UNIVERSITY OF CALIFORNIA

SANTA CRUZ

SOFTWARE SELECTION FOR RELIABILITY OPTIMIZATION USING TIME SERIES ANALYSIS AND MACHINE LEARNING

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

TECHNOLOGY AND INFORMATION MANAGEMENT

by

YALI MU

March 2013

The Thesis of YALI MU
is approved:

_____

Subhas Desa, Chair

_____

Patrick Mantey

_____

Arnav Jhala

_____

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Software Selection for Reliability Optimization using Time Series Analysis and Machine Learning

by

Yali Mu

A software product family generally has multiple product lines or software versions, each with, in general, a different functionality. Also with new software versions, functionality usually measured by the number of features, in general, increases. The availability of several versions of a software family raises several important technical issues related to trade-off between software functionality and software reliability.

In this thesis, we consider the following issue: given a family of different versions of the same software product, how does one select versions of the software that maximize reliability, are measured by the number of software bugs. To resolve this issue, we develop a framework integrating time series analysis and machine learning to classify software versions of a given family into clusters based on user-specified reliability metrics. This classification or separation then enables the user to optimize the selection of software for maximum reliability.

Time series analysis, in particular is used to predict the future evolution of bugs. Machine learning methods, in particular Expectation-Maximization clustering and K-means clustering are applied to prediction-based metrics to classify software into clusters. We demonstrate the application of this framework to reliability prediction, classification and optimization of network IOS products.

ACKNOWLEDGMENTS

First and foremost I wish to express my sincere gratitude to my advisor, DR. Subhas Desa for being a mentor and guide throughout the course of my research work. During my time at the University of California, Santa Cruz, Subhas has been a steady source of motivation and guidance.

Furthermore, I would like to thank the people I have worked with at Cisco for all their support and advice, in particular Chris Wong, Jarrett Fishpaw, Jiabin Zhao, Karen Zheng, and Brandon Virgin of the Smart Analytics Group.

I am also indebted to my parents for being my role models and an endless source of love and encouragement throughout my academic endeavours. I thank my husband and the rest of my family for providing me with a constant source of support.

I would like to thank my committee members Professors Patrick Mantey and Arnav Jhala for reviewing of my dissertation and providing insightful comments and suggestions.

The UCSC Network Management and Operations (NMO) Lab provided financial support for this research.

1 Introduction

1.1 Background and Motivation

A software product family generally has multiple product lines or software versions, each with, in general, a different functionality. As a result, the number of defects or "bugs" in these software versions accumulates rapidly. Bug data for a given software version is extremely important both for assessing software performance and for selection of the most reliable software version in a given software product family.

Software bugs are tracked in the organization database either from customers or from error reporting software. Accumulated bug reports contain valuable information that can be used to predict future bug trends, measure the reliability of software, and optimize the selection of software based on its reliability. In this thesis, we use methods from time series data analysis to estimate the number of bugs will be in the future. Using time series analysis, we get results from the prediction of defect data for every software version, and use these results to help us select the software version that is most reliable.

For every hardware system, there are many software versions for us to choose from, and sometimes it is hard for both customers and companies to choose the right software. Besides choosing software from the aspect of the feature sets it provides, we also need to optimize our selection by measuring how reliable the software is. Then we choose the "best" software from many software versions. In this thesis, we show how to use forecasting methods to measure reliability of each software version, and then using machine learning methods we show how to group different software versions in the same family into different clusters based on a selected prediction metric.

1.2 Thesis Contributions

The objective of the thesis is to optimize the selection of software with respect to reliability. We use existing time-history of accumulated detected software bugs to

predict the future evolution of bugs. We use time series methods as the means for predicting the accumulated number of bug. We then use machine learning methods to cluster the software version into different groups. The thesis contributions are as follows:

1. The use of time series analysis to predict the time history of bugs.

2. The use of machine learning techniques, in particular Expectation-Maximization clustering and K-means clustering, to classify software versions into different categories based on user-selected reliability metrics.

3. A framework that integrates data preparation, time series analysis and machine learning to optimize software selection for maximum reliability.

4. The application of this framework to the classification and selection of the most reliable software for network IOS software.

1.3 Structure of the Thesis

The remainder of this thesis is structured as follows. In section 2, this thesis will develop the theory it uses. The theory combines time series analysis and machine learning together to fulfil the goal of optimizing software selection. The approach discussed in section 2.1 gives an overview of the thesis and how the sections are related to each other. Then the three steps of the approach, data preparation, time series analysis, and clustering, are developed, respectively in section 2.2, 2.3 and 2.4. The defect data transformation is the pre-processing of raw data and getting the data into right format for prediction. Time series forecasting work describes how to use two time series methods to do prediction of the defect data. Then in optimization of software selection, we illustrate how we use the metric for clustering and do the optimization of software selection in the end. The experiments showing the effectiveness of our approaches is given in section 3.3, where the results of optimization are detailed. Finally, section 4 discusses future research directions conclusions.

2 Theory

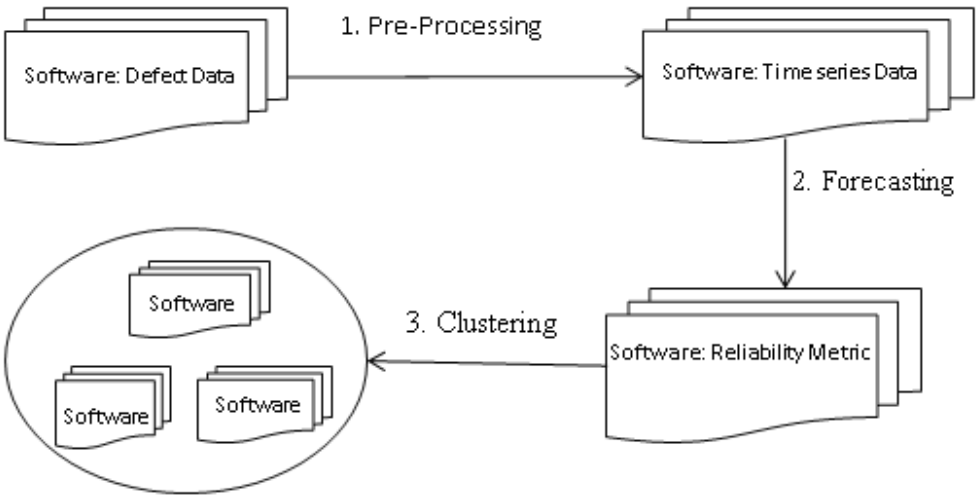
2.1 Approach

In this section, we will give an overview of this project and briefly talk about the approach to the goal. The approaches are divided into three major steps:

1. Data Preparation
2. Time Series Analysis
3. Clustering Algorithms

This process is shown in Figure 1 below. It represents the whole process for selecting reliable software inside one family platform.

Figure 1: Overview of the software Grouping



Step 1: The data preparation involves obtaining the raw data from database and doing feature selection based on the requirements for data, and also preprocessing to get the data into the right format for later steps. For different datasets, there are different ways of preprocessing. This step is the preparation step of data transformation for time series forecasting. The data we use are log files containing the defect data of every software release. It needs certain steps to put it into time series format.

Step 2: We use time series model to do the prediction based on historical data. After the model is tested and proved to be accurate. It then applied to all the data

in the log file and get prediction results for every software version. The prediction results are used to form up the metric we need for clustering the software. Because the metric contains information directly related to defect number and therefore is associated with the reliability of the software. This step is time series forecasting, which produces the reliability metrics that will be used in the later step.

Step 3: This step takes the results from the previous steps and produces the final results. By using the prediction results, we measure the reliability of every software and select one group of software which is most reliable by doing clustering to software inside one family. We used machine learning techniques that is clustering in our approach. The clustering methods we used are Expectation-Maximization (EM) algorithm and K-Means algorithm. They are used to group different software inside one family. The group which appears to be most reliable inside one family is recommended for use. In this way, we will help the software consulting engineers select the most reliable versions of software based on the customer needs.

Based on the three steps approach, we divided this section into three subsections below. Each subsection will illustrate the step in details and explain how the three steps are related and cooperated together to get the final results.

2.2 Data Preparation

This step is preparation step for time series forecasting. In order to get a good forecasting of the bugs for software, we need to put the bug data into time series format. For different companies, the bugs are stored in different ways, and there are some process that need to be done to get the data into time series format. The procedure includes following steps:

1. Get the defect data of the software, which contains the relevant information of the bug and its effected software versions and also its occurring date.

2. Transform the data into the format of software version and corresponding number of bugs happened every day.

3. Further process the bug data of every software version into number of bugs

happened every week.

After these steps, the data is in the format we need and can be used to do the prediction. This is a general concept of what we may do in the preprocessing step, things will change according to the format of the original dataset we get from database. In the later implementation step, we will talk in detail about what we did in out project towards our software datasets.

2.3 Time Series Analysis

The approach to forecasting is using time series data analysis to predict the bug number and distribution ahead of time to get the results that will be useful in the analysis of the software. The time series analysis is used because the bug numbers are coming up along with time and therefore we processed them into weekly time series data. In the first subsection, the thesis will talk about the datasets we used in this project after the preprocessing. Then it will illustrate two major methods to do the prediction. The first method introduced is the Box and Jenkins methodology, which includes autoregressive (AR) models, the integrated (I) models, and the moving average (MA) models. These three classes depend linearly on previous data points. Combinations of these ideas produce autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models [1]. Another method described in this thesis is called Exponential Smoothing, which also has many models and can be combined with state space models to improve forecasting results [2]. The state space models were originally developed for estimation and control in space applications. These models can efficiently fit those bug data, and have good predictions of them.

2.3.1 Time Series Dataset

A time series is defined as an ordered sequence of data points with respect to time. Time series have traditionally been used in the field of econometrics, but nowadays, it has many applications in a wide range of fields. The time series datasets we use for the experimental results in this thesis are the accumulated number of bugs per

week for some version of software released by Cisco Systems. For each software version, there is a dataset representing the number of reported bugs over time. We use 16 datasets to train our forecasting model. These datasets are plotted in Figure 2. In order to illustrated the forecasting methods used we will focus our discussion on dataset version "12.4(24)T" plotted in Figure 3. We will refer to this dataset as "Data1" in later discussions. It should be noted from Figure 2, that software bug data is by their nature non-seasonal. As a result we will not touch upon seasonality considerations in our model.

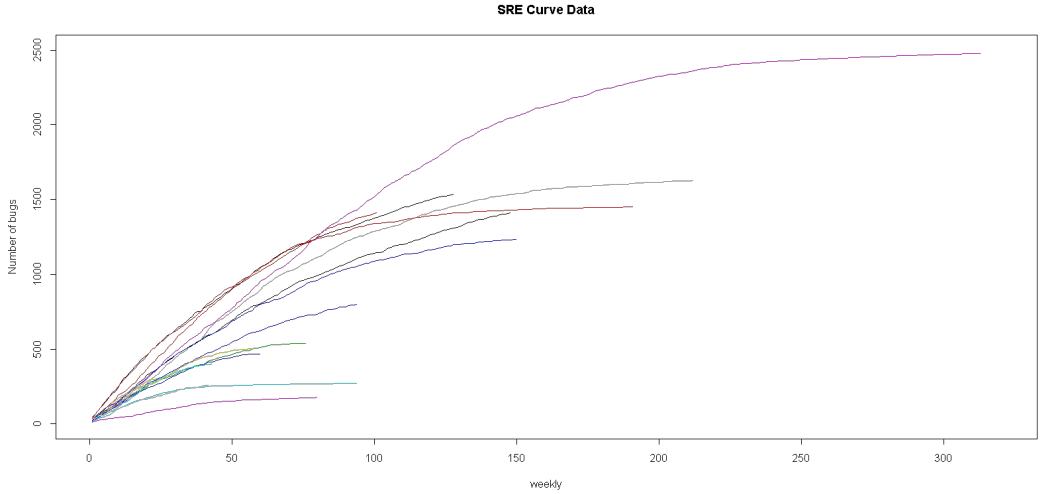


Figure 2: Accumulated number of bugs for software.

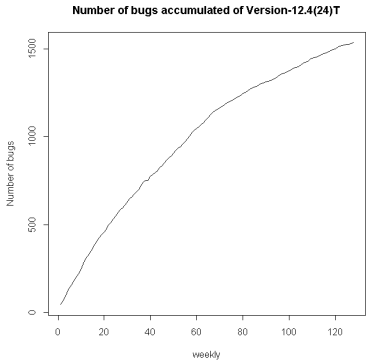


Figure 3: Accumulated number of bugs for "Data1".

Decomposition

Time series decomposition decomposes a time series into trend, seasonal, cyclical and irregular components. The trend component accounts for the long term trend

of the data. The seasonal component represents seasonal variations while the cyclical component considers repeated but non-periodic fluctuations in the data. Finally, the residuals can be seen to be the unpredictable irregular changes in the data collected. As a first step in extracting each of these components, we can use a moving average to pre-process the time series. This will filter out any random "white noise" from the data, to make the time series smoother and puts emphasis on certain informational components contained in the time series. The equation for moving average is given by:

$$T_t = \frac{1}{2a+1} \sum_{k=-a}^{a} X_{t+k}$$

where t means the time and 2a+1 are the last observed data points for the moving average interval. Applying this equation to time series, we can get the trend of the dataset. Another possibility for evaluating the trend of a time series is to use nonparametric regression techniques. Many statistical software packages define methods to decompose time series data using various established techniques [3].

Stationary and Non-stationary

The prevailing assumption of time series analysis, according to the classical linear regression model, is that the residuals of the estimated equations are stochastically independent from each other. This means that its statistical properties are essentially constant through time [3]. We call data series which satisfy this property stationary. Methods like Regression, Autoregressive, and Moving Average and mixed methods all require that the dataset is stationary. We can check this property using decomposition methods. If the trend component of the residual is going up or down all the way, then the time series is non-stationary, but if the trend fluctuates with a constant variation, the model is most likely to be stationary. Modelling methods like ARIMA and exponential smoothing can also process non-stationary time series.

2.3.2 Box-Jenkins Models

This subsection introduces the Box-Jenkins models, by first explaining how these models are built and then how to test these models and produce forecasts. Later, it illustrates some sample models in exponential smoothing, and then combines it

with state space models to compute forecast for the datasets. Finally it compares the two models and draws some conclusions about them.

The Box-Jenkins approach consists of a four-step iterative procedure [4]:

- Tentative identification: historic data is used to tentatively identify an appropriate Box-Jenkins Model
- Parameter estimation: historic data is used to estimate the parameters of the tentatively identified model
- Diagnostic checking: various diagnostics are used to check the adequacy of the tentatively identified model and if need be to suggest an improved model, which is then regarded as a new tentatively identified model
- Forecasting: once a final model is obtained, it is used to forecast future time series values

ARMA Models

ARMA models are combined by two parts: autoregressive and moving average. It is always denoted by ARMA (p,q), where p, q are non-negative integers that refer to the order of autoregressive and moving average parts of the model. We can build only autoregressive or moving average model by setting the other one to 0. The general equations for ARMA models are as follows (Z_t is white noise): X_t represents the present value and ϕ is parameter for autoregressive and θ is parameter for moving average.

$$\begin{aligned}
AR(p): &\quad X_t = \phi_1 X_{t-1} + \ldots + \phi_p X_{t-p} + Z_t \quad && \{Z_t\} \sim WN(0, \sigma^2) \\
MA(q): &\quad X_t = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q} && \{Z_t\} \sim WN(0, \sigma^2) \\
ARMA(p,q): &\quad X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t && \{Z_t\} \sim WN(0, \sigma^2) \\
&\quad + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}
\end{aligned}$$

Tentative Identification

In order to use Box-Jenkins models, we have to first determine whether the time series we wish to forecast is a stationary time series. If it is not, we must transform the time series into a series of stationary time series values. If the original time series values are non-stationary and non-seasonal, then using the first differencing transformation or the second differencing transformation will usually produce stationary

time series values. The first differencing transformation is dividing two adjacent values to generate a new value, and the second differencing is to repeat the first differencing based on its results. The results of applying differencing to one of the sample datasets "Data1" are seen in Figure 4. The original dataset is not stationary, and the first difference is also not stationary. The second and the third difference are stationary and we can use the second difference to perform forecasting.

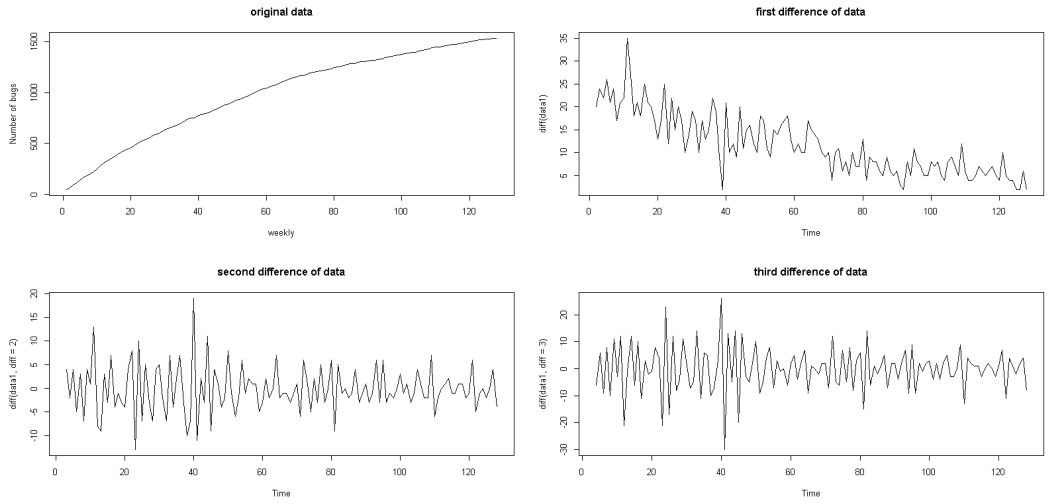


Figure 4: Difference of "Data1".

The ACF (Autocorrelation Function) measures the liner predictability of the series at present time based only on last time. It has a range from -1 to 1, of which 1 means positive relationship, 0 means little or no relationship and 1 mean negative relationship of two nearby time points. ACF can be used to help us find a working series of stationary time series values. If the ACF of time series values either cuts off fairly quickly or dies down fairly quickly, then the time series values should be considered stationary. If the ACF of time series values dies down extremely slowly, then the time series values should be considered non-stationary. In such case, data transformation is necessary. We then compute the ACF for the transformed data. The ACF of sample dataset- "data1" is shown in Figure 5.

PACF (Partial Autocorrelation Function) is used to measure the autocorrelation between one number and the k-th nu mber afterwards. The numbers between them are removed. In time series analysis, the PACF plays an important role in data

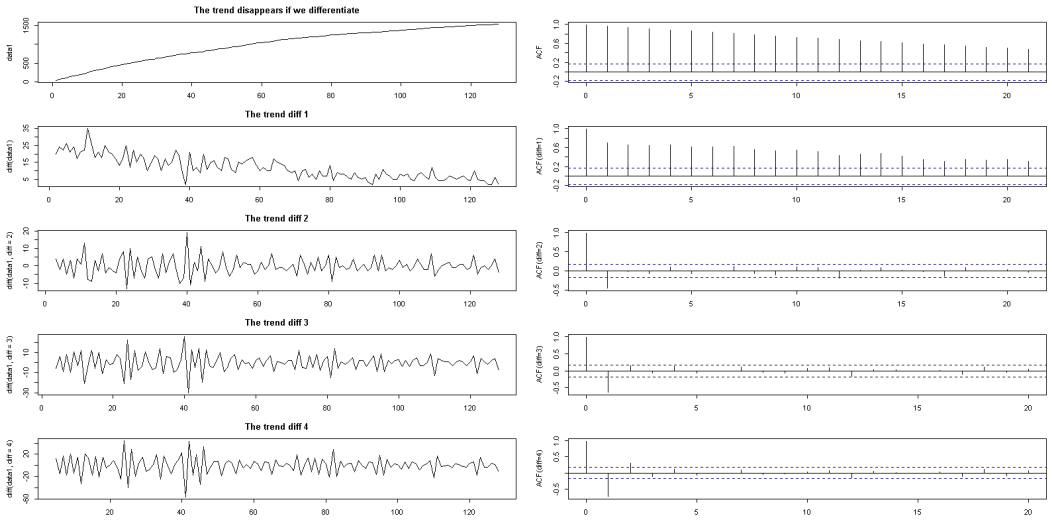


Figure 5: Difference and ACF of "Data1".

analyses aimed at identifying the extent of the lag in an autoregressive model. The use of this function was introduced as part of the Box-Jenkins approach to time series modeling, where by plotting the PACF one could determine the appropriate lags p in an AR(p) model or in an extended ARIMA(p,d,q) model.

We can use the ACF and PACF to determine how the ARMA model would be like. For the non-seasonal moving average model, the ACF cuts off after lag k, and PACF dies down in a fashion dominated by damped exponential decay. For the non-seasonal autoregressive model, the PACF cuts off after lag k, and ACF dies down in a damped exponential fashion. From Figure 6, we can conclude that the ACF cuts off after lag 1, and the PACF dies down, so the model for this dataset is MA(1), which is after two times of difference. The model for MA(1) is $Z_t = a_t - \theta_1 a_{t-1}, Z_t = y_t - 2y_{t-1} + y_{n+2}$, where y_t is the value observed at time t.

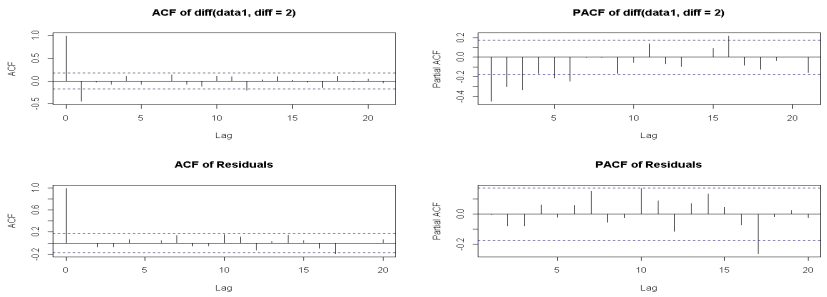


Figure 6: ACF and PACF of residuals of "Data1".

Parameters Estimation

After the model is identified, we need to estimate parameters in the ARMA models. we can use the least squares point estimate or the maximum likelihood estimation to calculate the value of θ and ϕ. Using the least squares point estimate to calculate value of θ in MA(1) model built for dataset-"Data1", we estimated θ to be 0.88578 as shown in Table 1.

Table 1: Conditional Least Squares Estimation of "Data1".

Conditional Least Squares Estimation					
Parameter	Estimate	Standard Error	t Value	Approx Pr¿—t—	Lag
MU	-0.14217	0.04011	-3.54	0.0006	0
MA1,1	0.88578	0.04151	21.34	¡.0001	1

Constant Estimate	-0.14217
Variance Estimate	14.07289
Std Error Estimate	3.751385
AIC	692.732
SBC	698.4045
Number of Residuals	126

Diagnostic Checking

After the parameters estimation, we have to check whether the model we build is accurate. We can use some values and test to check the model. The first one introduced in this thesis is the t value, which is calculated by the equation $t = \frac{\hat{\theta}}{S_{\hat{\theta}}}$, where θ denote any particular parameter in a Box-Jenkins model, and $\hat{\theta}$ denote the point estimate of θ, and $S_{\hat{\theta}}$ denote the standard error of the point estimate $\hat{\theta}$. If the absolute value of t is large, then we should include θ in our model. In our test, the t = 21.34 from Table 1, which is much larger than zero. As a result θ is included in our model.

Analysing residuals is a good way to check whether the model is adequate. we can check the ACF and PACF of the residuals. From Figure 7 it is evident that both ACF and PACF cut off after lag 1, so the model is accurate.

Another checking value is called the P value, which is used to test the hypothesis that the model is correct. Normally, if the value of P is greater than 0.05, the model is adequate. The P values of our model are all above 0.05 according to our calculations from Figure 8, so the model is adequate.

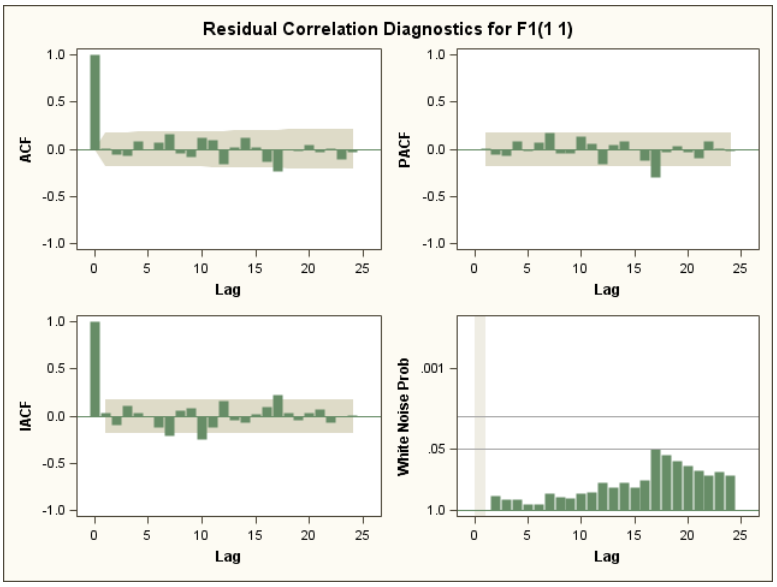


Figure 7: Residual Correlation Diagnostics for "Data1".

Forecasting

From the above analysis, the model we build is adequate. But in cases where the model is not adequate, we need to change equations or set up another model. After the checking is done, we can use this model to forecast values of future time points. Based on the model of MA(1), we use R to compute forecasts for our dataset "Data1". The modelling and forecasting is divided into four parts by the number of training and test data points. The training data used for building up the model are the first 50 weeks, the first 80 weeks, the first 100 weeks and the full dataset. With the exception of the full data set, remaining time series data for each trial run was used to test the accuracy of our model. The results are plotted in Figure 9.

ARIMA Models

The ARIMA models are developed from ARMA models, and have an added parameter named d, which represents the degree of difference. This model can deal with non-stationary time series by differencing dataset first. There are many software tools like SAS and R that can automatically build up an ARIMA model. The ARIMA model for the ARMA model build for "Data1" is ARIMA(0,2,1).

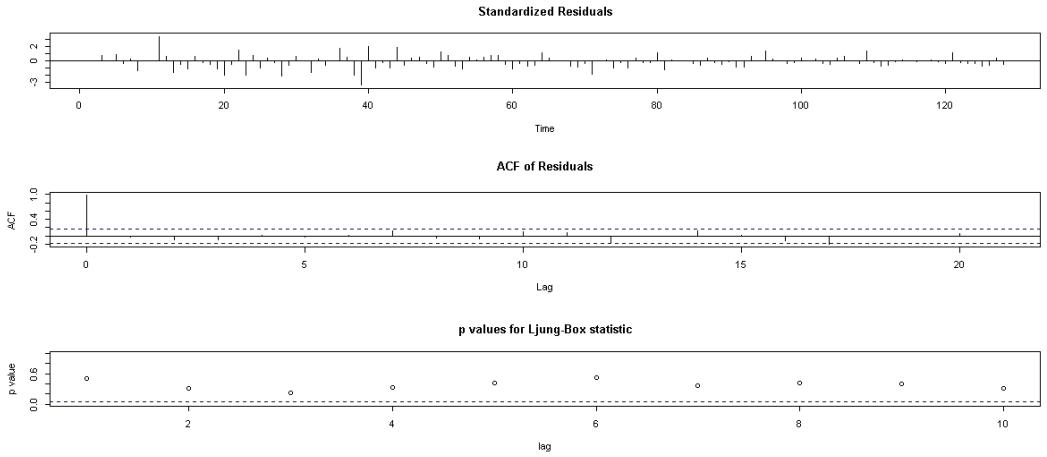


Figure 8: Diagnostic Checking of time series "Data1".

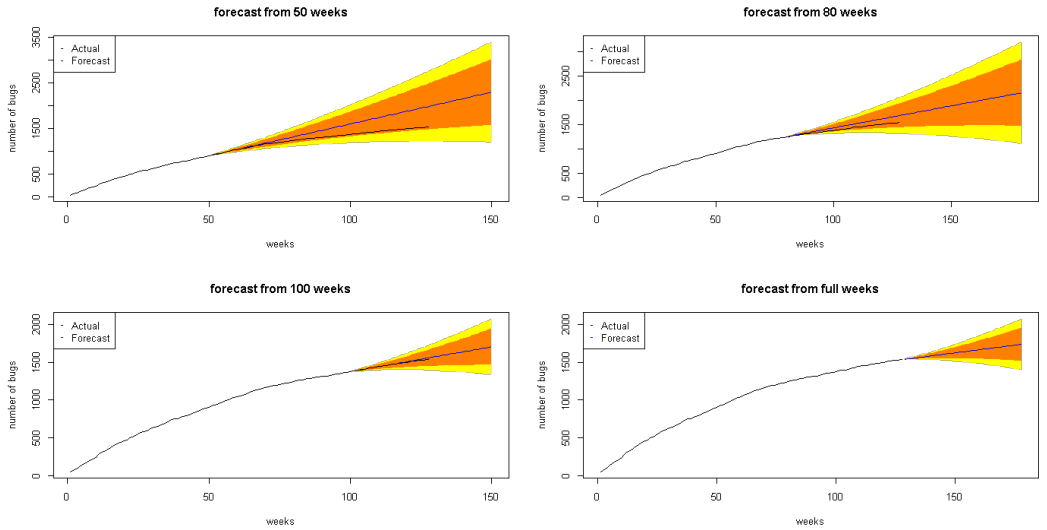


Figure 9: Forecasting results for "Data1".

2.3.3 Exponential Smoothing

Exponential smoothing is a very popular scheme to produce a smoothed time series and assigns exponentially decreasing weights as the observation get older. That is, more recent data points affect the forecast trend more heavily than older data points. This unequal weighting is accomplished by using one or more smoothing constants. It provides a forecasting method that is most effective when the components like trend and seasonal factors of the time series may changing over time. Historically, the exponential smoothing methods were intuitive methods not based on any for-

mal statistical models, but the state space models have now provided a statistical framework for the exponential smoothing methods.

The basic equation for predicting the next value of a given time series x_t at the period $t = T$ is to take weighted sums of past observations. It is as follows (α is the smoothing constant): $\hat{X}_{t=T}(1) = \alpha \cdot X_T + \alpha(1-\alpha) \cdot X_{T-1} + \alpha(1-\alpha)^2 \cdot X_{T-2} + \ldots$ This basic form should only be used when time series data has no systematic trend and seasonal components. The "Holt-Winters" developed from exponential smoothing is used to deal with time series containing trend and seasonal variation. There are three smoothing parameters required: α for the level, β for the trend, and γ for the seasonal variation.

In this subsection, we will look at Holt-Winters methods first, and then introduce the state space model with a single source of error for the exponential smoothing models. The exponential smoothing models underlying state space is notated as ETS model by Hyndman et. [5] The ETS stands for Error, Trend, and Seasonal, which is a summary of all exponential smoothing models underlying state space.

Holt-Winters Method

The Holt-Winters exponential smoothing is used when the data exhibits both trend and seasonality. The two main HW models are the Additive Model for time series exhibiting constant seasonality and the Multiplicative Model for time series exhibiting increasing seasonality. Because we can see from the overall curve of the "Data1" that the seasonality is constant, we only need to use the additive Holt-Winters method. The additive Holt-Winters prediction function (for time series with period length p) is as follows: $\hat{Y}$[t+h]=a[t]+h*b[t]+s[t-p+1+(h-1) mod p], where a[t], b[t] and s[t] are given by

$$
\begin{aligned}
a[t] &= \alpha(Y[t] - s[t-p]) + (1-\alpha)(a[t-1] + b[t-1]) \\
b[t] &= \beta(a[t] - a[t-1]) + (1-\beta)b[t-1] \\
s[t] &= \gamma(Y[t] - a[t]) + (1-\gamma)s[t-p]
\end{aligned}
$$

Using the formulas above, the Holt-Winters model was used to model time series data of "Data1". The prediction results are plotted in Figure 10. It can be seen

from the results that Holt-Winters method is not a very accurate prediction for this dataset. This is because the trend is additive and the curve will increase all the way. The accuracy of forecasting results of Holt-Winters model is list in Table 2:

Table 2: Accuracy analysis of Holt-winters Model for "Data1".

ME	RMSE	MAE	MPE	MAPE	MASE
334.284	621.403	335.495	33.322	33.491	28.615

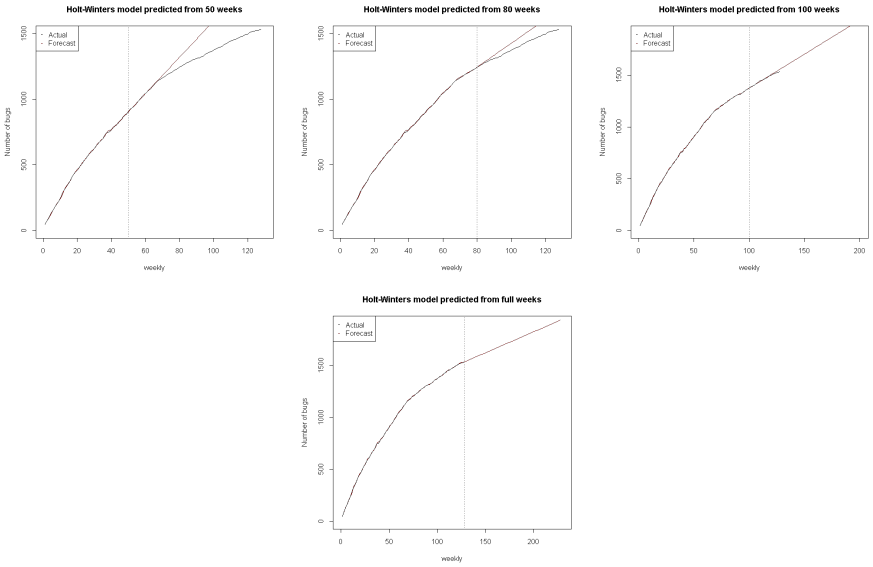


Figure 10: Holt-Winters Forecasting of "Data1".

ETS Models

Every exponential smoothing method has a corresponding state space model. An exponential smoothing method is an algorithm for producing pint forecasts only. The underlying stochastic state space model gives the same point forecasts, but also provides a framework for computing prediction intervals and other properties. In this subsection, we will illustrate how the state space models combine with exponential smoothing. The combination is called ETS model, and the following table lists all the combinations of trend and seasonality in this model. There error term has two values that are additive and multiplicative. The trend term has five values that are None, Additive, Additive Damped, Multiplicative, and Multiplicative Damped. The seasonal component has three values None, Additive, and Multiplicative.

The observed time series are denoted by y1, y2, $\dots y_n$. A forecast of y_{t+h} based on

Table 3: ETS models combination.

Trend Component	Seasonal Component		
	N (None)	A (Additive)	M (Multiplicative)
N (None)	N, N	N, A	N, M
A (Additive)	A, N	A, A	A, M
Ad (Additive damped)	Ad, N	Ad, A	Ad, M
M (multiplicative)	M, N	M, A	M, M
Md(Multiplicative damped	Md, N	Md, A	Md, M

all of the data up to time t is denoted by $y_{t+\hat{h}|t}$. To illustrate the method, we give the point forecasts and updating equations for method (A, A), the Holt-Winters additive method:

$$Level : L_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(L_{t-1} + b_{t-1})$$

$$Growth : B_t = \beta^*(L_t - L_{t-1}) + (1-\beta^*)b_{t-1}$$

$$Seasonal : S_t = \gamma(y_t - L_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$$

$$Forecast : \hat{y}_{t+h|h} = L_t + b_t h + s_{t-m+hm+}$$

Where m is the length of seasonality, and hm+ = [(h-1) mod m] + 1. The three smoothing parameters are α, β, and γ. For each combination of model, they have a set of corresponding formulas to model and forecast. In the dataset of "Data1", we just need to investigate on the ETS(A, Ad, N) from the analysis of AIC (Akaikes Information Criterion), see details in [6].

ETS(A, Ad, N) Model

The ETS(A, Ad, N) model has additive error, damped additive trend and no seasonality. It is the same as the damped Holt's method. The state space equation has an observation equation and one or more state equations. The observation equation and state equations for the ETS(A, Ad, N) model is as follows (where ϕ is the damping parameter).

State Equations: $L_t = L_{t-1} + \phi B_{t-1} + \alpha\varepsilon_t$; $B_t = \phi B_{t-1} + \alpha\gamma\varepsilon_t$

Observation Equations: $y_{t+\hat{h}|t} = L_t + \phi B_{t-1} + \varepsilon_t$

The algorithms and modelling frameworks for automatic time series forecasting are implemented in R's forecast package. Using this model to model and predict the dataset-"Data1", we obtain the results as shown in Figure 11. The R command to

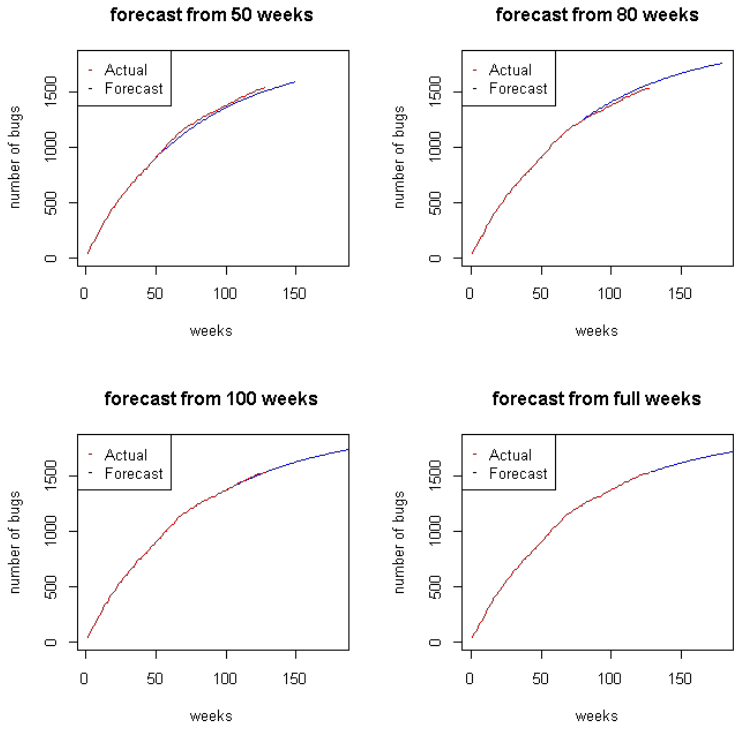


Figure 11: Forecasting results for "Data1".

compute the results for this model given below:

ETS (A,Ad,N) Call: forecast::ets(y = data1, model = "AAN", damped = TRUE, lambda = TRUE, opt.crit = c("mae"), nmse = 10, bounds = c("admissible"), restrict = TRUE)

Box-Cox transformation: lambda= 1

Smoothing parameters: alpha = 1.0037 beta = -0.0121 phi = 0.9869

Initial states: l = 19.8422 b = 24.4775 sigma: 3.5098

AIC AICc BIC 952.4796 952.9714 966.7397

From Figure 11, we can see that this model is more accurate than what we have before. This is because the properties of this dataset-"Data1" fit well of this method. It has additive trend and the trend is not increasing at the fixed amount, but having a decreasing growth. So the damped additive trend with additive error fits well for this dataset. This model is better than the ARIMA models for this dataset. So we use ETS models to predict the rest 15 datasets.

2.3.4 Model Comparison and Selection

From the results, we can see that for the dataset "Data1", the ARIMA model is good to some extent. For the Box and Jenkins methods, however, the dataset does not have the required combinations of trend and periods to make the method fit the data. On the other hand, exponential smoothing excels at accurately predicting the trend for dataset-"Data1", while the ARIMA models are not very suitable. This is easy to understand because the overall curve of this dataset has a damped additive trend and no periods. When we use an ETS model, we can set the trend to be damped additive and adjust the parameters to be adequate for prediction.

According to the above analysis, we have to choose methods for prediction based on the overall structure of our dataset. We have to understand the features of these models and choose the best one that fits. If the curve is simple and only combines with level (which is the first point of dataset) and trend, then we may use some simple methods like linear regression and simple exponential smoothing (Not discussed in this thesis, see [7]). If all these methods will not give good results, we may look at ARIMA models and some other exponential smoothing models. The exponential smoothing state space models are all non-stationary. If a stationary model is required, ARIMA models are better.

2.4 Clustering Algorithms

After the forecasting, we will get the reliability metrics for every software version. The reliability metrics are derived from forecasting results, which are three values: "Maximum number of bugs", "When will the maximum number of bugs be reached" and "How many bugs haven't been discovered yet". They are used as the attributes of every software version in the clustering. By analysing the metrics we get for every version, we find that we want to find software with low values of the three attributes. Because they are unlabelled data, so we need to use unsupervised methods to cluster them into different groups and recommend the group with the lowest combined value to use.

2.4.1 Reliability Metrics

For the prediction of the many software in every family, we focus on three values we want to get out of the prediction which are useful for evaluating our software. The three values we need to find in our prediction for every dataset, which are maximal value for predicted bugs; the number of weeks to reach this maximal value; and the difference between the maximal value and the number of bugs at present. The three values are important because the maximum number of bugs means that the total number of bugs can be found in the life cycle of this software. If the number is small, it means the software is rather stable and has fewer error. Another value is when will the maximum number of bugs be reached, it also represents the stage of this software at its life cycle, when it compared with its number of weeks at present. The third value is very straight forward to mean how many bug haven't been discovered yet.

We use Figure 12, which shows the time history of accumulated software bugs, to define three metrics for software reliability. With respect to Figure 12, we define the following variables of interest:

B(t) = cumulative number of detected bugs at time t;

T = present time;

B_T= B(T) = number of bugs that have been detected at present time, T;

B_M= max number of accumulated bugs;

T_M = time at which B(t) = B_M;

B' = estimate of the number of undetected bugs = $(B_M - B_T)$;

$T' = (T_M - T)$.

The three software reliability metrics that we use are max number of accumulated bugs, B_M; the time at which max number of bugs reaches, T_M; and the estimate of the number of undetected bugs, B'.

Clustering algorithms are used when we want to find similar data in a dataset and group them by their similarity. We want the points in a cluster to be as similar as possible to each other and as different as possible to other points in another cluster. In our project, we use the clustering algorithms to group the software

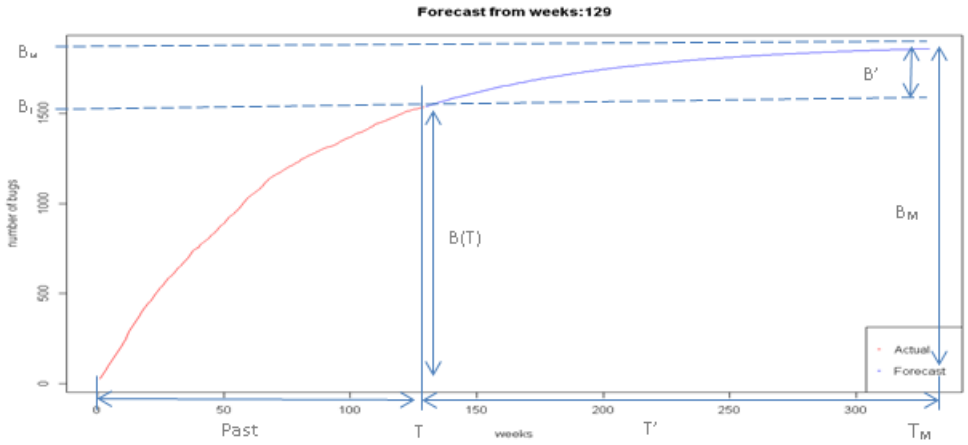


Figure 12: Forecasting metric for the software reliability.

versions into different clusters by the three features. The clustering algorithms we use are Expectation-Maximization(EM) algorithm and K-means. In this subsection, we will give some background of the two clustering algorithms.

2.4.2 EM Algorithm

EM algorithm is an unsupervised clustering method which using statistical method to estimate the parameters of the mixture model of the data. EM finds the clusters by determining a mixture of Gaussians that fit a given data set. Each Gaussian has an associated mean and convariance matrix. The approach of EM is probabilistic and it looks for the component density parameters that maximize the likelihood of the sample. The log likelihood given the sample $\chi = x^t{}_t$ is [9]

$$\mathcal{L}(\Phi|\chi) = log\Pi_t p(x^t|\Phi) = \sum_t \log \sum_{i=1}^{k} p(x^t|\mathcal{G}_i)p(\mathcal{G}_i)$$

where Φ includes the priors $p(\mathcal{G}_i)$ and also the sufficient statistics of the component densities $p(x^t|\mathcal{G}_i)$.

In order to solve for the parameters, the algorithm use iterative optimization of two steps. One is Expectation(E) step and another is Maximization(M) step. The goal of the algorithm is to find the parameter Φ that maximizes the likelihood of the observed values of χ. It needs to use an extra hidden variables Z and express the

underlying model using both, to maximize the likelihood of the joint distribution of X and Z, the complete likelihood $\mathcal{L}(\Phi|\chi, Z)$. The two steps are below, where l indexes iteration.

$$E - step : \mathcal{Q}(\Phi|\Phi^l) = E[\mathcal{L}(\Phi|\chi, Z)|\chi, \Phi^l]$$

$$M - step : \Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)$$

Iterate the two steps above, the E-step is used to estimate Z given χ and current Φ and the M-step is used to fine new Φ^l given Z, χ, and old Φ. In general lines, the algorithm's input are the data set (x), the total number od clusters(M), the accepted error to converge(e) and the maximum number of iterations. For each iteration, first is executed the E-step(Expectation), that estimates the probability of each point belongs to each cluster, followed by the M-step(Maximization), that re-estimate the parameter vector of the probability distribution for each class. The algorithm finishes when the distribution parameters converges or reach the maximum number of iterations.

2.4.3 K-means Algorithms

K means is used to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. It is a simple learning algorithm for clustering analysis. The first step is to find k reference vectors which best represent data, $m_j, j = 1, \cdots, k$. Then calculate the value of $\|x^t - m_i\| = \min_j \|x^t - m_j\|$ for every x by using nearest reference. The re-construction error for every iteration is defined as

$$E(m_{i\,i=1}^{\chi}) = \sum_t \sum_i b_i^t \|x^t - m_i\|$$

where $b_i^t = 1$ if $\|x^t - m_i\| = \min_j \|x^t - m_j\|$ and 0 otherwise.

The reference vector is set to the mean of all the instances that it represents. This is an iterative procedure because once we calculate the new m_i, b_i^t change and need to be recalculated, which in turn affect m_i. These two steps are repeated until m_i stabilize. The pseudo-code of the k means algorithm is given in 13.

Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to k random $\boldsymbol{x}^t$
Repeat
  For all $\boldsymbol{x}^t \in \mathcal{X}$
    $b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$
  For all $\boldsymbol{m}_i, i = 1, \ldots, k$
    $\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$
Until $\boldsymbol{m}_i$ converge

Figure 13: K-means algorithm. [9]

K-means algorithm is for clustering, that is for finding groups in the data, where the groups are represented by their centres, which are the typical representatives fo the groups. In this project, we used k means to separate our data into different groups.

2.4.4 Number of Clusters K

Choosing how many number of clusters to divide the dataset is a very important part of clustering methods. How to choose the number of K is a nagging problem and there is no agreed upon solution. Sometimes, people just declare it arbitrarily and sometimes the problem determines K. Even through there is no absolute answer for it, there are many ways to help us to get the optimal number for K.

In EM algorithm implemented in Weka, it used cross validation to select the number of clusters. The cross validation divided the data into n parts. Then for each part of the data, it use this part as the testing set and the other n-1 parts as the training sets. A clustering model is computed for each iteration and the value of goal function, like compute the distances from the points to the centroids, calculated for the test set. This process repeat for n times. These n values are calculated and averaged for each alternative number of clusters and the cluster number selected that minimizes the test set errors.

In K Means, however there is no implemented method to find the value of K. We can also use cross validation to compute the value of K before using K Means. There are other ways to get the optimal K value. There is one method named "Rule of Thumb", which sets the number of k to $k = \sqrt{n/2}$. The n is the number of data

points. Another way is to use the Information Criterion Approach, that is construct a likelihood function for the clustering model. Then based on the K means model, determine the information criterion values. Further information can be found in [12].

Besides those methods to determine the value of K, we can also consider the problem from the aspect of actual needs for the number of clusters in reality. In our problem, the number of K means the number of segments regarding to the software reliability. For example, if we want to segment our software into 4 segments: Very high reliability; High reliability; Median reliability, and low reliability. And for the four of them, we can highly recommend the first segment and recommend the second segment and fairly recommend the third segment and not recommend the last segment. In this case, we are not dividing our data based purely on the attributes of data, but divide our data based on the real needs of the problem.

3 Implementation

In this section, we will talk about how we applied the three steps approach to real problems in details. The implementation involves three major steps including defect data transformation, time series forecasting and optimization of software selection. Step 1 and 2 are the preparation steps for the later ones. From the last step, we will get our results and have the most reliable software selected from many relative software versions.

The software we want to measure are Internetwork Operating System(IOS) Software released from Cisco Systems. These IOSs are belonging to different family platforms and they are selected inside one family as IOSs inside one family tend to have similar features and functions.

3.1 Defect Data Transformation

The defect data for the IOSs we want to analyse are mainly the bug data recorded in the Oracle database of Cisco. The table we use is called "SD_BUG_DATA". There are many attributes in this bug data. The attributes of the table is shown in Table

4.

Table 4: Attributes of SD_BUG_DATA.

SWQ_BUG_KEY	BUG_ID
HEADLINE_TEXT	ALT_HEADLINE_ACTIVE_FLAG
ALT_HEADLINE_TEXT	ALT_RELEASE_NOTE_ACTIVE_FLAG
ALT_RELEASE_NOTE_TEXT	ALT_RELEASE_NOTE_TEXT
ALT_RELEASE_NOTE_TEXT3	KEYWORD_TEXT
LIFECYCLE_STATE_CODE	FEATURE
LIFECYCLE_STATE_NAME	PRODUCT_CODE
PROJECT_CODE	RELEASE_NOTE_STATUS_CODE
RELEASE_NOTE_TEXT	RELEASE_NOTE_TEXT2
RELEASE_NOTE_TEXT3	S1S2_WITHOUT_WORKAROUND_FLAG
BUG_LAST_MODIFIED_DATE	SECURITY_STATUS_NAME
SECURITY_STATUS_ID	SEVERITY_NAME
SOFTWARE_COMPONENT_NAME	SUBMITTER_USER_ID
VERSION_AFFECTED_NAMES1	VERSION_AFFECTED_NAMES2
VERSION_AFFECTED_NAMES3	VERSION_APPLY_TO_NAMES
VERSION_FIXED_IN_NAMES	VERSION_NAMES
VERSION_TO_BE_FIXED_NAMES	CREATE_DATE
UPDATE_DATE	RELEASE_NOTE_BLANK_CODE
REGRESSION_BUG_FLAG	SEVERITY_CODE
SWQ_SEVERITY_KEY	PRODUCT_RUNS_IOS_FLAG
PRODUCT_NAME	SWQ_SW_COMPONENT_KEY
SWQ_MDF_SERIES_KEY	SWQ_MDF_SOFTWARE_FAMILY_KEY
SWQ_MDF_PRODUCT_INTERFACE_KEY	SWQ_SUB_TECHNOLOGY_KEY
SWQ_TECHNOLOGY_KEY	TECHNOLOGY_NAME
SUB_TECHNOLOGY_NAME	SW_COMPONENT_NAME
SW_COMPONENT_DESCR	SWQ_LIFECYCLE_STATE_KEY
RESOLUTION_STATUS_CODE	

For this project, we mainly focus on the number of the bugs for each IOS, and the time associated with each bug record. First, we need to do feature extraction from this data. Selecting the attributes that are useful for this project and store them in a different table for further analysis. Then, after we create our own table for bug analysis, we pre-processing the raw data to clear the data and put them into desired format for next step.

3.1.1 Feature Extraction

From the table 4 , we can see there are many features for the bug data, of which there are some features that are useful for this project as we want to predict the number of bugs for every IOS. For our project, we only use five relevant features:

1. BUG_ID
2. VERSION_AFFECTED_NAMES1
3. VERSION_AFFECTED_NAMES2
4. VERSION_AFFECTED_NAMES3
5. CREATE_DATE

By creating a table from the five features, we can get the data we need. The BUG_ID stores the ID of every bug, and the VERSION_AFFECTED_NAMES1, VERSION_AFFECTED_NAMES2, VERSION_AFFECTED_NAMES3 store all the names of the IOS versions affected by this bug, and the create date is the time of occurring this bug. A sample data of this table is shown in table 5.

Table 5: Sample Data for the Selected Attributes of the SD_BUG_DATA.

BUG_ID	CSCsb77521	CSCsb80410
VERSION _AFFECTED _NAMES1	12.2(18r)S3 12.2(18r)SX1 12.2(18r)SX2 12.2(18r)SX3 12.2(18r)SX4 12.2(99)SX1003 12.2(99)SX1004 12.2(99)SX1005 12.2(99)SX1006 12.2(99)SX1007 12.2(99)SX1008 12.2(99)SX1009 12.2(99)SX1010 12.2(99)SX1011 12.2(99)SX1012 12.2(99)SX1013 12.2(99)SX1014 12.2(99)SX1015 12.2(99)SX1016 12.2(99)SX1017 12.2(99)SX1018 12.2(99)SX1019 12.2(99)SX1020 12.2(99)SX1021 12.2(99)SX1022 12.2(99)SX1023 12.2(18)IXA 12.2(18)SXD 12.2(18)SXD1 12.2(18)SXD2 12.2(18)SXD3 12.2(18)SXD4 12.2(18)SXD5 12.2(18)SXD6 12.2(18)SXD7 12.2(18)SXE ...	12.2(9)S 12.2(11)S 12.2(11)S1 12.2(11)S2 12.2(11)S3 12.2(14)S 12.2(14)S1 12.2(14)S2 12.2(14)S3 12.2(14)S4 12.2(14)S5 12.2(14)S6 12.2(14)S7 12.2(14)S8 12.2(14)S9 12.2(14)S9a 12.2(14)S10 12.2(14)S11 12.2(14)S11a 12.2(14)S12 12.2(14)S13 12.2(14)S13a 12.2(14)S14 12.2(14)S15 12.2(14)S16 12.2(14)S17 12.2(14)S18 12.2(14r)S1 12.2(14r)S2 12.2(14r)S3 12.2(14r)S4 12.2(14r)S5 12.2(14r)S6 12.2(14r)S7 12.2(14r)S8 12.2(14r)S9 12.2(17r)S1 12.2(17r)S2 12.2(17r)S4 12.2(17r)S5 12.2(17r)S6 12.2(18)S 12.2(18)S0a 12.2(18)S1 12.2(18)S2 12.2(18)S3 12.2(18)S4 12.2(18)S5 12.2(18)S6 12.2(18)S7 12.2(18)S8 12.2(18)S9 12.2(18)S10 12.2(18)S11 12.2(18)S12 12.2(18r)S3 12.2(20)S 12.2(20)S1 12.2(20)S2 12.2(20)S2a 12.2(20)S3 12.2(20)S4 12.2(20)S4a ...
VERSION _AFFECTED _NAMES2		
VERSION _AFFECTED _NAMES3		
CREATE _DATE	05/10/2006 18:21:59	05/10/2006 18:21:59

There are many IOS versions affected by the same bug, and we want to find for every IOS, its bug records and their created date. Based on the data we have, we need to do further process in order to get the data into right format for prediction. We need to separate the IOSs and for each of them, record its bugs and created date. This process needs lots of space, so in order to save space for processing, we

only create one table for storing BUG_ID and BUG_DATE. Then for every selected IOS, we store its BUG_ID and BUG_DATE by the reusing the same table.

3.1.2 Data Preprocessing

After the first step in feature extraction, we get three attributes we need to do the prediction of bugs number for every IOS. The three attributes are the bug id, the IOS affected by the bug, and the create date for this bug. Then we create another table to process the data, which means for every IOS, we get bugs that affected it and its date of creating. In order to use time series analysis to predict the bug number, we have to do more process for this data. We need to count the accumulated number of bugs for every IOS in the interval of week from the first date to the last date.

Table 6: Sample Data after the Preprocessing step 3 of IOS "12.1(19)EA1b".

BUG_DATE	BUG_COUNT
5/10/2006	16,808
5/24/2006	36
6/3/2006	16
6/8/2006	1
6/13/2006	2
6/15/2006	1
6/19/2006	1
6/20/2006	2
6/23/2006	1
6/28/2006	1
7/6/2006	2
7/12/2006	1
7/15/2006	1
7/18/2006	1
7/25/2006	2
8/2/2006	2
8/6/2006	2
8/8/2006	1
8/10/2006	1
8/22/2006	1
8/24/2006	1
...	...

There are some steps for us to get the final right data for prediction.

- Get the IOS name for which we want to do prediction;
- Store the data of BUG_ID and BUG_CREATE_DATE for this IOS by searching the data from table 5 using SQL;

- Count the number of bugs for the same day and store it in a table of two attributes: BUG_COUNT and BUG_DATE in a increasing order. Table 6 shows the sample results after this step.
- Group the accumulated number of bug counts into weeks and this is the data we used to do prediction.

Table 7: Sample Data for the Final Step of IOS "12.1(19)EA1b".

Number of Weeks	Accumulated Bugs Number
1	18
2	36
3	47.2
4	52.8
5	55.5
6	59.33333333
7	61
8	62.75
9	64
10	66.28571429
11	68.25
12	70
13	73.5
14	74.5
15	75.5
16	80
17	83
18	86
19	88
20	90
21	91.8
22	92.73333333
23	95
24	98
25	101
26	101.5384615
27	103
28	104.1428571
...	...

In the last step, we get the first date of the bug record for each IOS, and then count the total number of bugs happened in the first week and record the number, and in this way, we get the accumulated number of bugs into weeks until the last date of the bug record. After this process is done, we get the final datasets we used to do prediction. Because the bugs number has no seasonality, we choose to store it in weekly in order to do prediction.

We choose 48 IOSs to build up and test our model. The 48 IOS are chosen randomly from the whole IOSs datasets. Then we repeat the steps above to get the data into right format. The 16 of them are used as training data, and 32 of them are used as testing data in the following analysis.

3.2 Time Series Forecasting

We developed the appropriate model based on the training data in the theory section, now we need to find out apply the selected model to our datasets. This involves first find the requirements for the data that can be applied to the model and also get the accuracy by using testing data. In this, we need to find how many minimal data points we need in order to have a good forecasting for one dataset. Then based on the requirements and the model we have, we use testing data to go over the forecasting process and get the prediction accuracy. At last, we applied all the data in the 36 families to the model and get the results for evaluating the software.
Then in the end, the thesis will talk about using the forecasting results to build up the reliability metrics which will be used in the clustering.

3.2.1 Requirements for data

The requirement is how many minimal data points we need in order to have a good forecasting for one dataset. We try to use training data to forecast for 52 weeks ahead. For the 16 training data we have, for each of the 16 IOS release, forecasting for 26 and 51 weeks ahead based on 13, 17, 21, 26, 30, 34, 39, 43, 47, 51 weeks of data. Then, we analyse the forecasting results and its difference and error. Take the "Data1" for example, its accuracy analysis is showing in table 8.
After doing the same analysis for every dataset in training data, the minimal number of weeks of data need to do a good prediction can be summarised. If the prediction error is less than 5%, then we say the prediction is good. For the total 16 datasets we have the prediction results indicate that for more than 75% of data have good prediction after 21 weeks of data used, and more than 93% of data have good prediction after 26 weeks of data used. The information is summarised in table 9.

Table 8: Analysis of forecasting results for "Data1"

Weeks of actual data used for prediction	12.4(24)T			
	plus 51 weeks	plus 51 weeks Actual	Difference	Prediction error(%)
13	1078	1480.013332	402.0133321	37.29
17	1137	1305.870919	168.8709195	14.85
21	1170	1156.275403	13.7245968	01.17
26	1210	1207.845787	2.154213117	00.18
30	1247	1153.941799	93.05820132	07.46
34	1276	1276.196635	0.196634956	00.02
39	1307	1306.539397	0.460602972	00.04
43	1326	1286.350738	39.64926163	02.99
47	1357	1358.442376	1.442376223	00.11
51	1382	1378.840285	3.159714526	00.23

Table 9: Prediction results for all the training data

Table(X–prediction is not good; V–prediction is good)

IOS versions	# of weeks used to do prediction									
	13	17	21	26	30	34	39	43	47	51
12.4(24)T	X	X	V	V	V	V	V	V	V	V
15.0(1)M	V	V	V	V	V	V	V	V	V	V
15.1(1)T	X	X	V	V	V	V	V	V	V	V
15.1(2)T	X	V	V	V	V	V	V	V	V	V
15.1(3)T	X	X	X	X	V	V	V	V	V	V
15.1(4)M	X	V	V	V	V	V	V	V	V	V
12.2(18)SXF	V	V	V	V	V	V	V	V	V	V
12.2(33)SXH	X	V	X	V	V	V	V	V	V	V
12.2(33)SXI	X	V	V	V	V	V	V	V	V	V
12.2(33)SRC	V	V	V	V	V	X	V	V	V	V
12.2(33)SRD	V	V	V	V	V	V	V	V	V	V
12.2(33)SRE	V	V	X	V	V	V	V	V	V	V
12.2(33)XNE	X	X	V	V	V	V	V	V	V	V
12.2(33)XNF	X	X	X	V	V	V	V	V	V	V
15.0(1)S	X	V	V	V	V	V	V	V	V	V
15.1(1)S	V	V	V	V	V	V	V	V	V	V

We identified that between 21 and 26 weeks of data can be used to get a good prediction of 52 weeks ahead, then we narrow our search to find exactly number of weeks need to get more than 90% of good predictions. We use 22, 24 and 26 three number to retry the prediction and get the results in table 10. From table 10 we can see that given the 16 training datasets, predict for 52 weeks ahead, more than 90% of forecasting is good after 24 weeks; 100% of forecasting is good after 28 weeks. So the best starting point for forecasting is 24 weeks for these training data.

After we know that 24 is the minimal amount of data need to have a forecasting. We select our testing data and predict for these who satisfies the requirements. The 32 testing data all satisfies the requirement. We test its accuracy on all the testing data. We still use the difference of the predicted value minus the actual value and then divide the actual value as the percentage of error. If the percentage of error is less than 5%, we say the prediction is good. For this standard, 87.5 % of testing

Table 10: Prediction results of using 22, 24 and 26 weeks of data
Table(X–prediction is not good; V–prediction is good)

IOS versions	# of weeks used to do prediction		
	22	24	26
12.4(24)T	V	V	V
15.0(1)M	X	V	V
15.1(1)T	V	V	V
15.1(2)T	V	V	V
15.1(3)T	X	V	X
15.1(4)M	V	V	V
12.2(18)SXF	V	V	V
12.2(33)SXH	X	V	V
12.2(33)SXI	V	V	V
12.2(33)SRC	V	V	V
12.2(33)SRD	V	V	V
12.2(33)SRE	V	V	V
12.2(33)XNE	V	V	V
12.2(33)XNF	V	V	V
15.0(1)S	V	V	V
15.1(1)S	V	V	V

data have good predictions after using the first 24 weeks of data. This accuracy is a little bit lower than the accuracy on the training data, but it is acceptable for more than 85%. So the 24 weeks of data is identified as the starting point for predictions, and any dataset that contains data less than 24 weeks are not able to predicted accurately.

3.2.2 Forecasting based on Family

The requirement for prediction is that the minimal number of week needs to use for forecasting 52 weeks ahead is 24 weeks. It used to select the datasets from the data we need to forecast. We have 36 families that have many IOSs need to forecast. The first step is get the datasets for each IOS in every family. This step involves repeat the data pre-processing and then get the right format for prediction. After this, we can predict for every IOS in the 36 families. In our prediction process, we only predict the datasets that satisfies our requirement. Most of the datasets for IOS satisfies our requirement and only a few can't be predicted.

Then we predict for over 5,000 IOS software releases across 36 different product families, and get the reliability metrics for each prediction. The Table 11 shows the

partial results for prediction of the 2 families.

Table 11: Partial Prediction Results for Family: C3560 and C3750.

C3560			
IOS Name	MAX BUGS	# of weeks reaching Max	Difference (MAX-Present)
12.1(19)EA1c	272	295	0
12.2(25)SEE1	5015	368	19
12.2(25)SEE2	4979	415	32
12.2(25)SEE3	4644	382	23
12.2(35)SE5	5466	925	755
12.2(37)SE	5241	769	491
12.2(37)SE1	5197	766	501
12.2(40)SE	5033	785	574
12.2(44)SE2	4171	595	356
C3750			
IOS Name	MAX BUGS	# of weeks reaching Max	Difference (MAX-Present)
12.1(14)EA1a	282	295	0
12.1(19)EA1c	272	295	0
12.2(25)SEE1	5015	368	19
12.2(25)SEE2	4979	415	32
12.2(25)SEE3	4644	382	23
12.2(35)SE5	5466	925	755
12.2(37)SE	5241	769	491
12.2(37)SE1	5197	766	501
12.2(40)SE	5033	785	574

The number of bugs when one IOS release reaches its maturity is called the maximal value of bugs. We measure this maximal value of bugs is by measuring the number of bugs remain the same for over one year. When the bug number not changing for over one year, we mark this value as the maximal value of bugs for this IOS. The time for which the number reaches at first it the number of weeks we want to use as number of weeks for reaching the maturity. This maximize value and the number of weeks for reaching this maximization, and the difference between the maximization and the number of bugs at present are three major factors for building up the forecasting measurement metrics for evaluating IOS releases. When grouping many IOS releases inside one family, the performance of one IOS is compared with others inside one family based on their reliability metrics. The forecasting results are used to optimize the selection inside the family. Basically for the results of one IOS release, the less the three values the better the quality of this IOS. When putting the

three measure values together, we will put weight on them and using the weighted combined metrics to evaluate these IOS releases. [8]

3.3 Optimization of Software Selection

3.3.1 Optimize using Clustering

In this section, we talks about how we use the forecasting result-reliability metrics to optimize the selection of software versions. For these IOSs, they are divided by different families according to their features. There are total 36 families that have IOS for more than 20 and we need to optimize the selection of IOSs inside family by using the reliability metrics. In the previous section, we collect the name of IOS and do the prediction for every IOS for one family. In this section we cluster them into different groups based on the reliability metrics, and rank the groups according to their level of performance.

We have introduced two clustering methods K Means and EM in this project. We use Weka to do the clustering. For the 36 families, we select one family C3560 to illustrate how we use clustering to optimize our selection of IOS inside one family. The instances are IOSs in this family, and the attributes for every instance are the three metrics. The clustering algorithm is used to find the IOS with the lowest value of the three attributes. Because the range of the three attributes are different, we need first to normalize the three attributes into the range of 0 to 1. Then we perform the two algorithms to our datasets. For family C3560, there are totally 52 IOS, and so 52 instances. Import the data into weka and first normalize the value of three attributes. Then in later subsection, we will talk about the two clustering methods and their results.

3.3.2 Clustering Results of EM

Performing the EM algorithm in the dataset, we get four clusters based on the cross validation. In the later experiment of K Means, we also use four as our number of clusters. The number of clusters may change according to the real needs of the grouping. In our experiment, we use four as it is the optimal value for this family

based on the cross validation. If there are any other requirements from customers or software engineers for different ways of segment the software versions, other value may also be applied. For our four clusters, there are 12 IOS in cluster 1, and 7 IOS in cluster 2, and 11 IOS in cluster 3, and 22 IOS in cluster 4. The plot of the data points in different clusters is shown in Figure 14. The names of the IOS and clustering results is shown in Table 12.

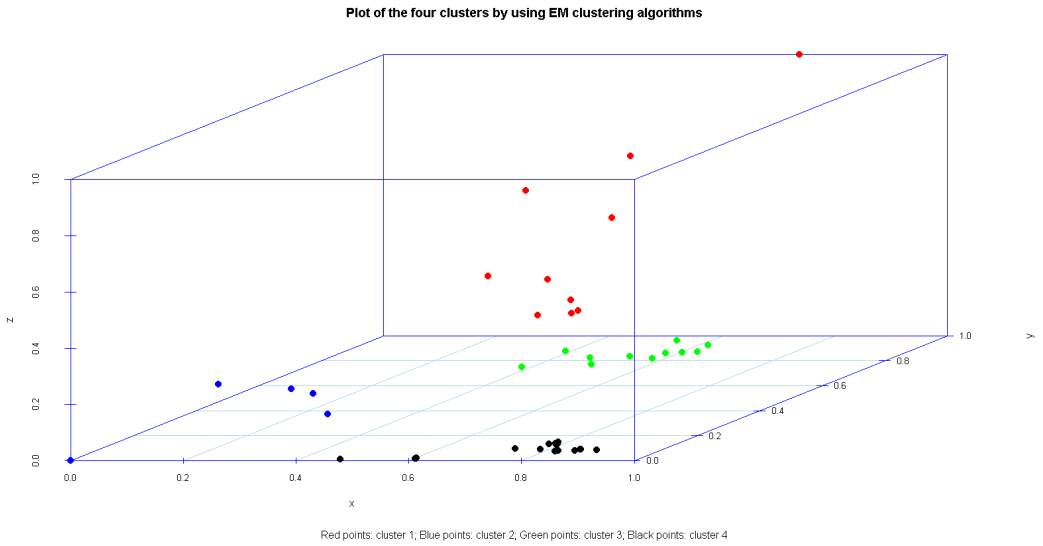


Figure 14: Clustering results of family C3560 by EM.

Table 12: Partial Clustering Results for IOS in family C3560.

Number	IOS	Attribute1	Attribute2	Attribute3	Cluster Name
37	12.2(44)SE3	0.717251	0.329256	0.387688	cluster1
38	12.2(44)SE5	0.709825	0.32314	0.379175	cluster1
39	12.2(44)SE6	0.695925	0.345566	0.416503	cluster1
40	12.2(46)SE	0.654227	0.313456	0.377865	cluster1
41	12.2(50)SE	0.597867	0.446993	0.445972	cluster1
42	12.2(50)SE1	0.621858	0.609582	0.59332	cluster1
43	12.2(50)SE2	0.621858	0.609582	0.59332	cluster1
45	12.2(50)SE4	0.428218	0.683996	0.656843	cluster1
47	12.2(52)SE	0.737624	1	1	cluster1
48	12.2(53)SE	0.489528	0.453109	0.454486	cluster1
49	12.2(53)SE1	0.489528	0.453109	0.454486	cluster1
50	12.2(53)SE2	0.569307	0.763507	0.743942	cluster1
1	12.1(19)EA1b	0.00019	0	0	cluster2
2	12.1(19)EA1c	0.00019	0	0	cluster2
3	12.1(19)EA1d	0	0	0	cluster2
44	12.2(50)SE3	0.343298	0.157492	0.167649	cluster2
46	12.2(50)SE5	0.395849	0.109072	0.117223	cluster2
51	12.2(55)SE	0.294935	0.173802	0.178127	cluster2
52	12.2(55)SE1	0.161081	0.181448	0.189915	cluster2

From Figure 14 and Table 12, we can see that the best IOS to select is in cluster 2, and then cluster 4, and then cluster 3 and cluster 1 is the last one to select from. Based on the values of the three metrics of IOS, we select the best ones into cluster 2, and select IOS from this group in order to have minimal number of bug both in

the present and in future. The selected IOS are more reliable comparing with other IOS in the same group.

3.3.3 Clustering Results of K-means

We also used K-means to do the clustering for our datasets. The results are almost the same as using EM algorithm except few points in the boundary. The cluster 1 has 8 instances, which corresponding cluster 1 in EM. The cluster 2 has 21 instance which corresponding cluster 4 in EM. The cluster 3 has 8 instances which corresponding cluster 2 in EM. The cluster 4 has 15 instances which corresponding cluster 3 in EM. The plot of the results is shown in Figure 15 and the Table 13 gives the partial IOS of clusters.

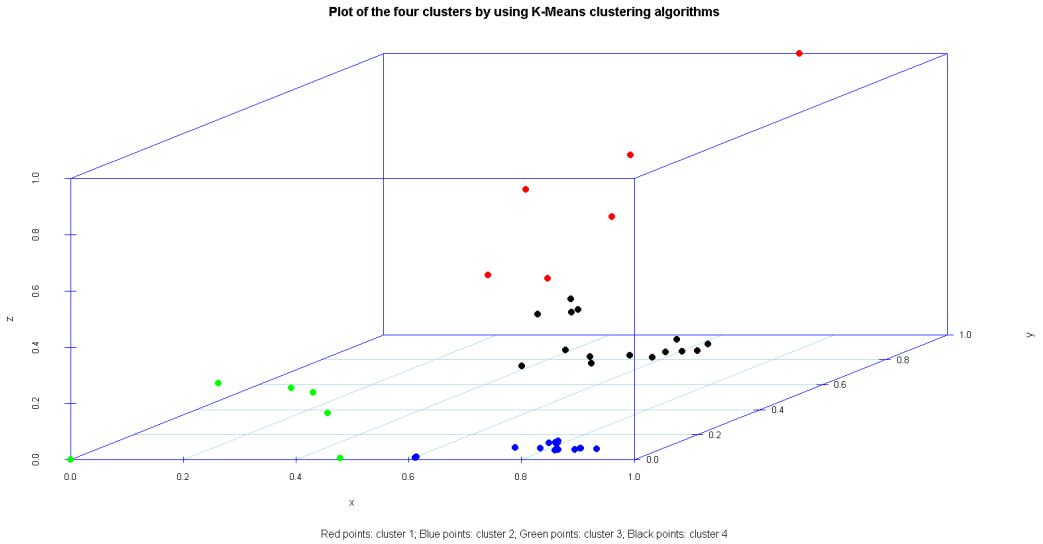


Figure 15: Clustering results of family C3560 by K-means.

Table 13: Partial K-means Clustering Results for IOS in family C3560.

Number	IOS	Attribute1	Attribute2	Attribute3	Cluster Name
41	12.2(50)SE	0.597867	0.446993	0.445972	cluster1
42	12.2(50)SE1	0.621858	0.609582	0.59332	cluster1
43	12.2(50)SE2	0.621858	0.609582	0.59332	cluster1
45	12.2(50)SE4	0.428218	0.683996	0.656843	cluster1
47	12.2(52)SE	0.737624	1	1	cluster1
48	12.2(53)SE	0.489528	0.453109	0.454486	cluster1
49	12.2(53)SE1	0.489528	0.453109	0.454486	cluster1
50	12.2(53)SE2	0.569307	0.763507	0.743942	cluster1
5	12.2(20)SE	0.614242	0	0.008513	cluster2
6	12.2(20)SE1	0.611196	0	0.006549	cluster2
7	12.2(20)SE3	0.612338	0	0.006549	cluster2
8	12.2(20)SE4	0.613861	0	0.008513	cluster2
9	12.2(25)SE	0.930693	0.005097	0.034709	cluster2
10	12.2(25)SEA	0.844821	0.007136	0.055665	cluster2
11	12.2(25)SEB	0.858149	0.008155	0.055665	cluster2
12	12.2(25)SEB1	0.856626	0.007645	0.057629	cluster2

From the plotting, and the clustering results by K-means, we find that the best group to select from is cluster 3. Then the second is cluster 2, and then third cluster 4, and the last one is cluster 1. When we choose the IOS from the same family, we can use this clustering results to help us choose the most reliable IOS when they all satisfy our requirements in other aspects.

4 Conclusions and Future Work

In order to choose the most suitable and reliable software release for a particular application, data analysis techniques that can process data into useful information is needed. This thesis proposes a novel approach that applies bug prediction and clustering method together to extract useful information from bug log files to optimize software selection. Different kinds of modelling techniques are explored at each stage of the project. First, time series analysis techniques are used to estimate software reliability by predicting the total number of bugs in a particular version. We found that using exponential smoothing yields better prediction results than Jenkins Box methods. Then in the application stage, we use our forecasting results to built up set of software reliability metrics from our data. These metrics can then be used as input to clustering methods which will group software versions based on rankings. We give priority to clusters of software that have a minimal number of total bugs (both found and not yet discovered). The most stable software versions are then put forward in our recommendations.

In this thesis we focused on reliability as a criterion for software selection. However, other aspects of software such as the feature set can equally be important in making decisions. For instance one version of the software could have been heavily optimized compared to a previous version and the customer needs that gain in spite of the additional bugs. Also the severity of bugs is not taken into consideration in our analysis. Typically we would pick a version with many minor bugs that have workarounds over one with a major bug without a workaround that can cause the entire system to fail. Further research work in this area is needed to incorporate all these factors make our recommendations more reliable and meaningful.

References

[1] GARETH, JANACEK, *PRACTICAL Time Series. University of East Anglia UK.* 2001.

[2] Rob J Hyndman, *Forecasting based on State Space Models for Exponential Smoothing.* Department of Econometrics and Business Statistics, Monash University, 2002.

[3] Richard J. Povinelli, *Time Series Data Mining: Identifying Temporal Patterns for Characterization and Prediction of Time Series Events.* Marquette University. 1999.

[4] Bruce L Bowerman, *Forecasting, Time Series and Regression: An Applied Approach.* 2005.

[5] Robert H. Shumway, David S Stoffer, *Time Series Analysis and Its Application with R Examples.* University of California, Davis, CA. 2006.

[6] Everette S. Gardner, Jr, *Exponential Smoothing: The State of the Art.*University of Houston, Houston, Texas. 2005.

[7] Hyndman, Koehler, Ord and Snyder, *Forecasting with exponential smoothing: the state space approach.* Springer-Verlag: Berlin. 2008.

[8] Leon Wu, Boyi Xie, Gail Kaiser, Rebecca Rassonneau, *BUGMINER: Software Reliability Analysis Via Data Mining of Bug Reports.*

[9] Ethem Alpaydin, *Introduction to Machine Learning; second edition.* The MIT Press, Cambridge, Massachusetts, London, England.

[10] Musa, John, Anthony Iannino, and Kazuhira Okumoto, *software Reliability.* McGraw-Hill, 1987.

[11] Jan G. De Gooijer, Rob J. Hyndman, *25 years of time series forecasting*, Forecasting 22, 2006.

[12] T. Hastie, R. Tibshirani, J.H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Second Edition, Springer.