 10%)
New Features	-6.141746 (< 1%)	18.86246 (< 1%)	0.2832424 (> 10%)	-12.1244 (< 1%)	73.50509 (< 1%)	0.028846 (> 10%)

Table 13 Stationarity test results for the NetBeans *platform* time series data, with a sampling period of 7 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-6.8546 (< 1%)	23.4952 (< 1%)	1.9320 (< 1%)	-22.9636 (< 1%)	263.6646 (< 1%)	0.02620 (> 10%)
Improvements	-13.9027 (< 1%)	96.64276 (< 1%)	0.06701 (> 10 %)	-23.9283 (< 1%)	286.2845 (< 1%)	0.00844 (> 10%)
New Features	-10.0169 (< 1%)	50.1686 (< 1%)	2.4783 (< 1%)	-26.1357 (< 1%)	341.5365 (< 1%)	0.01208 (> 10%)

Table 14 Stationarity test results for the NetBeans *platform* time series data, with a sampling period of 14 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-4.78601 (< 1%)	11.4690 (< 1%)	1.1625 (< 1%)	-14.3822 (< 1%)	103.4296 (< 1%)	0.03728 (> 10%)
Improvements	-10.4056 (< 1%)	54.1394 (< 1%)	0.06183 (> 10%)	-19.4647 (< 1%)	189.4367 (< 1%)	0.01729 (> 10%)
New Features	-5.7482 (< 1%)	16.5211 (< 1%)	1.5325 (< 1%)	-17.1666 (< 1%)	147.3461 (< 1%)	0.02806b (> 10%)

Table 15 Stationarity test results for the NetBeans *platform* time series data, with a sampling period of 30 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-4.0439 (< 1%)	8.2138 (< 1%)	0.8163 (< 1%)	-8.7011 (< 1%)	37.8870 (< 1%)	0.04038 (> 10%)
Improvements	-6.8425 (< 1%)	23.4209 (< 1%)	0.05968 (> 10%)	-11.7327 (< 1%)	68.8281 (< 1%)	0.03475 (> 10%)
New Features	-4.1963 (< 1%)	8.8044 (< 1%)	1.0125 (< 1%)	-11.5676 (< 1%)	66.9154 (< 1%)	0.08033 (> 10%)

Table 16 Stationarity test results for the NetBeans *java* time series data, with a sampling period of 7 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-6.2924 (< 1%)	19.7971 (< 1%)	1.4979 (< 1%)	-22.5341 (< 1%)	253.8932 (< 1%)	0.02850 (> 10%)
Improvements	-14.2133 (< 1%)	101.0122 (< 1%)	0.1397 (> 10%)	-25.8415 (< 1%)	333.8919 (< 1%)	0.00801 (> 10%)
New Features	-12.5811 (< 1%)	79.1419 (< 1%)	1.6665 (< 1%)	-27.8207 (< 1%)	386.9947 (< 1%)	0.00922 (> 10%)

Table 17 Stationarity test results for the NetBeans *java* time series data, with a sampling period of 14 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-4.1489 (< 1%)	8.6086 (< 1%)	1.7996 (< 1%)	-14.8878 (< 1%)	110.8247 (< 1%)	0.04114 (> 10%)
Improvements	-10.6512 (< 1%)	56.7236 (< 1%)	0.62672 (< 1%)	-20.0450 (< 1%)	200.9024 (< 1%)	0.01392 (> 10%)
New Features	-8.3221 (< 1%)	34.6290 (< 1%)	0.57192 (> 2.5%)	-20.9486 (< 1%)	219.4221 (< 1%)	0.02217 (> 10%)

Table 18 Stationarity test results for the NetBeans *java* time series data, with a sampling period of 30 days.

Time Series	Un-differenced data			Differenced data		
	ADF (τ_2)	ADF(φ_1)	KPSS	ADF (τ_2)	ADF(φ_1)	KPSS
Bugs	-3.3551 (< 5%)	5.6322 (< 5%)	0.5672 (> 2.5%)	-8.6438 (< 1%)	37.3794 (< 1%)	0.07085 (> 10%)
Improvements	-6.1447 (< 1%)	18.8829 (< 1%)	0.1011 (> 10%)	-11.8473 (< 1%)	70.1811 (< 1%)	0.02910 (> 10%)
New Features	-4.1530 (< 1%)	8.6242 (< 1%)	0.7231 (> 1%)	-13.4034 (< 1%)	89.8285 (< 1%)	0.05939 (> 10%)

Appendix C: Exploratory Modeling Results

These are the results from running exploratory modeling, where a variety of sliding window parameters were evaluated to determine their effect on validity and accuracy.

Table 19 Results of the sliding window for various parameter values, using the MongoDB *core server* dataset, with a sampling period of 7 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
36	0	266	0.0263	0.0425	7.1088	0.8105	0.6694
39	0	263	0.0152	0.0386	6.7192	0.8353	0.7309
42	0	260	0.0154	0.0547	7.0477	0.8347	0.7025
45	0	257	0.0156	0.0593	6.8563	0.8487	0.7395
48	0	254	0.0079	0.0595	7.0984	0.8608	0.7384
51	0	251	0.012	0.0726	7.085	0.8478	0.7304
54	0	248	0.0242	0.095	7.0005	0.8767	0.758
57	0	245	0.0041	0.0984	7.0746	0.8636	0.7409
60	0	242	0	0.0826	7.2357	0.8514	0.7027
63	0	239	0	0.0962	7.1432	0.8565	0.75
66	0	236	0.0085	0.0769	7.4012	0.8843	0.7361
69	0	233	0.0129	0.0565	7.0468	0.871	0.7465
72	0	230	0.0043	0.0611	7.1442	0.8651	0.7581
75	0	227	0	0.0529	6.9642	0.8884	0.7907
78	0	224	0	0.0714	7.2621	0.875	0.7788
36	1	265	0.0038	0.053	7.3308	0.316	0.216
39	1	262	0	0.0687	7.3186	0.3402	0.2582
42	1	259	0	0.0888	7.2981	0.3178	0.2331
45	1	256	0	0.0898	7.2555	0.309	0.2318
48	1	253	0	0.0988	7.4097	0.2544	0.2061
51	1	250	0	0.084	7.4077	0.2358	0.2096
54	1	247	0.004	0.0772	7.4128	0.2467	0.185
57	1	244	0.0041	0.0823	7.3926	0.2601	0.1928
60	1	241	0.0041	0.0958	7.3429	0.2811	0.212
63	1	238	0.0042	0.097	7.471	0.2804	0.1963
66	1	235	0.0043	0.1154	7.5238	0.2319	0.1739
69	1	232	0	0.1164	7.6218	0.2927	0.1854

72	1	229	0	0.1223	7.713	0.2388	0.1741
75	1	226	0	0.146	7.797	0.2591	0.1969
78	1	223	0	0.1256	7.7931	0.2769	0.1846
36	2	264	0.0341	0.0588	9.7407	0.2417	0.1625
39	2	261	0.0421	0.08	9.5765	0.2348	0.1609
42	2	258	0.031	0.064	9.6554	0.2692	0.2009
45	2	255	0.0039	0.0591	9.75	0.2552	0.1674
48	2	252	0.004	0.0558	9.538	0.2278	0.1688
51	2	249	0.0161	0.049	9.5859	0.2747	0.1803
54	2	246	0.0203	0.0498	9.7917	0.2533	0.1747
57	2	243	0.0247	0.0295	9.4243	0.2826	0.2
60	2	240	0.0208	0.0255	9.1724	0.2882	0.1921
63	2	237	0.0169	0.0215	9.1596	0.2895	0.2018
66	2	234	0.0085	0.0259	9.2681	0.2257	0.1814
69	2	231	0.0043	0.0217	9.2867	0.2889	0.1956
72	2	228	0	0.0219	9.27	0.2735	0.2108
75	2	225	0	0.0133	9.2443	0.3063	0.2432
78	2	222	0.0045	0.009	9.1373	0.2922	0.2237

Table 20 Results of the sliding window for various parameter values, using the MongoDB *core server* dataset, with a sampling period of 14 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
24	0	127	0.126	0.0631	12.2365	0.7019	0.5962
27	0	124	0.1048	0.036	12.0656	0.7196	0.5888
30	0	121	0.0744	0.0625	12.2339	0.7714	0.6381
33	0	118	0.0508	0.0446	12.1686	0.7944	0.6355
36	0	115	0.0087	0.0439	12.5667	0.8165	0.6789
39	0	112	0.0089	0.045	12.251	0.8585	0.7075
42	0	109	0.0092	0.0278	12.5108	0.8381	0.7524
45	0	106	0	0.0377	12.7371	0.8627	0.7157
48	0	103	0	0.0388	12.7419	0.8485	0.7475
51	0	100	0	0.03	12.1728	0.8866	0.7732
54	0	97	0	0	13.0601	0.8866	0.7423
24	1	126	0.0238	0	14.723	0.3659	0.2764
27	1	123	0.0813	0	14.6283	0.292	0.2301
30	1	120	0.0667	0.0089	13.8263	0.2883	0.2072
33	1	117	0.0598	0	13.7754	0.3273	0.2273

36	1	114	0.0439	0.0092	13.721	0.2963	0.213
39	1	111	0.036	0.0093	13.674	0.3019	0.217
42	1	108	0.0093	0.0093	13.7645	0.2642	0.2264
45	1	105	0.0095	0.0385	13.8128	0.26	0.22
48	1	102	0.0196	0.02	14.0921	0.3163	0.2143
51	1	99	0.0808	0.011	14.6563	0.2222	0.2
54	1	96	0.1354	0.0241	15.2364	0.2593	0.2099
24	2	125	0.136	0	19.1918	0.2315	0.1759
27	2	122	0.1557	0.0097	19.3405	0.2451	0.1471
30	2	119	0.1597	0	18.7677	0.29	0.19
33	2	116	0.181	0	18.7744	0.2	0.1474
36	2	113	0.1681	0.0106	17.9884	0.2688	0.2151
39	2	110	0.1364	0.0421	17.6966	0.1648	0.0989
42	2	107	0.0935	0.0206	17.8889	0.2316	0.1684
45	2	104	0.0481	0.0303	17.9562	0.2708	0.1667
48	2	101	0.0198	0.0404	17.6384	0.2	0.1368
51	2	98	0.0102	0.0825	18.1619	0.2472	0.1573
54	2	95	0.0316	0.0978	18.7985	0.2651	0.2169

Table 21 Results of the sliding window for various parameter values, using the MongoDB *core server* dataset, with a sampling period of 30 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
12	0	58	0.1897	0.1702	30.1084	0.5385	0.4103
15	0	55	0.1818	0.0444	29.8198	0.5349	0.4651
18	0	52	0.25	0	29.9369	0.5128	0.3846
21	0	49	0.2449	0.027	31.3845	0.5	0.3611
24	0	46	0.087	0.0952	28.7562	0.6053	0.4737
27	0	43	0.1395	0.1622	31.7467	0.5806	0.4839
30	0	40	0.15	0.0882	30.558	0.7097	0.5161
33	0	37	0.1892	0.2	31.319	0.5833	0.375
36	0	34	0.0882	0.1935	32.1424	0.6	0.44
12	1	57	0.1579	0.0833	38.4833	0.3409	0.2955
15	1	54	0.0556	0.1765	33.5892	0.4048	0.3095
18	1	51	0.1569	0.2326	29.4009	0.2121	0.1212
21	1	48	0.1875	0.2308	34.4193	0.2333	0.1667
24	1	45	0.0889	0.0732	34.4471	0.4211	0.2895
27	1	42	0.119	0	32.8154	0.3784	0.2432

30	1	39	0.2308	0.0333	34.0206	0.5172	0.3103
33	1	36	0.25	0.037	35.4527	0.4231	0.2692
36	1	33	0	0.0606	34.1955	0.4194	0.2581
12	2	56	0.1607	0	46.1684	0.1915	0.1489
15	2	53	0.0566	0	48.0847	0.16	0.14
18	2	50	0.08	0	47.4011	0.2174	0.1522
21	2	47	0.0851	0	50.933	0.186	0.093
24	2	44	0.1136	0	49.4234	0.1795	0.1026
27	2	41	0.1463	0	47.5708	0.1143	0.0857
30	2	38	0.1579	0.0312	40.017	0.1935	0.0968
33	2	35	0.2	0	43.9241	0.2143	0.1071
36	2	32	0.0625	0	49.3429	0.0333	0.0333

Table 22 Results of the sliding window for various parameter values, using the Hibernate *orm* dataset, with a sampling period of 7 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
36	0	592	0.0338	0.0437	3.7751	0.8921	0.7971
39	0	589	0.039	0.0424	3.8057	0.8985	0.7823
42	0	586	0.0666	0.0256	3.7109	0.9099	0.803
45	0	583	0.0858	0.0263	3.7575	0.9056	0.817
48	0	580	0.0345	0.0268	3.6948	0.9156	0.811
51	0	577	0.0347	0.0305	3.6648	0.9167	0.8167
54	0	574	0.0505	0.0312	3.7317	0.8996	0.8239
57	0	571	0.0578	0.0428	3.701	0.9107	0.8078
60	0	568	0.044	0.0442	3.6616	0.9191	0.8189
63	0	565	0.0531	0.0505	3.7796	0.9094	0.8051
66	0	562	0.0623	0.0512	3.8422	0.908	0.818
69	0	559	0.0716	0.0636	3.8496	0.9218	0.8333
72	0	556	0.0629	0.0691	3.764	0.9237	0.8227
75	0	553	0.0633	0.0714	3.7777	0.9148	0.8254
78	0	550	0.0618	0.0659	3.7236	0.9129	0.8423
36	1	591	0.0558	0	4.0317	0.3495	0.2455
39	1	588	0.068	0.0036	4.0605	0.337	0.2418
42	1	585	0.0547	0.0072	4.0237	0.3206	0.2277
45	1	582	0.0533	0.0036	3.9689	0.3224	0.235
48	1	579	0.0501	0.0073	3.869	0.3132	0.2418
51	1	576	0.0556	0.0074	3.8782	0.3074	0.2352

54	1	573	0.0593	0.0111	3.9158	0.2889	0.227
57	1	570	0.0702	0.0132	3.8293	0.3002	0.2199
60	1	567	0.06	0.0169	3.9171	0.3034	0.2137
63	1	564	0.0762	0.0115	3.9501	0.3029	0.2369
66	1	561	0.082	0.0155	3.9868	0.3097	0.2189
69	1	558	0.0878	0.0196	3.9754	0.2866	0.2184
72	1	555	0.0847	0.0276	3.9352	0.3097	0.2308
75	1	552	0.0924	0.018	3.9101	0.2988	0.2134
78	1	549	0.0838	0.0219	3.9379	0.3008	0.2276
36	2	590	0.1746	0.0719	5.4174	0.3341	0.2456
39	2	587	0.1942	0.0613	5.6012	0.3243	0.2117
42	2	584	0.2089	0.0584	5.4871	0.3425	0.2345
45	2	581	0.2306	0.0559	5.3992	0.3483	0.2607
48	2	578	0.1972	0.0517	5.1401	0.3568	0.275
51	2	575	0.2261	0.0764	4.9943	0.3382	0.2652
54	2	572	0.215	0.0958	4.7969	0.3498	0.2709
57	2	569	0.2478	0.0748	4.7112	0.3636	0.2803
60	2	566	0.2032	0.0754	4.8994	0.3573	0.2806
63	2	563	0.2114	0.0788	5.1818	0.3301	0.2396
66	2	560	0.225	0.076	4.8411	0.3441	0.2718
69	2	557	0.228	0.0651	5.0767	0.3657	0.2761
72	2	554	0.2184	0.0785	4.9937	0.3759	0.2607
75	2	551	0.2123	0.0876	4.9523	0.3409	0.2449
78	2	548	0.2172	0.0816	4.9851	0.3477	0.2335

Table 23 Results of the sliding window for various parameter values, using the Hibernate *orm* dataset, with a sampling period of 14 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
24	0	290	0.0655	0.0185	6.0408	0.8271	0.7368
27	0	287	0.0732	0.0338	6.2294	0.821	0.7198
30	0	284	0.088	0.0463	6.3016	0.8381	0.7368
33	0	281	0.0712	0.0345	6.1508	0.877	0.75
36	0	278	0.0432	0.0338	6.0623	0.8833	0.7588
39	0	275	0.0473	0.0573	6.0728	0.8907	0.7571
42	0	272	0.0772	0.0797	6.1146	0.8918	0.7619
45	0	269	0.0855	0.1057	6.2106	0.8727	0.7409
48	0	266	0.0489	0.1186	5.9274	0.8969	0.7578

51	0	263	0.0456	0.1315	5.8937	0.8853	0.7569
54	0	260	0.0462	0.1411	6.2151	0.8873	0.7418
24	1	289	0.0692	0.0297	6.0855	0.3563	0.2644
27	1	286	0.0629	0.0187	5.9908	0.365	0.2586
30	1	283	0.0777	0.0192	5.9715	0.4062	0.293
33	1	280	0.0893	0.0314	5.977	0.3887	0.2996
36	1	277	0.0903	0.0278	6.0303	0.3878	0.2898
39	1	274	0.0985	0.0324	6.0421	0.3891	0.2929
42	1	271	0.1181	0.0418	6.1093	0.4148	0.3144
45	1	268	0.1194	0.0254	6.0003	0.4174	0.2913
48	1	265	0.1472	0.0531	5.9811	0.3598	0.2664
51	1	262	0.1489	0.0807	6.0572	0.3561	0.2537
54	1	259	0.1622	0.0922	6.0686	0.3655	0.264
24	2	288	0.2014	0.0304	8.8933	0.2691	0.1973
27	2	285	0.1544	0.0456	8.6314	0.2565	0.1783
30	2	282	0.1773	0.069	8.6628	0.2685	0.1806
33	2	279	0.1828	0.0658	8.3941	0.2394	0.1549
36	2	276	0.0942	0.04	8.3626	0.275	0.1958
39	2	273	0.1209	0.0167	8.6061	0.2627	0.2119
42	2	270	0.1704	0.0089	8.3573	0.2387	0.1622
45	2	267	0.1723	0.0136	8.3428	0.2202	0.133
48	2	264	0.178	0.023	8.2138	0.2547	0.1981
51	2	261	0.1992	0.0144	7.9015	0.267	0.1748
54	2	258	0.2209	0.0199	8.0545	0.3046	0.2132

Table 24 Results of the sliding window for various parameter values, using the Hibernate *orm* dataset, with a sampling period of 30 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
12	0	134	0.2537	0.05	10.5601	0.8105	0.7474
15	0	131	0.2137	0.0291	9.5438	0.85	0.72
18	0	128	0.2266	0.0404	9.5969	0.8421	0.6947
21	0	125	0.264	0.0217	10.2667	0.8111	0.7
24	0	122	0.0656	0.0702	9.6204	0.8396	0.7075
27	0	119	0.0756	0.0636	9.7352	0.767	0.6505
30	0	116	0.0862	0.0849	9.9829	0.7835	0.7113
33	0	113	0.0973	0.0588	9.1539	0.8646	0.75
36	0	110	0.0455	0.1143	9.5042	0.8817	0.7204

12	1	133	0.1654	0	11.1269	0.4775	0.4144
15	1	130	0.1692	0	11.1249	0.463	0.3333
18	1	127	0.1969	0	10.9949	0.4216	0.3333
21	1	124	0.2339	0.0105	10.5803	0.4787	0.383
24	1	121	0.0413	0.0086	10.2685	0.5391	0.4522
27	1	118	0.0339	0.0175	10.8562	0.4732	0.4375
30	1	115	0.0435	0.0182	10.3656	0.537	0.4259
33	1	112	0.0446	0.0187	10.4198	0.4667	0.3619
36	1	109	0.0183	0.0093	10.3279	0.4434	0.3302
12	2	132	0.2348	0.099	16.3225	0.3736	0.2857
15	2	129	0.2248	0.08	18.469	0.3261	0.2283
18	2	126	0.2778	0.0769	17.1721	0.3214	0.2738
21	2	123	0.3496	0.0625	16.7591	0.3733	0.3067
24	2	120	0.2167	0.0532	15.9745	0.2697	0.2135
27	2	117	0.265	0.0349	15.4464	0.2289	0.1687
30	2	114	0.2719	0.0361	15.8677	0.1875	0.1375
33	2	111	0.2523	0.0361	14.4222	0.25	0.2
36	2	108	0.1574	0.0549	14.6457	0.2674	0.1512

Table 25 Results of the sliding window for various parameter values, using the NetBeans *platform* dataset, with a sampling period of 7 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
36	0	459	0.0654	0.0909	9.86	0.8897	0.7949
39	0	456	0.0636	0.0913	9.5371	0.8995	0.8093
42	0	453	0.0574	0.1148	9.5295	0.9074	0.8228
45	0	450	0.0667	0.1357	9.7326	0.8981	0.8402
48	0	447	0.0805	0.1484	9.9171	0.8943	0.8257
51	0	444	0.0901	0.1683	9.6058	0.9137	0.8304
54	0	441	0.093	0.1875	9.4234	0.9015	0.8369
57	0	438	0.0822	0.209	9.5227	0.9088	0.8176
60	0	435	0.0759	0.2463	9.0788	0.9175	0.8284
63	0	432	0.0903	0.2646	9.1516	0.917	0.8304
66	0	429	0.0816	0.2741	8.7717	0.9301	0.8636
69	0	426	0.0798	0.2832	9.0589	0.9253	0.8505
72	0	423	0.078	0.3103	8.5253	0.948	0.855
75	0	420	0.0738	0.3085	8.5665	0.9405	0.8662
78	0	417	0.0791	0.362	8.6697	0.9429	0.8694

36	1	458	0.0786	0.1232	9.6252	0.3784	0.2568
39	1	455	0.0659	0.1176	9.4768	0.352	0.2453
42	1	452	0.0774	0.1175	9.5606	0.356	0.2636
45	1	449	0.0935	0.1327	9.6163	0.3144	0.2408
48	1	446	0.0874	0.1425	9.4862	0.3324	0.2636
51	1	443	0.0609	0.125	9.2261	0.3489	0.261
54	1	440	0.0568	0.159	9.3312	0.3582	0.2464
57	1	437	0.0526	0.1618	9.078	0.366	0.2824
60	1	434	0.0507	0.1699	9.0127	0.3596	0.2778
63	1	431	0.0603	0.1852	8.8855	0.3485	0.2576
66	1	428	0.0537	0.1975	8.8611	0.3538	0.2523
69	1	425	0.0635	0.1985	8.9273	0.3605	0.2821
72	1	422	0.0687	0.2061	8.4446	0.3846	0.2917
75	1	419	0.0501	0.2161	8.2176	0.3558	0.2564
78	1	416	0.0529	0.2234	8.2652	0.3562	0.2647
36	2	457	0.1422	0.0281	12.885	0.294	0.2047
39	2	454	0.1586	0.0209	12.8026	0.3048	0.2273
42	2	451	0.1663	0.0319	12.865	0.2582	0.1758
45	2	448	0.1585	0.0345	12.836	0.3022	0.2115
48	2	445	0.1393	0.0287	12.2567	0.3172	0.2151
51	2	442	0.1335	0.0261	12.213	0.3083	0.2225
54	2	439	0.1412	0.0239	11.746	0.3043	0.2418
57	2	436	0.156	0.038	11.6564	0.3418	0.2655
60	2	433	0.1455	0.0568	11.382	0.3209	0.2521
63	2	430	0.1535	0.0604	11.2337	0.3363	0.2661
66	2	427	0.1639	0.056	10.9923	0.3175	0.2552
69	2	424	0.1557	0.0782	10.6024	0.3303	0.2242
72	2	421	0.1615	0.0878	10.4374	0.3665	0.2702
75	2	418	0.1722	0.0838	9.9606	0.3375	0.2587
78	2	415	0.188	0.1128	10.1103	0.3077	0.2174

Table 26 Results of the sliding window for various parameter values, using the NetBeans *platform* dataset, with a sampling period of 14 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
24	0	223	0.0493	0.0425	19.3307	0.8867	0.7882
27	0	220	0.05	0.0526	15.816	0.904	0.798
30	0	217	0.0599	0.0686	15.7743	0.9053	0.8053

33	0	214	0.0561	0.0545	14.8422	0.9162	0.8115
36	0	211	0.0427	0.0941	14.7299	0.9454	0.847
39	0	208	0.0385	0.125	14.5529	0.9543	0.88
42	0	205	0.0146	0.1584	14.7545	0.9412	0.8471
45	0	202	0.0396	0.201	14.0061	0.9548	0.8839
48	0	199	0.0302	0.2435	15.2696	0.9452	0.8836
51	0	196	0.0357	0.2646	14.7779	0.9353	0.9065
54	0	193	0.0415	0.2865	15.0171	0.9318	0.9091
24	1	222	0.0991	0.04	16.4695	0.4635	0.3438
27	1	219	0.0959	0.0253	15.2702	0.4611	0.3938
30	1	216	0.1111	0.0156	15.9387	0.3757	0.328
33	1	213	0.0939	0.0155	15.968	0.3947	0.2842
36	1	210	0.0762	0.0258	15.7459	0.3915	0.3069
39	1	207	0.0821	0.0263	15.2116	0.3243	0.2919
42	1	204	0.0784	0.0266	15.1269	0.3497	0.235
45	1	201	0.0796	0.027	14.038	0.3667	0.2889
48	1	198	0.0657	0.0324	14.2528	0.3408	0.2849
51	1	195	0.0821	0.0279	14.6695	0.2989	0.2414
54	1	192	0.099	0.0173	14.8401	0.3176	0.2471
24	2	221	0.1991	0.0565	22.0721	0.2814	0.2036
27	2	218	0.1835	0.0506	20.127	0.3669	0.2544
30	2	215	0.186	0.0229	21.079	0.345	0.2632
33	2	212	0.2311	0.0491	19.8758	0.2774	0.1871
36	2	209	0.2392	0.0503	19.6064	0.3311	0.2781
39	2	206	0.2379	0.0637	19.7054	0.3197	0.2517
42	2	203	0.2414	0.0649	19.7021	0.3264	0.2708
45	2	200	0.27	0.0822	19.4454	0.3284	0.2687
48	2	197	0.2335	0.106	18.2789	0.3481	0.2741
51	2	194	0.299	0.1029	18.8858	0.3115	0.2213
54	2	191	0.3246	0.1163	18.9413	0.3596	0.2807

Table 27 Results of the sliding window for various parameter values, using the NetBeans *platform* dataset, with a sampling period of 30 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
12	0	103	0.1942	0	43.6864	0.7229	0.6024
15	0	100	0.16	0.0238	34.8502	0.8171	0.622
18	0	97	0.2062	0.0519	32.3821	0.8082	0.7123

21	0	94	0.1915	0.0658	34.5304	0.7465	0.6761
24	0	91	0.011	0.1111	36.6959	0.725	0.6625
27	0	88	0	0.1364	33.2856	0.7368	0.6184
30	0	85	0.0118	0.0833	33.3657	0.7662	0.6753
33	0	82	0.0122	0.0741	33.0471	0.8267	0.68
36	0	79	0	0.0506	31.0204	0.84	0.7333
12	1	102	0.098	0.0326	39.2089	0.4045	0.3371
15	1	99	0.1212	0.069	35.8939	0.4444	0.3827
18	1	96	0.0729	0.0225	36.2631	0.3908	0.3103
21	1	93	0.0753	0.0581	34.3627	0.3827	0.2963
24	1	90	0.0111	0.1124	36.8309	0.3418	0.2532
27	1	87	0.0115	0.1512	37.2506	0.3836	0.2603
30	1	84	0	0.1548	37.2207	0.3239	0.2254
33	1	81	0	0.2222	39.2093	0.2698	0.1587
36	1	78	0	0.2308	37.0815	0.3333	0.25
12	2	101	0.198	0	55.6171	0.3457	0.2346
15	2	98	0.2347	0.0267	51.4706	0.274	0.2055
18	2	95	0.2316	0	52.889	0.3562	0.2466
21	2	92	0.2065	0	47.3575	0.2192	0.1507
24	2	89	0.0674	0	47.1952	0.241	0.1325
27	2	86	0.0465	0	46.1986	0.2317	0.1585
30	2	83	0.0482	0	46.4639	0.2405	0.1772
33	2	80	0.05	0.0132	42.8021	0.2133	0.1067
36	2	77	0.039	0	44.603	0.2162	0.1757

Table 28 Results of the sliding window for various parameter values, using the NetBeans *java* dataset, with a sampling period of 7 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
36	0	458	0.0328	0.1783	11.5142	0.9093	0.8297
39	0	455	0.0286	0.1833	11.6885	0.9114	0.8338
42	0	452	0.031	0.1872	11.3673	0.9326	0.8624
45	0	449	0.0267	0.1854	8.495	0.9326	0.8511
48	0	446	0.0471	0.1788	9.0924	0.9255	0.8596
51	0	443	0.0519	0.1929	8.3931	0.941	0.8791
54	0	440	0.0682	0.1854	9.0207	0.9431	0.8832
57	0	437	0.0824	0.197	8.6575	0.9441	0.8789
60	0	434	0.0691	0.203	8.3238	0.9503	0.8851

63	0	431	0.0742	0.2281	8.8945	0.9416	0.8636
66	0	428	0.0794	0.2487	8.3348	0.9459	0.8682
69	0	425	0.0729	0.2843	8.3855	0.9504	0.8723
72	0	422	0.0711	0.3112	8.1105	0.9556	0.8778
75	0	419	0.0644	0.3444	8.3474	0.9689	0.8794
78	0	416	0.0457	0.3552	8.0082	0.957	0.8945
36	1	457	0.0306	0.1174	9.6908	0.3785	0.289
39	1	454	0.0396	0.0963	9.6353	0.3909	0.2868
42	1	451	0.0421	0.1019	9.628	0.3376	0.2448
45	1	448	0.0335	0.1039	9.4252	0.3531	0.2577
48	1	445	0.0292	0.1181	9.6122	0.3675	0.2782
51	1	442	0.0249	0.1183	9.6928	0.3368	0.2605
54	1	439	0.0478	0.0909	9.6931	0.3289	0.2553
57	1	436	0.0665	0.0983	9.6654	0.3597	0.2807
60	1	433	0.0462	0.092	8.9048	0.3893	0.2773
63	1	430	0.0581	0.1111	8.6984	0.3444	0.2528
66	1	427	0.0468	0.1302	8.6851	0.3672	0.2655
69	1	424	0.0448	0.158	8.8131	0.3314	0.2551

Table 29 Results of the sliding window for various parameter values, using the NetBeans *java* dataset, with a sampling period of 14 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
24	0	223	0.148	0.1316	18.4175	0.903	0.8364
27	0	220	0.1455	0.1809	15.5545	0.9156	0.8182
30	0	217	0.1475	0.2108	13.2803	0.9247	0.8699
33	0	214	0.1355	0.2432	14.8439	0.9	0.8643
36	0	211	0.1469	0.3111	13.3972	0.9435	0.871
39	0	208	0.1442	0.3427	14.9439	0.9316	0.8718
42	0	205	0.1366	0.3277	15.3356	0.9328	0.8571
45	0	202	0.1733	0.3952	15.8706	0.9505	0.8812
48	0	199	0.1759	0.4268	14.7681	0.9574	0.9255
51	0	196	0.1735	0.4568	13.9321	0.9659	0.9205
54	0	193	0.1813	0.4937	14.5164	0.9625	0.9125
24	1	222	0.1171	0.0561	19.7705	0.427	0.3027
27	1	219	0.0913	0.1206	18.0539	0.3657	0.2857
30	1	216	0.1296	0.1489	18.0469	0.4312	0.3062
33	1	213	0.1408	0.1694	17.9844	0.3684	0.3158

36	1	210	0.1476	0.1955	17.6171	0.3889	0.3056
39	1	207	0.1304	0.1889	17.16	0.4247	0.2877
42	1	204	0.1225	0.2235	17.1311	0.446	0.3381
45	1	201	0.1343	0.2299	17.5275	0.4179	0.3284
48	1	198	0.1061	0.226	17.1213	0.4015	0.3066
51	1	195	0.0769	0.2389	16.7823	0.438	0.3358
54	1	192	0.0833	0.3523	17.5911	0.386	0.3158
24	2	221	0.2398	0.0536	26.4102	0.2642	0.2013
27	2	218	0.2385	0.0602	24.629	0.2756	0.1923
30	2	215	0.2279	0.0843	24.7462	0.2434	0.1711
33	2	212	0.2594	0.1019	24.0487	0.2837	0.1986
36	2	209	0.1866	0.1294	25.3042	0.277	0.2027
39	2	206	0.1796	0.1657	26.0777	0.2482	0.1702
42	2	203	0.1724	0.1845	26.7409	0.2263	0.1679
45	2	200	0.17	0.1627	25.4501	0.2446	0.1727
48	2	197	0.1726	0.1411	24.9162	0.2571	0.2
51	2	194	0.1907	0.1465	23.1309	0.2761	0.194
54	2	191	0.1675	0.1635	21.4298	0.2932	0.2481

Table 30 Results of the sliding window for various parameter values, using the NetBeans *java* dataset, with a sampling period of 30 days.

Window Size	Diff. Degree	Window Count	None Valid	Non-normal	RMSE	In 90% Interval	In 75% Interval
12	0	103	0.4563	0.0357	64.1359	0.7778	0.6852
15	0	100	0.36	0.0156	54.8333	0.8413	0.746
18	0	97	0.3299	0.0769	52.7232	0.85	0.7833
21	0	94	0.2447	0.1127	24.6878	0.9524	0.9206
24	0	91	0.1648	0.1316	27.4929	0.9394	0.8485
27	0	88	0.125	0.1558	27.3194	0.9692	0.8615
30	0	85	0.1294	0.1351	39.1019	0.9531	0.9062
33	0	82	0.1341	0.1831	41.7956	0.9138	0.8966
36	0	79	0.1646	0.1364	42.6994	0.9123	0.8772
12	1	102	0.0882	0.043	55.804	0.382	0.3371
15	1	99	0.0808	0.0659	38.211	0.4941	0.3647
18	1	96	0.0417	0.1304	31.0359	0.425	0.3125
21	1	93	0.043	0.1461	35.8527	0.4079	0.3421
24	1	90	0.0556	0.1294	42.9426	0.4054	0.2838
27	1	87	0.0575	0.1707	39.9849	0.3824	0.2794

30	1	84	0.0833	0.1039	40.1486	0.4203	0.3188
33	1	81	0.1358	0.1	40.234	0.4444	0.3175
36	1	78	0.1538	0.0455	39.5017	0.3492	0.1905
12	2	101	0.1386	0.0115	60.3041	0.3256	0.2209
15	2	98	0.1735	0.0123	59.1983	0.35	0.2625
18	2	95	0.1368	0.0732	53.2988	0.2895	0.2237
21	2	92	0.163	0.1299	44.2046	0.3582	0.2836
24	2	89	0.1461	0.0921	43.3983	0.3768	0.3188
27	2	86	0.1628	0.1111	39.5985	0.2812	0.2656
30	2	83	0.1566	0.1286	45.954	0.3115	0.2131
33	2	80	0.1875	0.1385	46.0134	0.2321	0.1964
36	2	77	0.1688	0.1094	44.9917	0.2807	0.193