# Chapter 1

# Introduction

## 1.1 Introduction

Two trends have become very popular[109] in software development community in current scenario: Agile Methodology [46] and Open Source Development [217]. Both these methods are efficient enough to improve software quality and responsiveness to changing customer requirements. Both of the methods involve frequent "releases" in short development time to handle the rapidly changing customer requirements. Both of these methodologies require rigorous testing and maintenance for improving the quality of software. The goal is to have a cost effective, efficient, robust and reliable software to be delivered which can fulfill desired requirements and achieve customer satisfaction.

Software Reliability[149] is one of the most challenging requirements determining the success of a software application. It can be defined as the capability of software for maintaining its desired performance under specific conditions for a specified period. Effective software maintenance and testing are needed to have a reliable software. Software maintenance is associated with a set of attributes that are related to the effort needed for making specific modifications in the software system. Software testing is associated with the effort needed for validating a software system. Due to rapidly changing customer requirements software maintenance and software testing consume a sizable effort in software development cycle.

A large number of defect prediction models have been proposed in the literature [62, 123, 112] to reduce the maintenance and testing effort. Software fails due to the existence of a large number of defects in the software application. The defects in the software are also called bugs, and they lead a software component to fail to perform as per desired requirements. There exist a large number of statistical and probabilistic model to predict software reliability from failure counts [67]. However, all the defect prediction models and failure count models described in the literature require access to underlying factors and internal characteristics of the software application. This is always not possible due to lack of access to internal source code of the particular application and also due to lack of cost effective tools and efficient techniques for extraction of internal parameters for modeling. There exists an alternative approach for modelling these defects (bugs) and failures which is not only cost effective but also does not require access to an internal characteristic of the software application. These class of models do not attempt to measure any internal parameters but instead use temporal patterns of a particular object (Example: Number of Bugs) across different versions of software applications to be modelled to identify the trend and forecast the value or pattern of the particular object. These class of models is called Time Series Models.

**The objective of this thesis is to identify the temporal patterns that exist in various forms like Bugs, Clones, Failure Intervals, etc. across different versions of a software application.** The thesis also used advanced time series analysis for modelling these increasing and decreasing patterns of a particular software characteristic (Bugs, Clones Failure Intervals) and also to give effective feedback to software developers and testing team in advance. This will reduce the effort invested in testing and maintenance. This will also help software managers to decide on resource allocation and effort investments.

Predicting failure intervals in advance will also improve the software reliability if timely corrective measures are taken by the developers.

## 1.2 Time Series Analysis

A time series can be defined as an ordered sequence of data values of a variable at equally spaced time intervals[35, 74]. A time series model is applied to understand the underlying forces and structures which produced the observed data. It also helps in modeling data for forecasting, monitoring or even feedback and feed forward control. Time series prediction refers to the process of prediction of future measure by analyzing the trend of past and current ones [197]. Figure 1.1 presents an example of time series.

### 1.2.1 Univariate Time Series

Let an observed discrete univariate time series be $\{S_1, S_2, ....S_T\}$ observed over equally spaced time points.

A fairly general model for the time series can be written as

$$S_t = g(t) + \varepsilon_t$$

Systematic part: $g(t)$, also called signal or trend, which is a deterministic function of time. Stochastic sequence: $\varepsilon_t$, a residual term also called noise, which follows some unknown probabilistic law.

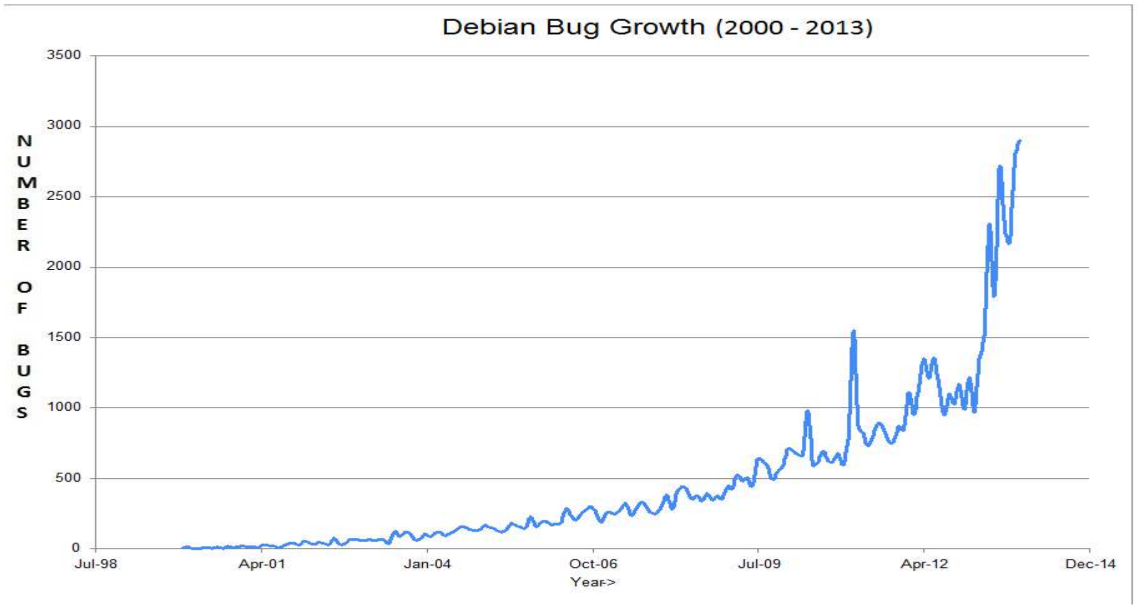

FIGURE 1.1: Example of Time Series

### 1.2.2 Modelling Time Series

The methods used to model a time series include Box-Jenkins ARIMA models, Box-Jenkins Multivariate Models, and Holt-Winters Exponential Smoothing (single, double, triple)[197]. In this section, we discuss the Autoregressive Model(AR Model), Moving Average Model(MA Model) and Autoregressive Integrated Moving Average processes (ARIMA) model in detail.

#### 1.2.2.1 Autoregressive Model

An AR model[197] is one in which $Y_t$ depends on $Y_{t-1}, Y_{t-2}, Y_{t-3}, ......\varepsilon_t$ etc.

$$Y_t = f(Y_{t-1}, Y_{t-2}, Y_{t-3}, ......\varepsilon_t)$$

A common representation of Autoregressive Model where it depends upon past p Values is $AR(p)$:

$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3} + ....\beta_p Y_{t-p} + \varepsilon_t$

$\beta_p$ are the parameters of the autoregressive part of the model. $\beta_p \neq 0$.

#### 1.2.2.2 Moving Average Model

A moving average model[197] is one in which $Y_t$ depends only on random error terms.

$Y_t = f(\varepsilon_{t-1}, \varepsilon_{t-2}, \varepsilon_{t-3}......)$

A common representation of Moving Average Model where it depends upon past $q$ Values is called $MA(q)$. The error terms $\varepsilon_t$ are assumed to be white noise process with zero mean and constant variance.

$Y_t = \beta_0 + \varepsilon_t + \Phi_1 \varepsilon_{t-1} + \Phi_2 \varepsilon_{t-2} + \Phi_3 \varepsilon_{t-3} ... \Phi_q \varepsilon_{t-q}$

$\Phi_q$ are the parameters of the moving average part. $\Phi_q \neq 0$

#### 1.2.2.3 ARMA Model

In ARMA model[197] time series is represented as a mix of both AR(p) and MA(q) Model. This model is referred as ARMA(p,q). The time series depends on p of its past own values and q past white noise disturbances.

$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3} + ....\beta_p Y_{t-p} + \varepsilon_t + \Phi_1 \varepsilon_{t-1} + \Phi_2 \varepsilon_{t-2} + \Phi_3 \varepsilon_{t-3} ... \Phi_q \varepsilon_{t-q}$

$\beta_p$ are the parameters of the autoregressive part of the model.

$\Phi_q$ are the parameters of the moving average part.

$\beta_p \neq 0$.

$\Phi_q \neq 0$

### 1.2.3 Stationary Time Series

Stationary Process is a stochastic process[82] whose joint probability distribution does not change when shifted in time. $P(Y_t) = P(Y_{t+k})$
$P(Y_t, Y_{t+k})$ does not depend on $t$.

A time series $Y_t(t = 1, 2, 3...N)$ is said to be stationary if it has its all statistical properties, such as mean, variance, and autocorrelation constant over time.

The stationarity of time series can be checked using ADF (Augmented Dickey-Fuller Test)[223]. Autocorrelation Function Plot (ACF)[174] can also be used to check the stationary behavior of time series.

#### 1.2.3.1 Augmented Dickey-Fuller Test

The ADF[223] test is used to check the presence of a unit root which leads to violation of assumption of classical linear regression in autoregressive time series models. Presence of a unit root indicates the non stationary behavior of a time series. The standard Dickey Fuller test estimates the equation as given below:
$\Delta y_t = Y_{t-1} + \varepsilon_t$
The Dickey Fuller test is only valid for AR(1) processes. If the time series is correlated at higher lags, the augmented Dickey Fuller test performs a parameter correction for higher order correlation, by adding lag differences of the time series. In this paper The ADF test is performed in the [web:reg] [4] (an Add-In to Excel written by Kurt Annen.). The unit-root existence is checked in the following formulations:

1. With Constant or intercept $\Delta y_t = \alpha + \gamma Y_{t-1} + \varepsilon_t$

2. With constant+trend $\Delta y_t = \alpha + \gamma Y_{t-1} + \beta \times t + \varepsilon_t$

The ADF test is basically $H_0 : \gamma = 0$
If the test statistic is less (Due to non-symmetrical nature of the test absolute value is not considered) than (a larger negative) the critical value, then the null hypothesis of $H_0 : \gamma = 0$ is rejected and it indicates absence of unit roots.

#### 1.2.3.2 Autocorrelation Function Plots.

The sample autocorrelation function (ACF)[211] for a series gives correlations between the series $x_t$ and lagged values of the series for lags of $1, 2, 3, ....$ and so on. The lagged values can be written as $Y_{t-1}, Y_{t-2}, Y_{t-3}, ...$ and so on. The ACF gives correlations between $Y_{t-1}$ , $Y_{t-2}$, $Y_{t-3}$, and so on.

$\rho_k = Corr(Y_t, Y_{t-p}) = \frac{Cov(Y_t, Y_{t-p})}{(\sqrt{var(Y_t)}\sqrt{var(Y_{t-p})})}$

For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Figure 1.2 presents the ACF plots for the non-Stationary and stationary time series respectively.

### 1.2.4 ARIMA Model[197]

1. ARIMA (p, d, q) consists of AR (p), MA(q) and ARMA (p, q) classes.

2. AR (p) is a autoregressive model of order P which represents the value of a series as a linear regression of previous P values.

3. The moving average MA (q) take care of the white noise terms. ARMA (p, q) is the combination of AR (p) and MA (q) model.

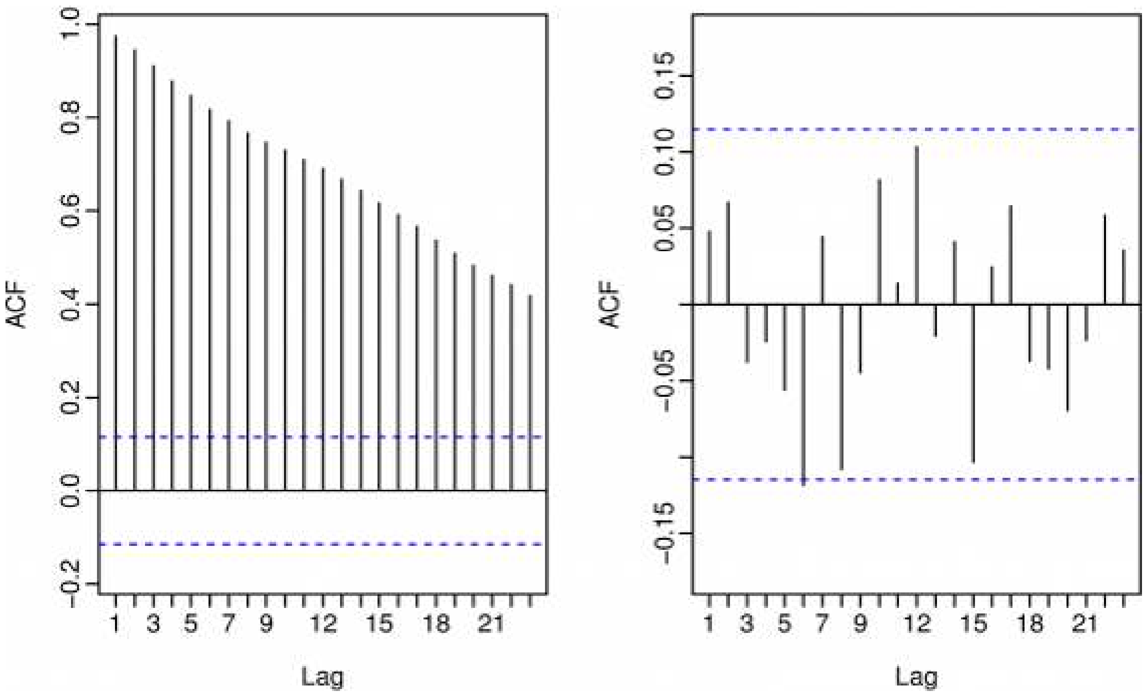

FIGURE 1.2: First One: Non-Stationary, Second One: Stationary

4. For a Non-Stationary time Series, the time series needs to be differentiated before applying ARMA (p, q) model. ARIMA includes the differentiating operator d.

#### 1.2.4.1 Calculating Model Parameter for ARIMA

1. Partial Auto Correlation are used to measure the degree of association between $Y_t$ and $Y_{t-p}$ when the effects of lags $1, 2, 3, (p-1)$ removed.

2. The theoretical ACFs and PACFs are available for various lags of auto regressive and moving average components. i.e. p and q.

3. Therefore, a comparison of Correlograms(Plot of ACFs versus lags) of the time series data with the theoretical ACFs and PACFs leads to selection of the Appropriate ARIMA(p,q).

ARIMA Model is implemented in three basis steps:

TABLE 1.1: Model Parameter for ARIMA

| Model | ACF | PACF |
|---|---|---|
| AR(p) | Spikes decay towards zero. | Spikes cutoff to zero |
| MA(q) | Spikes cutoff to zero | Spikes decay towards zero. |
| ARMA(p,q) | Spikes decay towards zero. | Spikes decay towards zero. |

1. Model Identification: In the Model identification phase the d value has to be set. It decides the stationary $(d = 0)$ or non-stationary $(d > 0)$ behavior of a time series. ACF and PACF plots are plotted to find out the parameters. The identification of $(p,q)$ is based on Akaike Information Criterion (AIC). The model with smallest AIC is chosen [77]

2. Estimation: In this phase, the coefficient $\beta_p$ and $\Phi_q$ are estimated [227].

3. Diagnostic Checking: The diagnostic phase deals with model adequacy by plotting the residuals. The model with the smallest residual is chosen [227].

## 1.3 Motivation

Fundamental analysis attempts to determine the value of an object by focusing on underlying factors that affect the movement of the object. Technical analysis (Time Series Analysis), on the other hand, looks only at the increasing and decreasing movement of a particular object and uses this data to predict its future movements. The advantages of time series modeling are that it does not consider the internal characteristic of an object into account. It only considers the temporal ordering into account. Time Series Modelling is frequently used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather forecasting, intelligent transport and trajectory forecasting, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering, and largely in any domain of applied science and engineering which involves temporal measurements[250].

In the field of Software Engineering, there also exist many complex processes like software defect prediction, software reliability production and also software clone evolution prediction. The internal mechanism of all these modelling techniques is a complex phenomenon as it requires access to underlying factors and internal characteristics . There are a number of fundamental techniques used for prediction of above mention attributes. However, a few research [227, 76] have been conducted on analyzing temporal patterns of above attributes across different versions of a software application. Below we provide a brief description of motivation behind our research work.

**Temporal Bug Pattern Prediction:** The likelihood of bugs in any software application depends upon many invariant parameters like code complexity, problem domain, amount code change and also on internal software process adopted [249]. A bug or defect may be caused during different stages of development of software like coding, design or testing phase. The causal modelling of the bug growth patterns in any software is a complex and tedious task as it inquiries many internal details of the software which sometimes is also impossible. The event of reporting a bug, fixing a bug and a new developer assigned to a project are all uncertain [76]. However in aggregate, all these random like interactions shows some rules and patterns, which is not purely random. Effective time series modelling approach is required to model these bug growth patterns.

**Clone Evolution Prediction:** As modern software is developed using Agile or Open Source Software Development, they are prone to frequent changes as per customer requirements. During software improvements, there is always a chance of code fragment being copied or changed slightly in the subsequent version. These exact or slightly modified code fragments are called clones. If we can detect these recurring code fragments and model them, it can be immensely helpful in software maintenance activities. Effective time series analysis is required to model these evolving cloned components.

**Software Reliability Prediction:** Modeling the software reliability is a challenging issue as it depends on many internal parameters like size of software, failure count, and also total time. The reliability prediction based on the internal parameters is always not accurate due to the complex relationship between the parameters. There are also many statistical models used for prediction of software reliability, but they are not so accurate. There exist a good deal of software reliability growth models (SRGMs), but they also suffer from unrealistic assumptions and dependency on the particular environment. An effective solution is to apply used a time series approach for software reliability prediction. Software reliability is usually measured regarding Time between Failure (TBF) [148].

## 1.4 Clones, Bugs and Failures

In this section, we provide a brief description of inter-relationship between the three software characteristics used for modelling. As discussed in the previous section, clones are duplicated passage of source code. Although reuse is a well-documented and controlled process, clones are undocumented and uncontrolled. Cloning enhances the probability of increasing number of bugs [216] in the software application due to the faulty code fragment being repeatedly used in different places. Many research papers have reported that cloning increases the probability of bug propagation[199]. The probability of a software component towards failure increases due to the increased number of bugs. So, there exists a direct relationship between clones, bugs, and failures, i.e. a potential increase in cloned fragments will lead to increase in a number of bugs in a software application, and this also lead to increased numbers of failures being reported. An increased number of failures will lead to a reduction in the Time between Failures (TBF) and make the software unreliable. Therefore, modeling these growing temporal patterns is a challenging issue in the field of software engineering. Figure 1.3 presents Clone-Bug-Failure relationship,

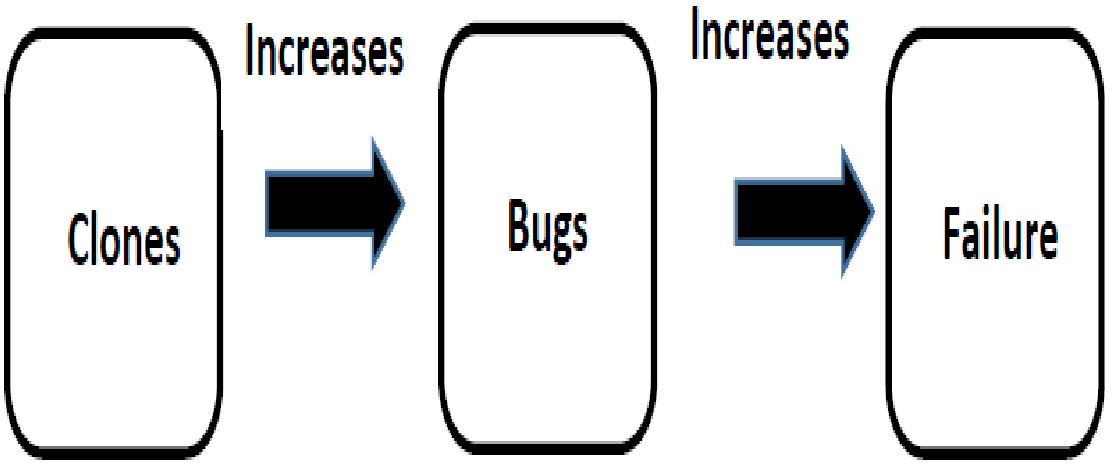

FIGURE 1.3: Clone-Bug-Failure Relationship

## 1.5 Thesis Objectives

Let an observed temporal sequence be $\{O_1, O_2, O_3, ....O_T\}$ observed over equally spaced time intervals.
A fairly general model for the time series can be written:

$O_t = f(t) + \varepsilon_t$ Here, we have two components: Systematic Part: $f(t)$ = The Component to be Modelled using Time Series.
Stochastic Part: $\varepsilon_t$, a residual term also called noise, which follows some unknown probabilistic law.

The objective is to predict $O_t$ from $p$ past observations.

$O_t = f(O_1, O_2, O_3, ....O_{t-p}) + \varepsilon_t$

The objective is also to have a suitable model which can predict the temporal patterns accurately with minimum value of error ($\varepsilon_t$).

## 1.6 Thesis Contribution

1. In the Thesis, Time Series Analysis Approach is used to improve Software Bug Prediction, Clone Evolution Prediction and Software Reliability Prediction.
2. Advanced Time Series Modeling using Statistical and Machine Learning Approaches have been used for modelling the temporal patterns.
3. The dataset used are obtained from Open Source Software Repository and also Closed Source Software Repository.
4. The results are evaluated using Standardized Evaluation Techniques and compared against other reported methods.

The Detailed Contribution is briefly described in this section.

### 1.6.1 Temporal Bug Pattern Prediction

The monthly, weekly and daily bug number information is extracted from the repositories. These extracted bug number data can be used for time series analysis.

Advanced time series techniques are used for prediction of the increases and decreases in bug numbers of a software system. The bug pattern across different versions of a software system is also predicted using Markov Model and Hidden Markov Model. A systematic comparison of result with traditional models is also presented.

### 1.6.2 Clone Evolution Prediction

In this work, the objective is on the identification of the different type of clone components and prediction of cloned components in another version of the software application. In the first part of our work, the cloned components are identified using various clone detection techniques[27]. In The second part, the evolution of the cloned components is predicted using advanced time series modeling. A large number of papers [27] have identified clones based on the same version of software systems. The primary work of this paper is to model the clones evolution across different versions of the software systems.

The analysis is done on 31 versions of ArgoUML ranging from version 0.9.5 to the latest stable release 0.34.0. This span a duration of over 6 years from 2006 to 2011. For the subsequent versions of the software, the clone amount was successfully predicted using the time series modeling techniques.

### 1.6.3 Software Reliability Prediction Using TBF

Software reliability is usually measured regarding Time between Failure (TBF). In this thesis, time series approach is used for software reliability prediction [148]. In the first phase, the Time between Failure (TBF) series for three different types of the software applications is collected . The next step is temporal modelling of TBF series for accurate prediction of software reliability. The models are validated on three different types of software systems: 1: Real Time System 2: Military System 3: Word processing System The failure interval data for these systems is available on CSIAC Software Reliability Dataset given by J Musa [147].

## 1.7 Thesis Organization

In this thesis, the temporal pattern across different versions of the various software application is modelled using time series analysis. The objective is to have cost effective and parameter independent models for analyzing important software characteristics like

Bugs, Clones, and Failures. This will help in reduction of maintenance and testing effort. Advance knowledge about failure intervals will be helpful in building a more reliable software. The Organization of the thesis is as follows.

Chapter 2 presents a comprehensive survey of various models on defect prediction, clone prediction and software reliability prediction. The chapter contains a description of different tools and techniques used for modelling above software characteristics. It also presents a brief description of the result and also reports the remarks and suggestion by the authors.

Chapter 3 introduces a time series approach for modelling the increases and decreases in bug numbers. The bug information about different software applications is collected from bug repositories [15]. A number of advanced techniques like Artificial Neural Network, Hybrid Model (ARIMA +ANN), Ensemble Model are applied for modelling the temporal bug number series. Finally, Markov Model and Hidden Markov Model(HMM) is also applied for predicting the temporal patterns across different versions of software applications.

Chapter 4 introduces a two phase modelling approach for predicting clone evolution. In the first phase, the cloned components have been identified using a systematic clone detection process. In the second phase, advanced time series modelling approaches like Hybrid Model (ARIMA +ANN) are used for predicting the evolution of cloned components across different versions of software applications. In another approach software metrics information is also used along with lag values of time series for improving the accuracy of the model. This chapter also introduces a prediction interval based approach using MOGA-NN for predicting clone evolution with more certainty. The results are compared with other reported methods.

Chapter 5 introduces a time series approach for modelling software reliability using Time

Between Failure(TBF) patterns. In the first phase, TBF series is collected for three different software systems from CSIAC Software Reliability Dataset. In the next phase, a point estimation approach using Hybrid Model (ARIMA +ANN) is applied to predict software reliability from TBF series. In the next section, an interval based estimation approach using MOGA-NN and ELM +KNN is also applied to predict software reliability with increased certainty and confidence. The chapter also presents a systematic comparison of results with different modelling techniques.

Chapter 6 concludes the dissertation and give direction for future enhancements.

The list of publications is presented in Appendix A

The Dataset used in this thesis is presented in Appendix B.