Reliability in open source software

POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Informatica e dei Sistemi – XXVI ciclo

Tesi di Dottorato

Reliability in Open Source Software

Najeeb Ullah

Tutore
Prof. Dr. Maurizio Morisio

28 February 2013

Table of Contents

List of Tables

List of Figures

Summary

Open Source Software is a component or an application whose source code is freely accessible and changeable by the users, subject to constraints expressed in a number of licensing modes. It implies a global alliance for developing quality software with quick bug fixing along with quick evolution of the software features.

In the recent year tendency toward adoption of OSS in industrial projects has swiftly increased. Many commercial products use OSS in various fields such as embedded systems, web management systems, and mobile software's. In addition to these, many OSSs are modified and adopted in software products. According to Netcarf survey more than 58% web servers are using an open source web server, Apache. The swift increase in the taking on of the open source technology is due to its availability, and affordability. Recent empirical research published by Forrester highlighted that although many European software companies have a clear OSS adoption strategy; there are fears and questions about the adoption. All these fears and concerns can be traced back to the quality and reliability of OSS.

Reliability is one of the more important characteristics of software quality when considered for commercial use. It is defined as the probability of failure free operation of software for a specified period of time in a specified environment (IEEE Std. 1633-2008). While open source projects routinely provide information about community activity, number of developers and the number of users or downloads, this is not enough to convey information about reliability.

Software reliability growth models (SRGM) are frequently used in the literature for the characterization of reliability in industrial software. These models assume that reliability grows after a defect has been detected and fixed. SRGM is a prominent class of software reliability models (SRM). SRM is a mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment. Due to defect identification and removal the failure rate (failures per unit of time) of a software system generally decreases over time. Software reliability modeling is done to estimate the form of the curve of the failure rate by statistically estimating the parameters associated with the selected model. The purpose of this measure is twofold: 1) to estimate the extra test time required to meet a specified reliability objective and 2) to identify the expected reliability of the software after release (IEEE Std. 1633-2008).

SRGM can be applied to guide the test board in their decision of whether to stop or continue the testing. These models are grouped into concave and S-Shaped models on the basis of assumption about cumulative failure occurrence pattern. The S-Shaped models assume that the occurrence pattern of cumulative number of failures is S-Shaped: initially the testers are not familiar with the product, then they become more familiar and hence there is a slow increase in fault removing. As the testers' skills improve the rate of uncovering defects increases quickly and then levels off as the residual errors become more difficult to remove. In the concave shaped models the increase in failure intensity reaches a peak before a decrease in failure pattern is observed. Therefore the concave models indicate that the failure intensity is expected to decrease exponentially after a peak was reached.

From exhaustive study of the literature I come across *three research gaps*: SRGM have widely been used for reliability characterization of closed source software (CSS), but 1) there is no universally applicable model that can be applied in all cases, 2) applicability of SRGM for OSS is unclear and 3) there is no agreement on how to select the best model among several alternative models, and no specific empirical methodologies have been proposed, especially for OSS. My PhD work mainly focuses on these *three research gaps.*

In first step, focusing on the *first research gap*, I analyzed comparatively eight SRGM, including Musa Okumoto, Inflection S-Shaped, Geol Okumoto, Delayed S-Shaped, Logistic, Gompertz and Generalized Geol, in term of their fitting and prediction capabilities. These models have selected due to their wide spread use and they are the most representative in their category. For this study 38 failure datasets of 38 projects have been used. Among 38 projects, 6 were OSS and 32 were CSS. In 32 CSS datasets 22 were from testing phase and remaining 10 were from operational phase (i.e. field). The outcomes show that Musa Okumoto remains the best for CSS projects while Inflection S-Shaped and Gompertz remain best for OSS projects. Apart from that we observe that concave models outperform for CSS and S-Shaped outperform for OSS projects.

In the second step, focusing on the *second research gap*, reliability growth of OSS projects was compared with that of CSS projects. For this purpose 25 OSS and 22 CSS projects were selected with related defect data. Eight SRGM were fitted to the defect data of selected projects and the reliability growth was analyzed with respect to fitted models. I found that the entire selected models fitted to OSS projects defect data in the same manner as that of CSS projects and hence it confirms that OSS projects reliability grows similarly to that of CSS projects. However, I observed that for OSS S-Shaped models outperform and for CSS concave shaped models outperform.

To overcome the *third research gap* I proposed a method that selects the best SRGM among several alternative models for predicting the residuals of an OSS. The method helps the practitioners in deciding whether to adopt an OSS component, or not in a project. We test the method empirically by applying it to twenty one different releases of seven OSS projects. From the validation results it is clear that the method selects the best model 17 times out of 21. In the remaining four it selects the second best model.

Following is the *list of publications* relevant to the subject of my thesis.

1. N. Ullah, M. Morisio, A. Vetro'. A method for selecting software reliability growth model to predict Open Source Software residual defect. In: IEEE Computer Journal. (in press)
2. N. Ullah. A method for predicting residual defects of Open Source Software. In: Software Quality Journal. (in press)
3. N. Ullah, M. Morisio .An Empirical analysis of Open Source Software Defect data through Software Reliability Growth Models. In: IEEE EUROCON 2013 in Zagreb, Croatia, 1-4 July 2013.
4. N. Ullah, M. Morisio . An Empirical Study of Reliability Growth of Open versus Closed Source Software through Software Reliability Growth Models. In: The 19th Asia-Pacific Software Engineering Conference, Hong Kong, December 4-7, 2012.
5. N. Ullah, M. Morisio, A. Vetro'. A Comparative Analysis of Software Reliability Growth Models using defects data of Closed and Open Source Software. In: 35th Annual IEEE Software Engineering Workshop, Heraclion, Crete, Greece, 12-13 October 2012.
6. N. Ullah, M. Morisio . An Empirical Study of Open Source Software Reliability Models. In: 2011 IEEE International Conference on Computational Intelligence and Computing Research, Cape Institute of Technology, Levengipuram, Kanyakumari, India, Dec 15-18, 2011.

Acknowledgements

All glory be to Allah, the Almighty and the most merciful. I am very thankful to my Allah with the blessing of Whom I achieve success in every aspect of my life.

First of all, I would like to express my gratitude to my parents for making me what I am. I would also like to thank my wife for being understanding and supportive throughout the duration of my PhD. And of course, a bundle of thanks and love to my little son Talha and my little daughter Laiba for being so brave to be without their *papa* for such a long period of time.

I would like to express profound gratitude to my supervisors Prof. Maurizio Morisio for allowing me to do my PhD work under his supervision. My professor has provided me with professional guidance and advice throughout my studies, and has always gone all the way to help me whenever necessary. I would also like to thank my senior colleague Dr. Antonio Vetro'. He has been a great source of inspiration and motivation for me. His problem solving skills have saved me a number of times whenever I was stuck in technical details of my work. It has been a privilege for me to work under both their able guidance.

I also appreciate the help and support extended to me by the staff at Department of Control and Computer Engineering. I am also indebted to all my colleagues and friends, who have been very helpful in both academic and extra-academic matters. While it may not be possible to mention everyone by name, you know who you are, and you will always have respect and appreciation from me.

Finally, a word of thanks to the Higher Education Commission of Pakistan for providing me fund for my studies at Turin.

1. Software reliability engineering concepts

Software reliability engineering is related to a very important software quality attribute: reliability. Software reliability is defined as the probability of failure-free operation for a specified period of time in specified environment. Software reliability is generally accepted as the key aspect in software quality since it quantifies software failure-which can make a powerful system inoperative. As a result, reliability is an essential ingredient in customer satisfaction.

In many engineering disciplines reliability engineering is a daily practiced technique. For instance civil engineers use it to construct buildings and computer hardware engineers use it to design chips and computer. Using a similar concept in these disciplines, we define software reliability engineering (SRE) as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability. SRE therefore includes:

1. Software reliability measurement, which includes estimation and prediction, with the help of software reliability models established in the literature.
2. The attributes and metrics and product design, development process, system architecture, software operational environment, and their implications on reliability.

This thesis composed of five chapters.

1. Chapter 1 reviews reliability concepts, the major classification of software reliability models and model evaluation techniques that appear in the literature.
2. Chapter 2 provides literature review and describes the *three research gaps* upon which my PhD research has focused.
3. Chapters 3, 4, and 5 present the studies that overcome the *three research gaps*.

1.1 Basic Definitions

We notice three major components in the definition of software reliability: failure, time, and operational environment. We now define these terms and SRE related

terminologies. These definitions have been taken from the handbook on software reliability engineering by lyu.

Failures: A *failure* occurs when the user perceives an unexpected behavior of the software system.

Fault: The cause of *failure* is said to be a *fault*. It is also referred to a *bug*.

In short, a software failure is an unexpected software behavior perceived by users, while a software fault is the identified cause of the software failure.

Defects: When the distinction between fault and failure is not critical, *defect* can be used as a generic term to refer to either a fault (cause) or a failure (effect).

Time: Reliability quantities are defined with respect to time. We are concerned with three types of time: the *execution time* for a software system is the CPU time that is actually spend by the computer in executing the software; the *calendar time* is the time people normally experience in terms of years, months, etc.; and the *clock time* is the elapsed time from start to end of computer execution in running the software. In measuring clock time, the periods during which the computer is shut down are not counted.

It is generally accepted that execution time is more adequate than calendar time for software reliability measurement and modeling. However, reliability quantities must ultimately be related back to calendar time for easy human interpretation, particularly when managers, engineers and customers want to compare them across different systems.

Operational Profile: The *operational profile* of a system is defined as the set of operations that the software can execute along with the probability with which they will occur. An operation is a group of runs which typically involve similar processing.

Software reliability measurement: Measurement of software reliability includes two types of activities: reliability *estimation* and reliability *prediction*.

Estimation: This activity determines current software reliability by applying statistical inference techniques to failure data obtained during system test or during system operation. This is a measure regarding the achieved reliability

from the past until the current point. Its main purpose is to assess the current reliability and determine whether a reliability model is a good fit in retrospect.

Prediction: This activity determines future software reliability based upon available software metrics and measures. Depending on the software development stage, prediction involves different techniques:

1. When failure data are available (e.g., software is in system test or operation stage), the estimation techniques can be used to parameterize and verify software reliability models, which can perform future reliability prediction.
2. When failure data are not available (e.g., software is in the design or coding stage), the metrics obtained from the software development process and the characteristics of the resulting product can be used to determine reliability of the software upon testing or delivery.

The first definition is also referred to as *reliability prediction* and the second definition as *early prediction*. When there is no ambiguity in the text, only the word *prediction* will be used.

Most current software reliability models fall in the estimation category to do reliability prediction. Nevertheless, a few early prediction models were proposed and described in the literation. A survey of existing estimation models and some early prediction models can be found later in this chapter.

Software reliability model: A software reliability *model* specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment. Figure 1.1 shows the basic ideas of software reliability modeling.

In Fig. 1.1, the failure rate of a software system is generally decreasing due to the discovery and removal of software failures. At any particular time (say, the point marks “present time”), it is possible to observe a history of the failure rate of the software. Software reliability modeling forecasts the curve of the failure rate by statistical evidence. The purpose of this measure is twofold: 1) to predict the extra time needed to test the software to achieve a specified objective; 2) to predict the expected reliability of the software when the testing is finished.

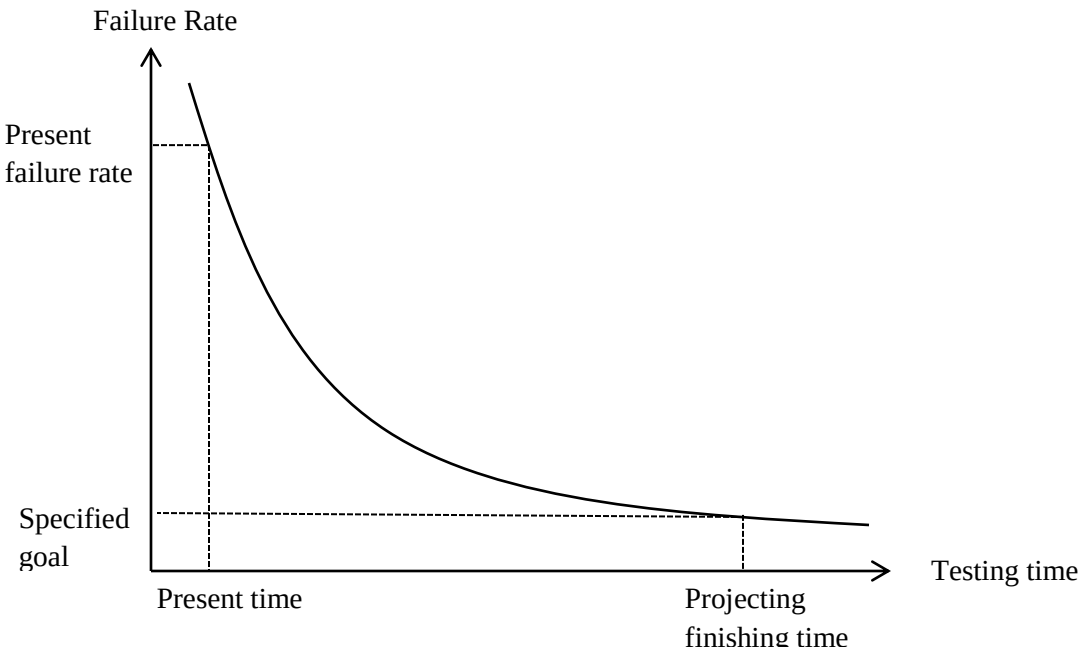

Figure 1-1: Basic ideas on software reliability modeling

1.2 Classification of software reliability models

The software reliability models are classified as Sharma et al. [26] described below:

1. **Early Prediction Model:** This type of model uses characteristics of the software development process from requirements to test and extrapolate this information to predict the behavior of software during operation [1].
2. **Software Reliability Growth Models (SRGM):** This type of model, based on failure behavior of software during testing, determines its behavior during operation. Hence, this category of models uses failure data information and trends observed in the failure data collected from system testing to derive reliability predictions [2]. The SRGMs are further classified as failure rate models, and NHPP models.
3. **Input Domain Based Models:** These models use properties of the input domain of the software to estimate a correctness probability from test cases that executed properly [3].

4. **Architecture Based Models:** This type of model measures the reliability of software on the basis of the relationship among different components and its interaction. These models base on logical complexity, decision point, program length, operands and operators of software [4]. The architecture based software reliability models are further classified into state based models, path based models, and additive models.
5. **Hybrid Black Box Models:** These models use the features of input domain based models, and SRGMs. Hence is said to be hybrid black box models.
6. **Hybrid White Box Models:** These models use selected features from both white box models, and black box models. Since these models consider the architecture of the system for reliability prediction, these models are therefore said to be hybrid white box models.

The proposed classification of software reliability models according to phases of Software development life cycle (SDLC) is shown in Fig. 1.2. The architecture based models are further classified by the method used to combine the architecture of the software with the failure behavior, whereas the SRGMs are further classified based on the software failure phenomenon, as given in Fig. 1.2.

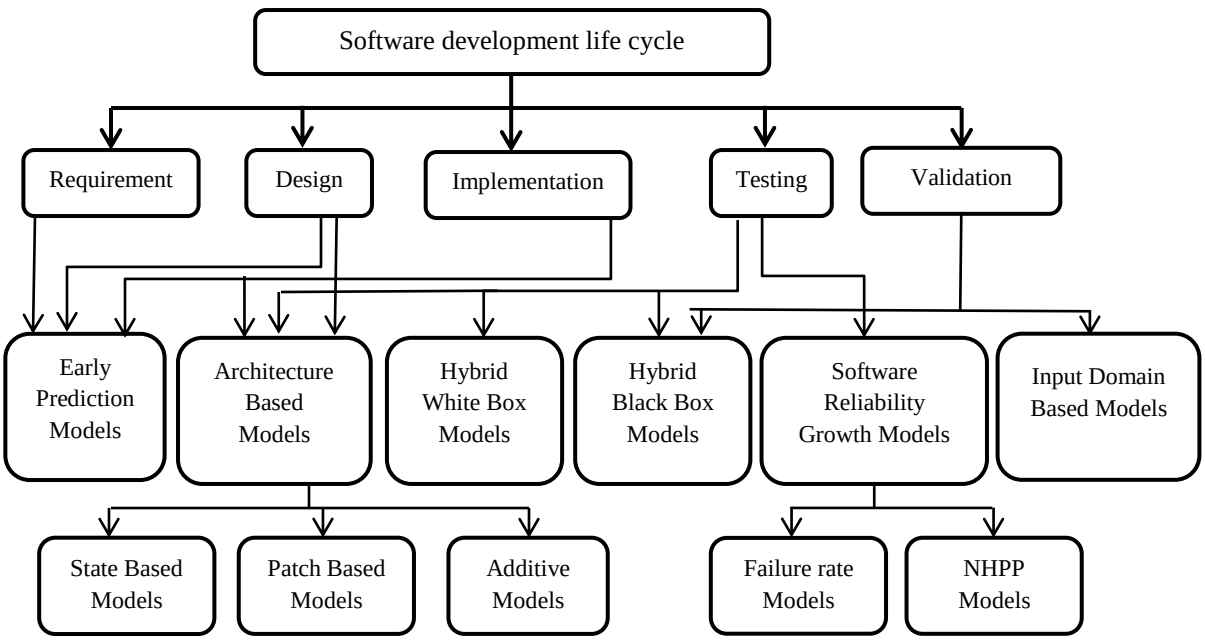

Figure 1-2: Classification of software reliability models

1.3 Comparison Criteria for Software Reliability Models

A model can be judged according to its ability to reproduce the observed behavior of the software, and to predict the future behavior of the software from the observed failure data. To investigate the effectiveness of software reliability models, a set of comparison criteria is proposed to compare models quantitatively. Following are the comparison criteria that appear in the literature.

1) The *Bias* is defined as [5], [6]

$$Bias = \frac{\sum_{i=1}^{k}(m(t_i) - m_i)}{k}$$

It is the sum of the difference between the estimated curve, and the actual data.

2) The *mean square error* (MSE) measures the deviation between the predicted values with the actual observations, and is defined as [7]

$$MSE = \frac{\sum_{i=1}^{k}(m_i - m(t_i))^2}{k - p}$$

3) The *accuracy of estimation* (AE) or *Prediction Relative Error* (PRE) can reflect the difference between the estimated numbers of all failures with the actual number of all detected failures. It is defined as [8]

$$PRE = \left|\frac{M_a - a}{M_a}\right|$$

where M_a, and a are the actual, and estimated cumulative number of detected failures after the test, respectively.

4) The *predictive-ratio risk* (PRR) is defined as [40]

$$PRR = \sum_{i=1}^{k}\frac{(m(t_i) - m_i)}{m(t_i)}$$

which measures the distance of model estimates from the actual data against the model estimate.

5) The *variance* is defined as [5], [6]

$$Variance = \sqrt{\frac{1}{1-k}\sum_{i=1}^{k}(m_i - m(t_i) - Bias)^2}$$

which is the standard deviation of the prediction bias.

6) The Root Mean Square Prediction Error (RMSPE) is a measure of the closeness with which the model predicts the observation. It is defined as [5], [6]

$$RMSPE = \sqrt{Variance^2 - Bias^2}$$

7) R^2 can measure how successful the fit is in explaining the variation of the data. It is defined as [8]

$$R^2 = 1 - \frac{\sum_{i=1}^{k}(m_i - m(t_i))^2}{\sum_{i=1}^{k}(m_i - \sum_{j=1}^{k}\frac{m_j}{n})^2}$$

8) The *sum of squared failures* (SSE) is defined as [9]

$$SSE = \sum_{i=1}^{k}(m_i - m(t_i))^2$$

9) The *Theil statistic* (TS) is the average deviation percentage over all periods with regard to the actual values. The closer Theil's Statistic is to zero, the better the prediction capability of the model. It is defined as [10]

$$TS = \sqrt{\frac{\sum_{i=1}^{k}(m(t_i) - m_i)^2}{\sum_{i=1}^{k} {m_i}^2}} * 100\%$$

In (1)–(9), k represents the sample size of the data set, and p is the number of parameters.

1.4 Summary

In this chapter the definitions of major terms in SRE are provided, and fundamental concepts in software reliability modeling and measurement are discussed. Further, major classes of software reliability models and their evaluation metrics are described.

2 Literature review[1]

In this chapter we analyze the relevant literature. First we consider the issue of the selection of a model to characterize the reliability of a software product. Then we analyze the application of reliability modeling to open source software (OSS).

2.1 Model Selection

Over the past 40 years many Software reliability growth models (SRGM) have been proposed for software reliability characterization. With so many models available, the question is therefore which model to choose in a given context. Musa et al. [27] have shown that some families of models have certain characteristics that are considered better than others; for example, the geometric family of models (i.e. models based on the hyper-geometric distribution) have a better prediction quality than the other models. Goel [12] stated that different models predict well only on certain data sets; and the best model for a given application can be selected by comparing the predictive quality of different models. But there is also convincing evidence that there is no way to select a model a priori, in function of the characteristics of a project [26, 33, 11]. So, many authors propose methods for model selection [32, 33, 34]. Brocklehurst et al. [11] proposed that the nature of software failures makes the model selection process in general a difficult task. They observed that hidden design flaws are the main causes of software failures. Ghaly et al. [13] analyzed the predictive quality of 10 models using 5 methods of evaluation. They observed that different criteria of model evaluation select a different model as best predictor. Also, some of their methods were rather subjective as to which model was better than others. Khoshgoftaar [14] suggested Akaike Information Criteria (AIC) as best model selection method. Subsequent work by Khoshgoftaar and Woodcock [15] proved the feasibility of using the AIC for best model selection. Khoshgoftaar and Woodcock [16] proposed a method for the selection of a reliability model among various alternatives using the log-likelihood function (i.e. a function of the parameters of the models). They applied the method to the failure logs of a project. Lyu and Nikora [17] implemented Goodness-of-Fit (GOF) in their model selection tool. In [18] the authors have proposed a method for software reliability model selection based on experiences from past software projects, which considers the information about models used in past software projects. But the problems in this method are how to measure the similarity of software projects and how to make full use of past project experiences. Stringfellow et al

[1] **The content of this chapter has been modelled according to N. Ullah, Morisio M, Vetro'.A. A method for predicting residual defects of Open Source Software. In: Software Quality Journal. (in press)**

[34] developed a method that selects the appropriate SRGM and may help in decision making on when to stop testing and release the software.

Table 2-1: Summary of papers about model selection

Paper	Models applied	Focus of paper	Project data set	Empirical evaluation result
Sharma et al., [26]	16 SRGM	Fitting	2 CSS	Different models must be evaluated, compared and then the best one should be chosen
Sukert, [33]	3 Non-geometric models	Prediction	4 CSS	No universally applicable model available. A model behaves different for different datasets.
Brocklehurst et al., [11]	9 SRGM	Prediction	1 CSS	A user cannot select a model a priori from the available models and know that it is the best for a given context.
Ghaly et al., [13]	10 SRGM	Prediction	3 CSS	Different methods of model evaluation select different model as best predictor.
Khoshgoftaar, [14]	5 SRGM	Prediction	1 CSS	During development in system test phase the models prediction were evaluated 5 times after different interval of time; 1 week, 3 weeks, 1 week, and 5 weeks. 5/5 selects best model
Stringfellow et al., [34]	4 SRGM	Prediction	3 CSS	The proposed method selects best model in all cases.

Error! Reference source not found. summarizes papers about model selection that have n empirical evaluation. The first column reports the reference of the paper, the second column reports the reliability models applied, the third column reports the focus of the paper (fitting = evaluating models on goodness on fit, without considering their predictive capability, prediction = fitting models and evaluating them on their predictive capability), the fourth column reports the number and type (CSS = closed source software) of projects used for validation, and the rightmost column reports the key result of the paper. Overall there is agreement that models should be selected case by case. There is no universally accepted selection measuring criterion, and all the criteria reported have been evaluated on very few projects.

2.2 Application of SRGM to Open Source Software

More recently the literature has concentrated on OSS. The key research question is whether OSS behaves, from a reliability point of view, in the same way as CSS. If yes, the next question is which model, or family of models, should be used. Mohamad et al. [19] examined the defect discovery rate of two OSS products with software developed in-

house using 2 SRGM. They observed that the two OSS products have a different profile of defect discovery. Zhou et al. [20] analyzed bug tracking data of 6 OSS projects. They observed that along their developmental cycle, OSS projects exhibit a similar reliability growth pattern with that of closed source projects. They proposed the general Weibull distribution to model the failure occurrence pattern of OSS projects. Rossi et al. [21] analyzed the failure occurrence pattern of 3 OSS products applying SRGM. They also proposed that the best model for OSS is the Weibull distribution. However, Rahmani et al. [22] obtained a different result: they compared the fitting and prediction capabilities of 3 models using failure data of 5 OSS projects and they observed that Schneidewind model was the best while Weibull was the worst one. Fengzhong et al. [23] examined the bug reports of 6 OSS projects. They modelled the bug reports using nonparametric techniques and they suggested that Generalized Additive (GA) models and exponential smoothing methods are suitable for reliability characterization of OSS projects. In Li et al. [24] the results show that SRGM can be used for reliability characterization of OSS projects.

Table 2-2: Summary of papers about reliability in OSS

Paper	**Models applied**	**Focus of paper**	**Project data set**	**Empirical evaluation result**
Mohamad et al., [19]	2 SRGM	Fitting	2 OSS & 1 CSS	OSS has different profile of defect arrival from CSS SRGM cannot be used for OSS
Zhou et al., [20]	Weibull distribution	Fitting	8 OSS	OSS has same profile as CSS Weibull distribution suitable for OSS
Rossi et al., [21]	6 SRGM	Fitting	3 OSS	SRGM can be used for OSS, Best model is Weibull
Rahmani et al., [22]	3 SRGM	Prediction	5 OSS	Weibull fits best but does not predict well compared to the other selected models
Fengzhong et al., [23]	Non-parameterized models	Fitting	6 OSS	Non-parameterized models are most suitable
Li et al., [24]	2 SRGM	Fitting	6 OSS	SRGM can be used for OSS

Error! Reference source not found..2 (that has the same format as **Error! Reference urce not found.**) summarizes papers about the characterization of reliability in OSS that have an empirical evaluation. In summary the evidence is somehow conflicting, but also suggesting that OSS projects can be characterized by the same models used for CSS.

From exhaustive study of literature we came across *three research gaps*, which are briefly described in figure 2.1.

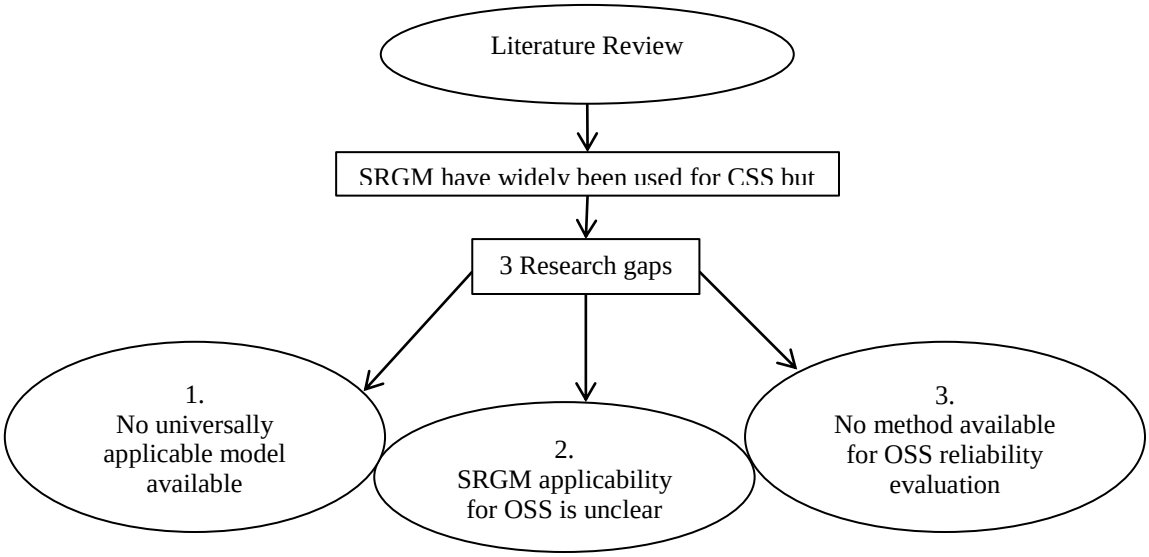

Figure 2-1: Literature review: Research

Research gaps: Herein we briefly describe the *research gaps* upon which my PhD research has focused.

1. No universally applicable model available that can be used for reliability characterization of every project.
2. The available literature regarding applicability of SRGM to OSS is limited and unclear.
3. There is a dilemmas of model exist and a distinctive method is needed to be developed for selecting appropriate model among several alternative models that characterizes very well the reliability of an OSS component before its adoption.

These *three research gaps* are addressed in chapter 3, 4 and 5 respectively. In the next section we comprehensively describe the SRGM.

2.3 Background

2.3.1 Software Reliability Growth Models (SRGM)

In the literature software reliability models (SRM) are mainly classified as white box and black box approaches. Architecture based models are known as white box models since these models attempt to measure the reliability of a software system based on its structure that is normally architected during the specification and design of the product. Relationships among software components are thus the focus for white box software

reliability measurement. Based on architecture based models decision are being made as how to design reliability into system [28]. SRGM are known as black box models because these models treat the entire software system as a single entity, thus ignoring software structures and components interdependencies. These models quantify the reliability in testing phase and influence the release decision. The models rely on the testing data collected over an observed time period [28].

SRGM assume that reliability grows after a defect has been detected and fixed. SRGM can be applied to guide the test board in their decision of whether to stop or continue the testing. These models are grouped into concave and S-Shaped models on the basis of assumption about cumulative failure occurrence pattern. The S-Shaped models assume that the occurrence pattern of cumulative number of failures is S-Shaped: initially the testers are not familiar with the product, then they become more familiar and hence there is a slow increase in fault removing. As the testers' skills improve the rate of uncovering defects increases quickly and then levels off as the residual errors become more difficult to remove. In the concave shaped models the increase in failure intensity reaches a peak before a decrease in failure pattern is observed. Therefore the concave models indicate that the failure intensity is expected to decrease exponentially after a peak was reached. There are some basic assumptions which are similar for all these kinds of models [29].

- When a defect is detected it is removed immediately.
- A defect is corrected instantaneously without introducing new defect into the software.
- The software is operated in a similar manner as that in which reliability predictions are to be made.
- Every defect has the same chance of being encountered within a severity class as any other defect in that class.
- The failures, when the defects are detected, are independent.

Software Reliability Growth Models measure and model the failure process itself. Because of this, they include a time component, which is characteristically based on recording times t_i of successive failures i ($i \geq 1$). Time may be recorded as execution time or calendar time. These models focus on the failure history of software. These models use a non-homogeneous Poisson process (NHPP) to model the failure process. The NHPP is characterized by its mean value function, m (t). This is the cumulative number of failures expected to occur after the software has executed for time *t*. Let {N (t), t> 0} denote a counting process representing the cumulative number of defects detected by the time *t*. A SRGM based on an NHPP with the mean value function (MVF) $m(t)$ can be formulated as [29].

P {N (t) = n} = $\frac{m(t)^n}{n!} e^{-m(t)}$, n = 1, 2……..

Where $m(t)$ represent the expected cumulative number of defects detected by the time *t*. The MVF, $m(t)$ is non-decreasing in time *t* under the bounded condition $m(\infty)$ = a, where 'a' is the expected total number of defects to be eventually detected. Knowing its value can help us to determine whether the software is ready to be released to the customers and how much more testing resources are required. The MVF, $m(t)$ provides prediction of the expected total number of failures at time *t*. The MVF, $m(t)$ quite often is defined as a parametric function depending on two or more unknown parameters. Generally, we can get distinct NHPP models by using different non-decreasing mean value functions.

For this thesis we selected eight SRGMs due to their wide spread use in literature and they are the most representative in their category. Brief comprehensive descriptions of MVF, $m(t)$ for each model are given as fallow:

2.3.1.1 Musa Okumoto:

This model was proposed by J. D. Musa and K. Okumoto. It assumes that failure occurrence pattern follows a concave shape and does not include the learning curve [30]. The MVF is

$$m(t) = aln(1 + bt), \quad a > 0, b > 0$$

Where 'a' is the expected total number of defects to be eventually detected and 'b' is the defect detection rate.

2.3.1.2 Inflection S-Shaped:

It was proposed by Ohba and its underlying concept is that the observed software reliability growth becomes S-shaped if defects in a program are mutually dependent, i.e., some defects are not detectable before some others are removed [31]. The mean value function is

$$m(t) = a\frac{1-\exp[-bt]}{1+\Psi(r)\exp[-bt]}, \ \Psi(r) = \frac{1-r}{r}$$

$$a > 0, b > 0, r > 0$$

The parameter 'r' is the inflection rate that represents the ratio of the number of detectable defects to the total number of defects in the software, 'a' is the expected total number of defects to be eventually detected, 'b' is the defect detection rate, and Ψ is the inflection factor.

2.3.1.3 Goel Okumoto:

This model, proposed by Goel and Okumoto, is one of the most popular NHPP model in the field of software reliability modeling [29, 30, 31]. It is also called the exponential NHPP model. Considering defect detection as a NHPP with an exponentially decaying rate function, the mean value function is postulated in this model as

$$m(t) = a(1 - \exp[-bt]),\ a > 0, b > 0.$$

Where 'a' is the expected total number of defects to be eventually detected and 'b' represents the defect detection rate.

2.3.1.4 Delayed S-Shaped:

Delayed S-Shaped model is a modification of the NHPP to obtain an S-shaped curve in order to model the cumulative number of failures detected such that the failure rate initially increases and later decays [29, 31]. This model can be thought of as a generalized exponential model with failure rate first increasing and then decreasing. The software defect detection process described by such an S-shaped curve can be regarded as a learning process because the testers' skills will gradually improve as time progresses. The MVF is

$$m(t) = a(1 - (1 + bt)\exp[-bt]), a > 0, b > 0.$$

Where 'a' and 'b' are the expected total number of defects to be eventually detected and the defect detection rate, respectively.

2.3.1.5 Logistic:

Generally software reliability tends to improve and it can be treated as a growth process during the testing phase. This reliability growth occurs due to the fact that a defect is fixed when it is detected. Therefore, under some conditions, the models developed to predict economic population growth could also be applied to predict software reliability growth. These models simply fit the cumulative number of detected defects at a given time with a function of known form. Logistic growth curve model is one of them and it has an S-shaped curve [31]. Its MVF is

$$m(t) = \frac{1}{1+k\exp[-bt]}, a > 0, b > 0, k > 0,$$

Where 'a' is the expected total number of defects to be eventually detected and 'k' and 'b' are parameters which can be estimated by fitting the defect data.

2.3.1.6 Yamada Exponential:

This model includes initial learning curve and belongs to S-shape class of SRGM [32]. The mean value function of this model is

$$m(t) = a\left(1 - e^{-r\left(1-e^{-bt}\right)}\right), a > 0, b > 0, r > 0.$$

The parameter 'r' is the inflection rate that represents the ratio of the number of detectable defects to the total number of defects in the software, 'a' is the expected total number of defects to be eventually detected, and 'b' is the defect detection rate.

2.3.1.7 Gompertz:

It is one of the simplest S-shaped software reliability growth models. Due to its simplicity many Japanese computer manufacturers and software houses have adopted this model. Gompertz Growth Curve Model is used in the Fujitsu and Numazu work [31]. Its MVF is

$$m(t) = ak^{b^t}, a > 0, 0 < b < 1, 0 < k < 1,$$

Where 'a' is the expected total number of defects to be eventually detected, 'b' and 'k' are parameters whose values are estimated using regression analysis.

2.3.1.8 Generalized Goel:

In order to describe the S-Shaped nature of software failure occurrence, Goel proposed a simple generalization of the Goel-Okumoto model with an additional parameter 'c' [31]. The mean value function is

$$m(t) = a(1 - \exp[-bt^c]), a > 0, b > 0, c > 0.$$

Where 'a' is the expected total number of defects to be eventually detected, 'b' and 'c' are parameters that reflect the quality of testing.

The next section provides a quick refresher on software reliability modelling.

2.3.2 Reliability Modeling

Software reliability model (SRM) is a mathematical expression that specifies general form of failure occurrence as a function of fault introduction, fault removal and operational environment [25]. SRM can both assess and predict reliability. In reliability assessment SRM are fitted to the collected failure data using statistical techniques (e.g. Linear Regression, Non Linear regression) based on the nature of collected data. In reliability prediction, the total number of expected future failures is forecasted on the basis of fitted SRM. Both assessment and prediction need good data, which implies accuracy, i.e. data is accurately recorded at the time the failures occurred and pertinence, i.e. data relates to an environment that resembles to the environment for which the

forecast is performed [25]. For reliability modeling, software systems are tested in an environment that resembles to the operational environment. When a failure (i.e. an unexpected and incorrect behavior of the system) occurs during testing, it is counted with a time tag. Cumulative failures are counted with corresponding cumulative time. SRM is fitted to the collected data and the fitted models are then used to predict the total number of expected defects (i.e. fault on the execution of which failure occur) in the software.

Hence, typically reliability modeling is composed of 5 steps: keeping a log of past failures, plotting the failures, determining a curve (i.e. Model) that best fits the observations, measuring how accurate the curve model is and then using the best fitted model predicting the future reliability in terms of predicting total number of expected defects in the software system.

Figure 2.2 gives an overview of the reliability modeling. In first phase reliability objective is defined and operational profile is developed, then software is tested in the environment resemble to the operational one. Failure data is collected in such a way that each failure is counted with the time tag in which the failure is detected. Apply the SRM to the collected failure data and estimate the reliability, if the predefined reliability objectives meet, deploy the software otherwise continue testing. After releasing the software the predicted reliability is validated in the field and feedback for next release is taken.

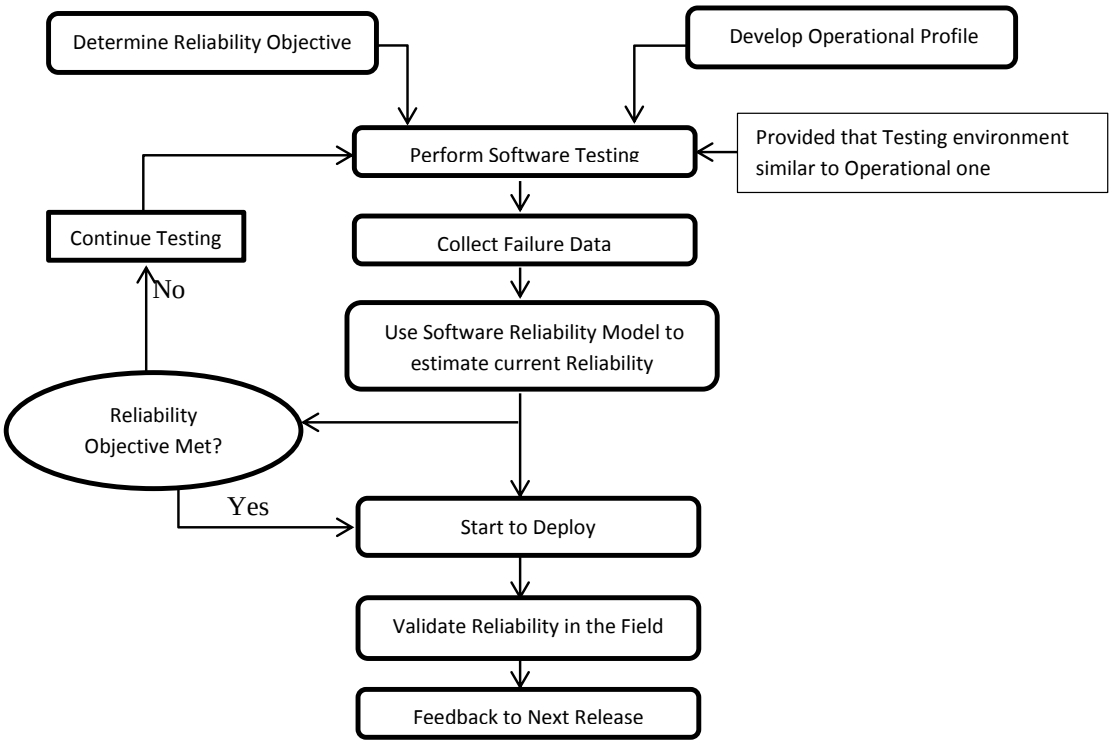

Figure 2-2: Software reliability engineering process overview during testing phase

3. Comparison of software reliability growth models[2]

Herein this chapter we focus on the first research gap: There is no agreement on what is the best reliability model for a given project, especially at its inception. Different models predict well only on certain data sets and the best model can be selected by comparing the predictive qualities of a number of models only at the end of a project. To overcome this research gap we briefly compare the reliability characterization and prediction quality of different SRGM in order to draw a general conclusion about the best fitting and best predictor models among them. Such knowledge will help project managers in the selection of a good SRGM model and in making an informed decision on the release of the product.

3.1 Goal

The goal and the research questions that have been driven to overcome the first research gap, can be summarized as given below using the GQM [51] template.

Object of the study	Analyze different SRGM models
Purpose	to compare
Focus	Their capability to characterize and predict the reliability of a project
Stakeholder	from the point of view of maintenance and quality managers
Context factors	in the context of industrial and open source systems

We describe the research questions and metrics that complete the GQM. The first step is analyzing the capability of models to simply fit the data sets. At this regard we define RQ1 and compare the fitting capability, in terms of goodness of fit (i.e. R2) of the models on the whole dataset. The second step is analyzing the capability of prediction. To this purpose we use the first two thirds of the data sets to fit models, and estimate the

[2] **The contents of this chapter have been modelled according to N. Ullah, Morisio M, Vetro'.A. A Comparative Analysis of Software Reliability Growth Models using defects data of Closed and Open Source Software. In: 35th Annual IEEE Software Engineering Workshop, Heraclion, Crete, Greece, 12-13 October 2012.**

remaining third. The two thirds threshold was selected following [50]. To the regard of prediction we have two different research question (RQ). RQ2 simply compares the models in terms of prediction relative error (PRE) and Theil's Statistics (TS). RQ3 tries to help in selecting a model, taking the point of view of a project manager who only has available part of the dataset and needs to select a model for prediction. So RQ3 analyzes if a model with a good fit (high R^2) is also a good predictor.

The RQs are now presented in detail.

3.1.1 RQ1: Which SRGM models fit best?

Or, in operational terms, which SRGM has the best R^2? Models are fitted on the whole data sets, and their R^2 are analyzed and compared. Model fitting is required to estimate the parameters of the models and produce a prediction of failures. Fitting can be done using Linear or Non Linear Regression (NLR). In linear regression, a line is determined that fit to data, while NLR is a general technique to fit a curve through data. The parameters are estimated by minimizing the sum of the squares of the distances between data points and the regression curve. We use NLR fitting due to the nature of data.

NLR is an iterative process that starts with initial estimated values for each parameter. The iterative algorithm then gradually adjusts these until to converge on the best fit so that the adjustments make virtually no difference in the sum-of-squares. A model's parameters do not converge to best fit if the model cannot describe the data. On consequence the model cannot fit to the data. On the contrary, in case of convergence of the iterative algorithm, the R2 is the metric that indicates how successful the fit is. R2 takes a value between 0 and 1, inclusive. The closer the R2 value is to one, the better the fit.

We consider a good fit when $R^2 > 0.90$. Given that the number of datasets is not statistically significant for at least Field datasets (10) and OSS datasets (6), we do not test the hypothesis. Instead we analyze and rank models based on their R^2.

3.1.2 RQ2: Which SRGM models are good predictors?

Or in operational terms, which models have best TS (for prediction accuracy) and PRE (for prediction correctness). Models are fitted on the first two thirds of the datasets, TS and PRE are computed on the remaining third.

Prediction capability can be evaluated under two points of view, accuracy and correctness. Accuracy deals with the difference between estimated and actual over a time period. Correctness deals with the difference between predicted and actual at a specific point in time (e.g. release date). A model can be accurate but not correct and vice versa.

For this reason we use the Theil's Statistic (TS) for accuracy and Predicted Relative Error (PRE) for correctness.

We consider a prediction as good if TS is below 10% and PRE is within the range [-10%, +10%] of total number of actual defects. As for RQ1, we do not build hypothesis testing given the low number of datasets in Field and OSS defect types, but we rank models based on their TS and PRE.

3.1.3 RQ3: A model with good fit is also a good predictor?

Or, in operational terms, a model with a good R^2 also has good TS and PRE? Models are fitted on two thirds of the data sets, the R^2 is computed on this fit, TS and PRE are computed on the remaining third of the dataset.

RQ2 tries to understand what models are best predictors, but takes an a posteriori view. RQ3 takes an a priori view, or uses the view of a project manager who has to decide what model to use before the end of the project (at two thirds of it), and only has the goodness of fit as a rationale for a decision.

3.2 Collected datasets

The goal of the study is to analyze the selected SRGM described in previous chapter using as many software failure data sets as possible. For this purpose we collected failure data from the literature. We have searched papers on IEEE Explorer, ACM Digital Library and in three journals, i.e. Journal of Information and Software Technology, the Journal of System and Software and IEEE software. For papers searching these strings have been used:

- *Software failure rate*
- *Software failure intensity*
- *Software failure Dataset*
- *Failure rate and Reliability*
- *Failure intensity and Reliability*

We found 2100 papers, 19 of which relevant for our study because they contained failure data sets on 38 projects. Among these, 32 projects were closed source and 6 were Open Source. In 32 closed source projects, 22 contain system test failure data and 10 contain field defect data collected from the operation phase. OSS projects data have no distinction between phases. Table 3.1 lists the references of the selected projects and their names. We have selected both system test and field defect data sets in order to evaluate the best fitting and best predictor SRGM for both phases (i.e. system test, operation) because the software reliability models are used for the prediction of failure in both

phases and the phase may be a factor for model selection. Six data sets contain defect data of two OSS Projects, Apache and GNOME. Three data sets have been collected from different versions of each of the two OSS projects.

Table 3-1: Projects Details of the collected Data Sets

Ref.	Project	Type	Phase	Dataset ID
[35]	Pocket Power View Software	Closed Source	System Test	DS1
[40]	Network Management System, R1	Closed Source	System Test	DS2
	Network Management System, R2	Closed Source	System Test	DS3
[44]	NTDS Data Set-Musa's Data Set	Closed Source	System Test	DS4
[46]	Real Time Command & Control System	Closed Source	System Test	DS5
	Dataset2 from Ohba's Study 1984	Closed Source	System Test	DS6
[37]	Tandem Computer Software, R1	Closed Source	System Test	DS7
	Tandem Computer Software, R2	Closed Source	System Test	DS8
	Tandem Computer Software, R3	Closed Source	System Test	DS9
	Tandem Computer Software, R4	Closed Source	System Test	DS10
[36]	Real Time Control system-Musa's Dataset1	Closed Source	System Test	DS11
[44]	Musa's Dataset2	Closed Source	System Test	DS12
[49]	Telecommunication Product	Closed Source	System Test	DS13
[46]	Large Medical Record System, R1	Closed Source	System Test	DS14
	Large Medical Record System, R2	Closed Source	System Test	DS15
	Large Medical Record System, R3	Closed Source	System Test	DS16
[48]	Wireless Network Management System, R3	Closed Source	System Test	DS17
[43]	Telecommunication Product B	Closed Source	System Test	DS18
	Telecommunication Product C	Closed Source	System Test	DS19
[38]	Software Testing Data of Armored Force Eng. Institute, China	Closed Source	System Test	DS20
	National Tactical Data System	Closed Source	System Test	DS21
[41]	Wireless Telecommunication Product, R3	Closed Source	System Test	DS22
[39]	Stratus-I	Closed Source	Operational	DS1
	Stratus-II	Closed Source	Operational	DS2
[40]	Network Management System, R1	Closed Source	Operational	DS3
[48]	Wireless Network Management System, R1	Closed Source	Operational	DS4
	Wireless Network Management System, R2	Closed Source	Operational	DS5
	Wireless Network Management System, R3	Closed Source	Operational	DS6
[42]	Telecommunication Product for voice and Data	Closed Source	Operational	DS7
[45]	PSO Product A	Closed Source	Operational	DS8
[41]	Wireless Telecommunication Product, R1	Closed Source	Operational	DS9
	Wireless Telecommunication Product, R2	Closed Source	Operational	DS10
[47]	GNOME V2.0	Open Source		DS1
	GNOME V2.2	Open Source		DS2
	GNOME V2.4	Open Source		DS3
	Apache V2.0.35	Open Source		DS4
	Apache V2.0.36	Open Source		DS5
	Apache V2.0.39	Open Source		DS6

3.3 Results

3.3.1 RQ1: Which SRGM models fit best?

First we consider the basic capability of a model to fit the dataset (fits or not), irrespective of the goodness of fit (R^2). Figure 3.1 reports for each model on the X axis the percentage of datasets fitted (axis Y) in each data group (colour bars). Musa Okumoto fitted to all data sets in each group, most models also fit, except Yamada exponential, Gompertz, Generalized Goel that fit poorly, especially field test and system test datasets.

Now let's analyze goodness of fit too. Figure 3.2 reports how many times a model is the one with best R^2. For instance Musa Okumoto has the best R^2 on 60% of field defect datasets. Musa Okumoto is top performer on field defects data; Gompertz has very good results on OSS datasets. But apart from that there is no clear winner.

However, analyzing the top performer only as in Figure 3.2 can be misleading, in case many models fit with a similar R^2 the same dataset. Therefore in boxplots of Figure 3.3 we report the boxplots of R^2 per model and per dataset category. It should be reminded however that boxplots of Figure 3.3 excludes the models that did not fit at all the datasets (Figure 3.1) and is therefore meaningful for all models except Yamada exponential, Gompertz, and Generalized Goel. Musa Okumoto remains the best performer (no outliers and narrow boxplot on all categories of datasets). Next to it the other models have also narrow boxplot (always better than 0.9, the threshold depicted as a red horizontal line) but some outliers. It should be noted that all models behave extremely well (R^2 close to 1 and no outliers) on the OSS datasets.

In summary:

- Considering plain fitting (Fig 3.1), Musa Okumoto fits all datasets, Yamada exponential, Gompertz, Generalized Goel fit all OSS datasets but behaves poorly on the others, and the others fit at least 80% of the data sets.

- Considering the R^2 of models that fit the datasets (Figure 3.5). Musa Okumoto has always a very good fit (better than 0.9), the others, except Generalized Goel, also perform quite well but with outliers.

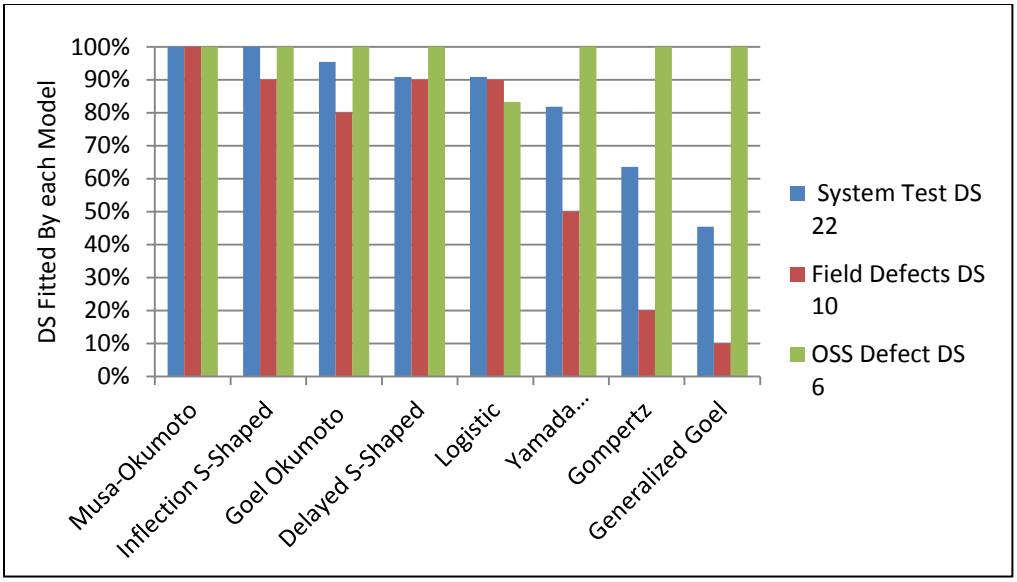

Figure 3-1: No of DS fitted by each Model

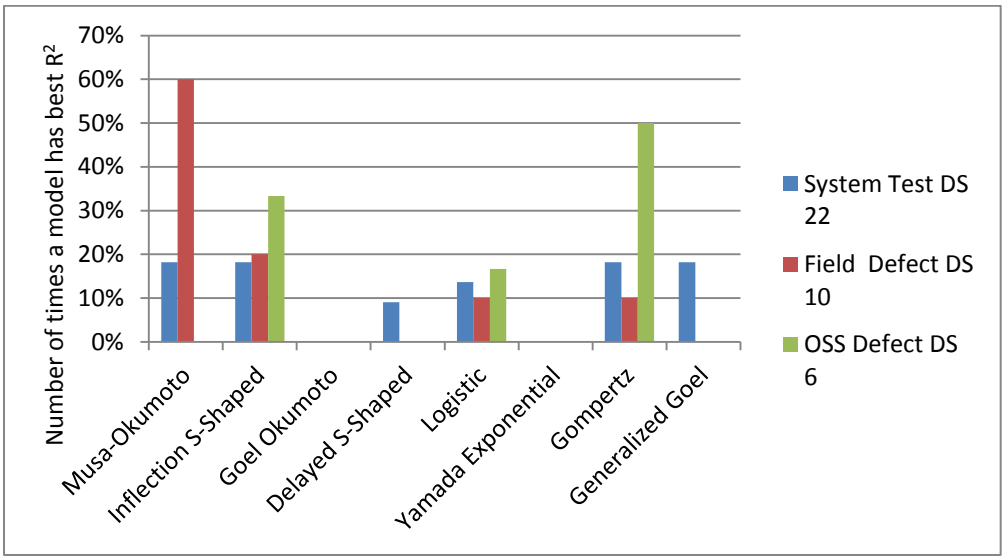

Figure 3-2: Ranking on Best Fitting-R^2

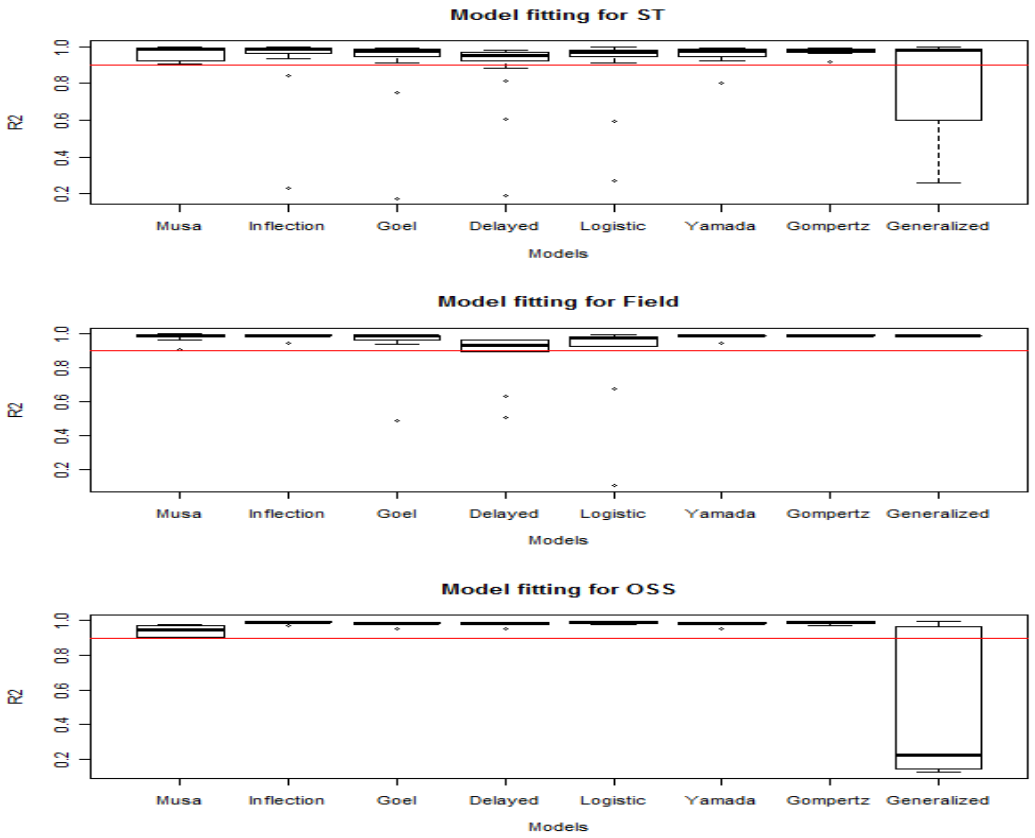

Figure 3-3: Box Plots of fitting (R^2) values

3.3.2 RQ2: Which SRGM models are good predictors?

For models that could fit the data set we used the first two-third data points of the data sets to train the model, and predicted the last third. We analyse their predicting capability in terms of accuracy and precision.

Accuracy

Figure 3.4 shows the number of times a model is the best predictor in terms of TS. It is clear that Musa Okumoto outperforms the others in both the industrial datasets. The Logistic Model is directly behind it. On the contrary in the case of OSS we got several ties (this explains why the sum of percentage might be greater than 100%), and the best model is Logistic followed by Inflection S-Shaped and Gompertz.

Figure 3.5 reports the TS values for all datasets. The red line represents the 0.1 threshold, usually considered indicator of good accuracy. Figure 6 allows us to discuss good models, instead of best model as in Figure 3.4.

In System Test data, all models show variations, however Musa Okumoto and Inflection S-Shaped have a median below the 0.1 threshold. This happens also for Gompertz and the Generalized Goel models; however they do not fit in several cases (Figure 1).

In Field data sets all models except Delayed S-Shaped have a median below the threshold, but indeed show variation. Musa-Okumoto is the first in 60% of datasets. In OSS the situation is completely different, only Gompertz and Inflection fit all data sets and have TS below 0.1.

In summary:

- On industrial data sets (System test and field data) accuracy is slightly better for Musa Okumoto and Inflection S Shaped, but all models have variations.
- On OSS data sets only Gompertz and Inflection fit all data sets and provide nearly always TS below 0.1.

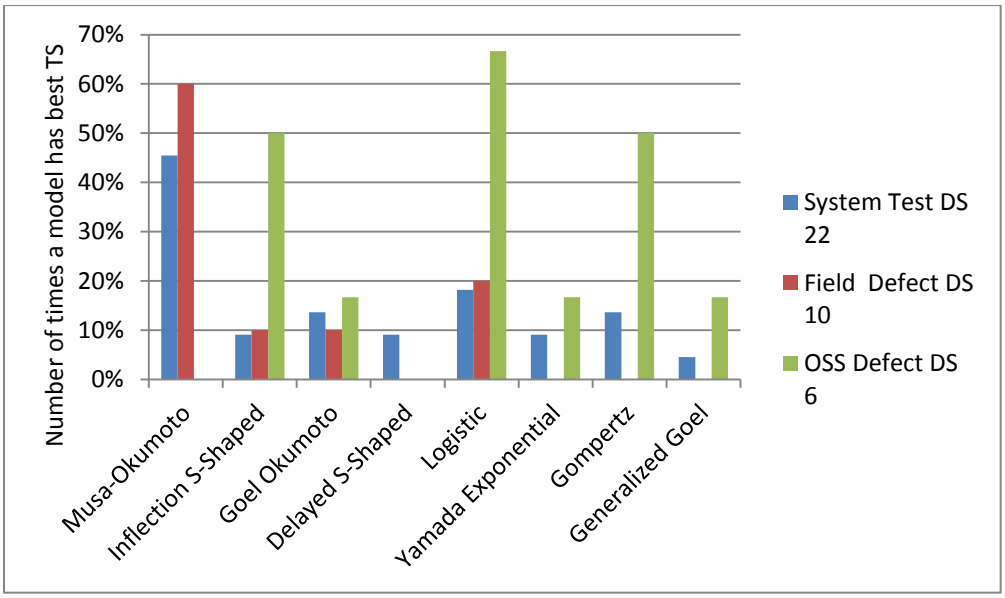

Figure 3-4: Ranking on best prediction: TS

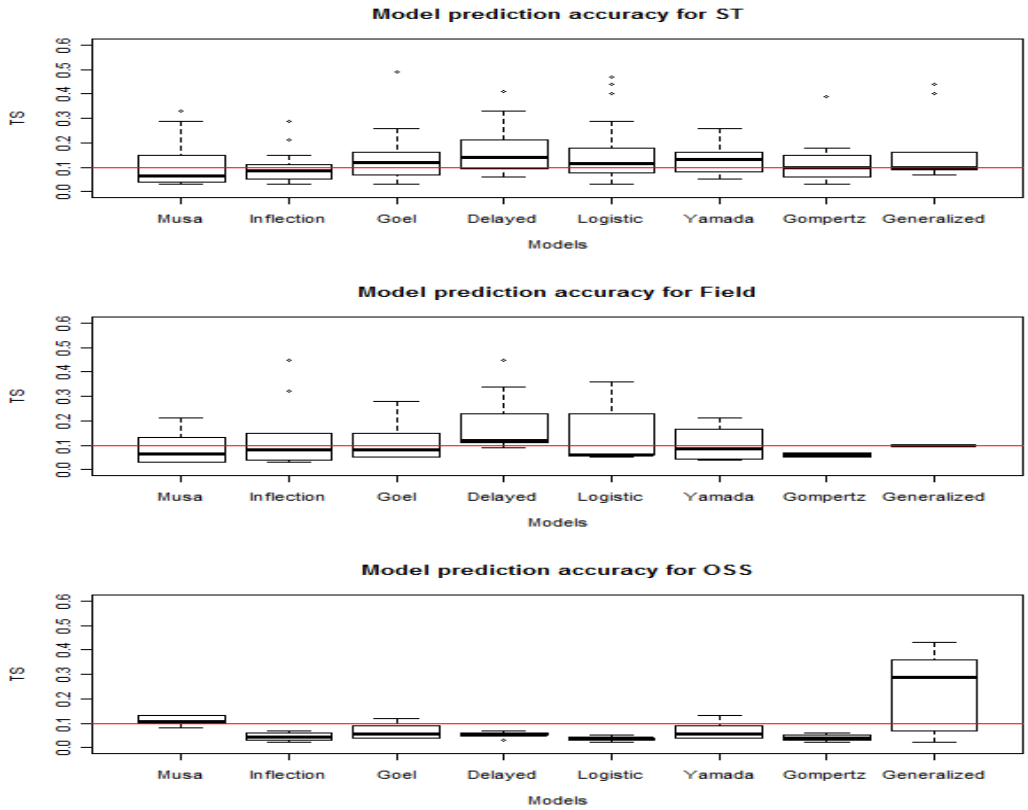

Figure 3-5: Box Plots of Prediction Accuracy (TS) values

Correctness

Correctness results are shown in the boxplots of Figure 3.7. The red lines represent the range ±10% of total number of actual defects.

In System test only Musa Okumoto, Inflection S-Shaped, Goel Okumoto and Yamada Exponential are reasonably within the range ±10% (but Yamada fits less datasets). Worse, all the others tend to underestimate the actual number of faults.

In field data, the same first three models (Musa Okumoto, Inflection S-Shaped, and Goel Okumoto) are reasonably within range. Gompertz is fully within range, but fits only 20% of data sets.

Finally, in OSS, only Gompertz and Yamada fit all models and are within range.

Looking at ranks in the bar chart diagram in Figure 3.6, we observe that Musa Okumoto is the best model in terms of correctness in the majority of both system test and field defects dataset. Inflection S-Shaped is directly behind Musa Okumoto in the case of system test data sets while Logistic is directly behind this in the case of field defect data sets. On contrary in the case of OSS Logistic and Delayed S-Shaped are the best ones.

In summary:

- On industrial data sets, Musa Okumoto, Inflection S-Shaped and Goel Okumoto fit most datasets and provide good accuracy and prediction.
- On OSS data sets Gompertz and Yamada fit all data sets and provide optimal accuracy and prediction.

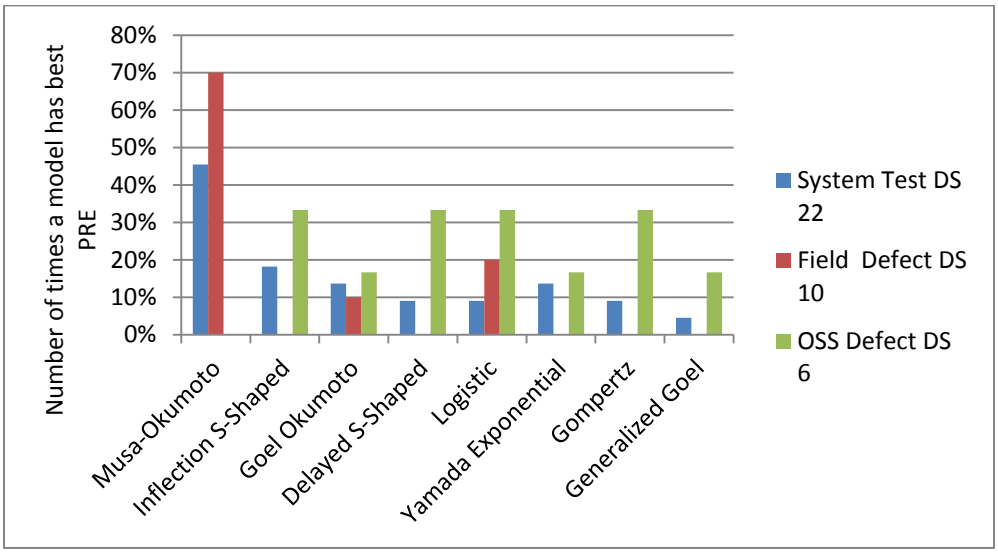

Figure 3-6: Ranking on best prediction: PRE

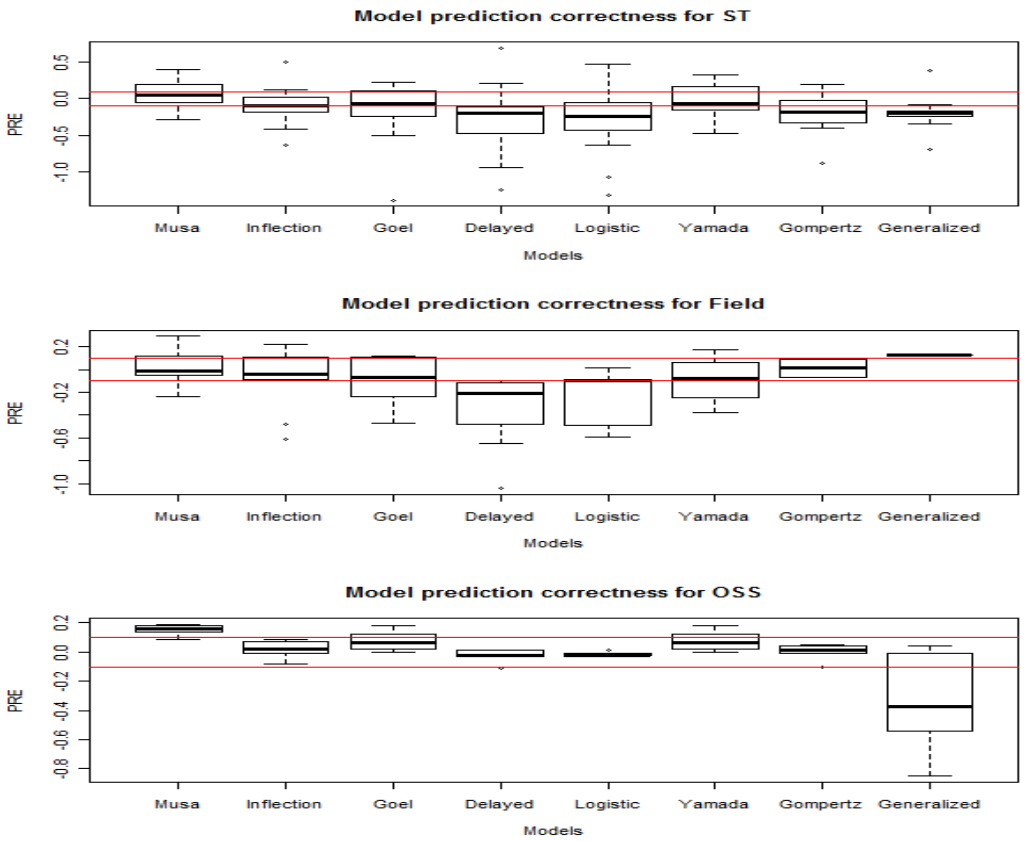

Figure 3-7: Box Plots of Prediction Correctness (PRE) values

3.3.3 RQ3: A model with good fit is also a good predictor?

Here we fit models on two thirds of the data sets and we analyze if the ones with good R^2 are also good predictors. Table 3.2 reports models and data sets. A model is described by three cells per category of dataset. The first contains the number of times a model fits the dataset, with any R^2 (information is also in Figure 1), the second shows how many times the model fits a data set with R^2 better than 0.9 and predicts with TS < 0.1, the third shows how many times the model fits a data set with R^2 better than 0.9 and predicts with PRE within 10%. . We observe from Table 3.2 that:

- On industrial datasets, Musa and Inflection are the ones with better prediction capability – however this happens only in a bit more than half the datasets.

- On OSS datasets Gompertz and Inflection have good prediction capability for all datasets, followed by Logistic and Delayed.

Table 3-2: Fitting and prediction capability of models

	System test DS			Field DS			OSS DS		
Model	Fitted DS	R^2 >= 0.9 AND TS <0.1	R^2 >= 0.9 AND PRE within ± 0.1	Fitted DS	R^2 >= 0.9 AND TS <0.1	R^2 >= 0.9 AND PRE within ± 0.1	Fitted DS	R^2 >= 0.9 AND TS <0.1	R^2 >= 0.9 AND PRE within ± 0.1
Musa	22/22	13/22	12/22	10/10	7/10	5/10	6/6	3/6	1/6
Inflection	22/22	15/22	10/22	9/10	6/10	4/10	6/6	6/6	6/6
Goel	21/22	7/22	6/22	8/10	6/10	3/10	6/6	5/6	4/6
Delayed	20/22	6/22	3/22	9/10	1/10	2/10	6/6	6/6	5/6
Logistic	20/22	7/22	5/22	9/10	5/10	5/10	5/6	5/6	5/6
Yamada	18/22	3/22	4/22	5/10	2/10	1/10	6/6	5/6	4/6
Gompertz	14/22	6/22	3/22	2/10	2/10	2/10	6/6	6/6	6/6
Generalized	10/22	2/22	1/22	1/10	1/10	0/10	6/6	4/6	2/6

3.4 Discussion

We have attempted to derive general conclusion about best fitter and best predicting SRGM model applying eight different models to a wide range of datasets on failures.

Considering the three RQs the origin of datasets appears to be a factor. The performance of models differs slightly between System test and Field test datasets. On the other hand there is a clear difference between OSS and industrial data sets. The best performer models are Musa Okumoto and Inflection for industrial datasets, while Gompertz is the best for OSS datasets. This result should be inquired more in the future, especially because we consider only 6 OSS datasets.

Considering the dichotomy between fitting and prediction, In general a good fitting model is not always also a good predictor, especially on industrial datasets. Musa Okumoto and Inflection fit nearly all datasets, but are good predictors only on a bit more than 50% of the datasets. The situation changes for OSS datasets, but again the number of datasets we have is very small.

Results also show that models that have good performances in fitting and predicting in system tests are still good in fitting and predicting field defects. We think this is a very practical and important finding because quality and maintenance managers might choose only one model regardless of the phase of the software lifecycle data come from.

Results are quite different for the six open source data sets, where the Gompertz model obtained the best results, followed by Inflection. These results show that SRGM models also apply on OSS. However, given the low number of datasets, we need further research to generalize this finding.

It is interesting to notice that Musa Okumoto, that is the best model for industrial datasets, does not apply in OSS. However,

OSS datasets are significantly different from industrial datasets (there is no distinction between System Test failures and Field defects), so we cannot really derive any meaningful consequence.

We finalize this discussion with a message to the maintenance and quality manager that might read this analysis. We provided ranks and boxplots to show to the readers that certain models got good fitting\prediction performances in several datasets. Although the high number of datasets used (38) might make our findings generalizable, we strongly suggest the reader to define her own thresholds for fitting, accuracy and correctness of predictions and re elaborate the results according to those thresholds, using the boxplot provided. In fact we are convinced that each context has unique combination of characteristics that make some thresholds more appropriate than others, thus the choice of the SRGM should be based on characteristics of the context and the data it is going to be applied.

3.5 Threats to validity

We recognize a first conclusion threat in the methodology we used to answer research questions, because we did not apply hypothesis testing due to the low cardinality of some datasets, and answers are given more in a qualitative way rather than in a quantitative one. Moreover the choice of threshold is also not grounded in the literature.

This approach was conscious and deliberate though. We preferred to choose indicative thresholds and show all boxplots to not bias the reader: we strongly believe that classifying a model as good or bad is a task that strictly depends on the level of performances that the manager wants to achieve. Each reader might decide by herself whether the threshold we used is appropriate for her context or not, and classify differently the models by looking at the boxplots. However, we did not leave this threat uncontrolled and we provided the reader with ranks to identify the best model for each type of datasets and metric.

We observe a construct threat in the impossibility to adequately compare industrial datasets with OSS datasets due to the lack of information on the defect detection phase

(system test, operation) in latter one: we could not build a proper control strategy except avoiding a structured comparison between industrial and OSS results.

We notice a conclusion threat in the choice of not performing cross validation in prediction. However we grounded our choice in the literature.

Finally, an external threat is the low number of datasets in for field defects and OSS datasets. However Apache and GNOME projects both have large and well organized communities, where a great number of developers contribute to the projects. The large sizes of these two projects make them the state-of-the-art in terms of management of OSS projects.

3.6 Conclusions

We have studied selected SRGM in generalized way for the purpose to derive general conclusion.

We found that the Musa-Okumoto model is the best one in fitting and predictions in industrial datasets. Although also Inflection S-Shaped achieved very good results with respect to the metrics thresholds we adopted. The Musa-Okumoto model did not hold the same performances in OSS data, in which the Gompertz model applied better, followed by Inflection S-Shaped.

We also observed two other interesting facts: 1) models which have good performances with system test data sets also good performances with field defect data, and 2) models that fit very well system test data not always predict with same performances. The practical consequences and recommendations to quality and maintenance managers are: 1) choose only one model regardless of the phase of the software lifecycle, 2) identify and choose a model that is flexible enough in case the quality process is under definition or in generable susceptible of important changes.

Our future work will be devoted to the extension of the datasets used to increase the generalizability of these findings, especially in OSS projects where results and structure of datasets were very different from industrial ones.

4. Reliability growth of open versus closed source software[3]

Herein this chapter we focus on the second research gap: applicability of SRGM for OSS is unclear. As clear from the literature review section (i.e. Chapter 2) different studies report different results for the applicability of software reliability models in OSS projects reliability characterization. To overcome this research gap we empirically compare the reliability growth of OSS with that of CSS in order to investigate whether or not OSS reliability grows in a different way compared to CSS.

4.1 Goal

To achieve the goal, we analyze these research questions, which are presented in detail:

4.1.1 R.Q1: Does OSS behave, from a reliability point of view, in the same way as closed source software?

Or, in operational terms, the OSS failure occurrence trend can be represented by SRGM such like CSS? Models are fitted on the failure datasets of OSS and CSS, and their R^2 (i.e. goodness of fit) are analysed and compared. Model fitting is required to estimate the parameters of the models and produce a prediction of failures. Fitting can be done using Linear or Non Linear Regression (NLR). We use NLR fitting due to the nature of data.

We use a commercial program for curve fitting of the models to the collected defect datasets. In case of convergence of the curve fitting we use goodness of fit (GOF) test, R^2 in order to determine how well curve fit to the data.

We consider a good fit when $R^2 > 0.90$. This threshold categorizes the models as good and bad for OSS and CSS projects in term of fitting capability. We also rank models on their R^2 in order to determine models that best fit to OSS and CSS.

4.1.2 R.Q2: Which models should be used (for OSS)?

Or in operational terms, which models have best TS (for prediction accuracy) and PRE (for prediction correctness). We use the partial failure history of the products to accomplish the prediction as [22]. The first two thirds data points of the each datasets following [50], is used to estimate the parameters. These estimated values of the parameters are then applied to the entire time span for which failure data is collected in each dataset in order to compare the prediction qualities of the models.

[3] **N. Ullah, M. Morisio . An Empirical Study of Reliability Growth of Open versus Closed Source Software through Software Reliability Growth Models. In: The 19th Asia-Pacific Software Engineering Conference, Hong Kong, December 4-7, 2012.**

Prediction capability can be evaluated under two points of view, accuracy and correctness. Accuracy deals with the difference between estimated and actual over a time period. Correctness deals with the difference between predicted and actual at a specific point in time (e.g. release date). A model can be accurate but not correct and vice versa. For this reason we use the Theil's Statistic (TS) for accuracy and Predicted Relative Error (PRE) for correctness.

4.2 Studied projects

The goal is to analyze the reliability growth of OSS versus software developed in-house. Our focus is to derive more generalize results using as many software projects as possible. For this purpose we used the failure data collected from the literature described in chapter 3 (for details see chapter 3). We found 22 datasets of CSS but a little number, i.e. 6 datasets, about OSS projects from literature. We then extracted defect data about some other OSS projects from Apache repository.

We identified two notable and active open source projects from apache.org (https://issues.apache.org/). These projects are C++ Standard Library and JUDDI. The Apache C++ Standard Library provides a free implementation of the ISO/IEC 14882 international standard for C++ that enables source code portability and consistent behavior of programs across all major hardware implementations, operating systems, and compilers, open source and commercial alike. JUDDI is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. Both of these projects are considered stable in production. The 66% of the reported issues in first project have been fixed while in the second project 95% of the reported issues have fixed and closed. We collected defect data of the selected projects from apache.org using JIRA. JIRA is a commercial issue tracker. Issues can be bugs, feature requests, improvements, or tasks. JIRA track bugs and tasks, link issues to related source code, plan agile development, monitor activity, report on project status.

For each version we have collected all the issues reported at our date of observation together with the date at which they were reported (date of opening). For each open source project, we have considered all the major versions until April 2012. For C++ Standard Library we were able to get eight (8) versions. Unfortunately, JUDDI had not so many reports and versions we had to limit the versions to four (4) until October 2011. Table 4.1 lists the information of the projects.

After a deep inspection of the repositories and of their documentation, we have decided to focus on those issues that were declared "bug" or "defect" excluding "enhancement," "feature-request," "task" or "patch". For the same reason, we have considered only those

issues that were reported as closed or resolved after the release date of each version. Further, we excluded issues closed before the release date. These issues are typically found in the candidate (or testing) releases of projects.

Table 4-1: OSS Projects Details

Project	Version	Release Date
C++ Standard Library	V4.1.2	18/07/2005
	V4.1.3	30/01/2006
	V4.1.4	030/7/2006
	V4.2.0	29/10/2007
	V4.2.1	01/05/2008
	V4.2.2	30/06/2008
	V4.2.3	01/09/2008
	V5.0.0	31/05/2009
JUDDI	V0.9	14/06/2005
	V2.0	02/08/2009
	V3.0	26/10/2009
	V3.1.0	27/06/2011

4.3 Results

4.3.1 Models Fitting Results: (RQ1)

First we consider the basic capability of the models to fit the datasets (fits or not), irrespective of the goodness of fit (R^2). Table 4.2 reports models fitting results. A model is described by two cells per type of dataset. The first contains the number of times a model fits the dataset, with any R^2, the second shows how many times the model fits a data set with highest R^2 value among the entire fitted model to the given data set. Musa Okumoto and Inflection S-Shaped Models fitted to all data sets of in-house software. On contrary in the case of OSS beside these two models Goel Okumoto and Yamada Exponential also fitted to all data sets. Most models also fit, except Gompertz, Generalized Goel that fit poorly in datasets of software developed in-house.

Now let's analyse goodness of fit too. In table 4.2 the second cell of each model per category reports how many times a model is the one with best R2. For instance Musa Okumoto has the best R^2 on only 23% of closed source software (CSS) datasets while Gompertz has the best R^2 on 44% of OSS datasets. But apart from that there is no clear winner.

However, analysing the top performer only can be misleading, in case many models fit with a similar R2 the same dataset. Therefore in Figure 4.1 we report the boxplots of R^2 per model for both types of datasets. Musa Okumoto remains the best performer (no

outliers and narrow boxplot) for Closed Source Software (CSS) datasets. On contrary for OSS datasets Inflection and Gompertz remains the best performer. Next to it in the case of CSS datasets except Generalized the other models have also narrow boxplot (always better than 0.9, the threshold depicted as a red horizontal line) but some outliers. Whereas in the case of OSS datasets the models fitting boxplots are so narrow as like CSS datasets which clearly indicates that OSS failure occurrence pattern is not too similar to that of CSS failure occurrence pattern.

In summary:

- Considering plain fitting (table 4.2) Musa and Inflection fit all CSS datasets while along with Musa and Inflection, Yamada exponential and Goel Okumoto fit all OSS datasets. The others fit at least 80% of the data sets.

- Considering the R^2 of models that fit the datasets (Figure 4.1). All the models have very good fit (better than 0.9), except Generalized Goel but with outliers in the case of CSS while in OSS the median of Musa, Goel and Yamada are on the threshold.

Table 4-2: Results of the Models fitted to each type of Dataset

	CSS DS		OSS DS	
Model	Fitted DS	# of DS having Highest R^2 Value	Fitted DS	# of DS having Highest R^2 Value
Musa	22/22	5/22	18/18	0/18
Inflection	22/22	4/22	18/18	5/18
Goel	21/22	0/22	18/18	0/18
Delayed	20/22	2/22	16/18	1/18
Logistic	20/22	3/22	15/18	1/18
Yamada	18/22	0/22	18/18	0/18
Gompertz	14/22	4/22	17/18	8/18
Generalized	10/22	4/22	16/18	3/18

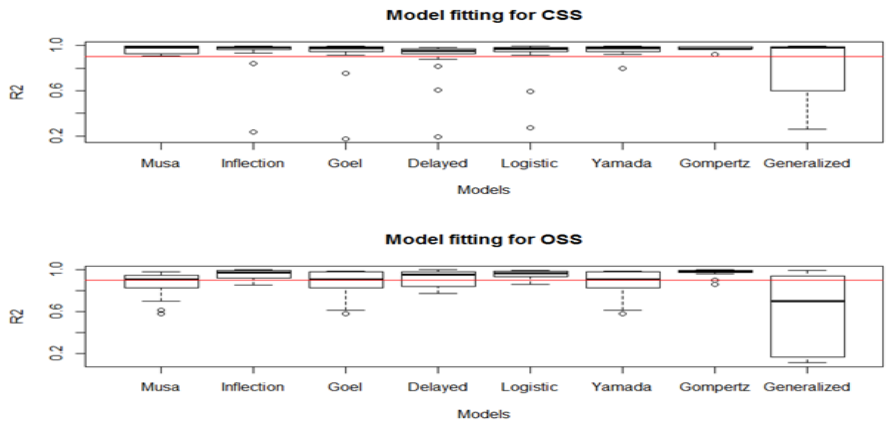

Figure 4-1: Box Plots of fitting (R^2) values

4.3.2 Models Prediction Results: (RQ2)

For models that could fit the data set we used the first two-third data points of the data sets to train the model, and predicted the last third. We analyse their predicting capability in terms of accuracy and precision.

Accuracy

Table 4.3 reports models prediction results. A model is described by two cells per category of dataset. The first contains the number of times a model is the best predictor in terms of TS, the second contains the number of times a model is the best predictor in terms of PRE value among the entire models for a given data set. It is clear that Musa Okumoto outperforms the others in the CSS datasets. On the contrary in the case of OSS Gompertz outperforms the others. We got several ties (this explains why the sum might be greater than total number of datasets).

Figure 4.2 reports the TS values for all datasets of both types. The red line represents the 0.1 threshold, usually considered indicator of good accuracy. Figure 4.2 allows us to discuss good models, instead of best model as in table 4.3.

In CSS data sets, Musa Okumoto and Inflection S-Shaped have a median below the 0.1 threshold. This happens also for Gompertz and the Generalized Goel models; however they do not fit in several cases (table 4.2).

In OSS data sets only Gompertz have a median below 0.1 and Inflection S-Shaped have a median close to 0.1. These two models also outperform in fitting as in table 4.2.

In summary:

- On CSS data sets accuracy is better for Musa Okumoto and Inflection S Shaped, but all models are close to the threshold.

- On OSS data sets only Gompertz and Inflection fit all data sets and provide median below 0.1, but with some variation. All other models have variations.

Correctness

Correctness results are shown in the boxplots of Figure 4.3. The red lines represent the range ±10% of total number of actual defects.

In CSS datasets only Musa Okumoto, Inflection S-Shaped, Goel Okumoto and Yamada Exponential are reasonably within the range ±10% (but Yamada fits less datasets). Worse, all the others tend to underestimate the actual number of faults.

In OSS datasets Inflection S-Shaped and Gompertz models are within the range but with outliers. All the others tend to underestimate the actual number of faults.

In summary:

- On CSS data sets, Musa Okumoto, Inflection S-Shaped and Goel Okumoto fit most datasets and provide good accuracy and prediction.
- On OSS data sets Musa. Inflection, Goel, Yamada and Gompertz fit all data sets and provide optimal accuracy and prediction.

Table 4-3: Models Predictions Results

	CSS DS		OSS DS	
Model	No of DS having lowest TS Value	No of DS having Lowest PRE Value	No of DS having lowest TS Value	No of DS having Lowest PRE Value
Musa	10/22	10/22	4/18	2/18
Inflection	2/22	4/22	3/18	3/18
Goel	3/22	3/22	1/18	3/18
Delayed	2/22	2/22	4/18	3/18
Logistic	4/22	2/22	5/18	3/18
Yamada	2/22	3/22	1/18	3/18
Gompertz	3/22	2/22	6/18	6/18
Generalized	1/22	1/22	3/18	2/18

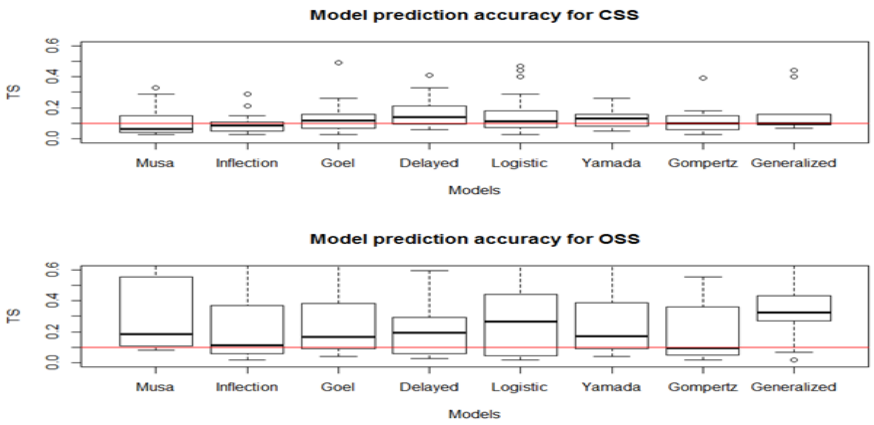

Figure 4-2: Box Plots of Prediction Accuracy (TS) values

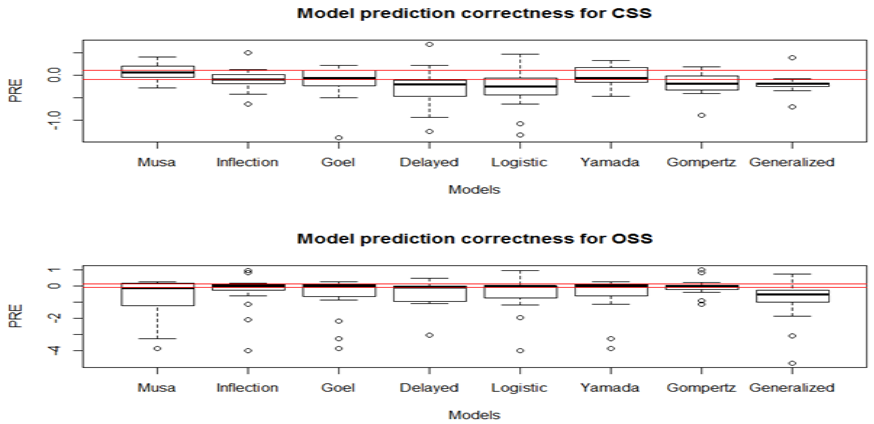

Figure 4-3: Box Plots of Prediction Correctness (PRE) values

4.4 Discussion

We have attempted to derive general conclusion about the reliability growth of OSS versus CSS applying eight different SRGM models to a wide range of datasets on failures.

The performance of models differs slightly between CSS and OSS datasets. The results show a slightly difference between failure occurrence pattern of CSS and OSS, which indicates a slightly difference in the reliability growth of the OSS from CSS. However the results show that SRGM can be used for the reliability characterization OSS. We observed different behavior of the models for OSS as compared to CSS. The best performer models are Musa Okumoto and Inflection for industrial datasets, while Gompertz and Inflection are the best for OSS datasets. This result should be inquired

more in the future, especially because we consider failure datasets of different versions of only 4 OSS projects.

These results show that SRGM models also apply on OSS. However, we need further research to generalize this finding.

It is interesting to notice that Musa Okumoto, that is the best model for CSS datasets, does not apply in OSS. However, OSS datasets are significantly different from CSS datasets (there is no distinction between System Test failures and Field defects), so we cannot really derive any meaningful consequence.

4.5 Threats to validity

We recognize a first conclusion is the choice of threshold is not grounded in the literature.

We observe a construct threat is the impossibility to adequately compare industrial datasets with OSS datasets due to the lack of information on the defect detection phase (system test, operation) in latter one: we could not build a proper control strategy except avoiding a structured comparison between industrial and OSS results.

We notice a conclusion threat in the choice of not performing cross validation in prediction. However we grounded our choice in the literature.

4.6 Conclusions

We have studied reliability growth of OSS versus CSS using SRGM in generalized way for the purpose to derive generalize conclusion.

We observed from the model fitting results that there is a slightly difference between failure occurrence pattern of OSS and CSS. This a clear indication that OSS reliability grow slightly in a different way as compared to CSS. This is because of the fact of changing the code frequently. Analyzing reliability growth of OSS versus CSS applying SRGM provided us the result of best model for CSS and OSS projects. We found that the Musa-Okumoto model is the best one in fitting and predictions in CSS datasets. Although also Inflection S-Shaped achieved very good results with respect to the metrics thresholds we adopted. The Musa-Okumoto model did not hold the same performances in OSS datasets, in which the Gompertz, Inflection S-Shaped and Goel Okumoto models applied better.

Our future work will be devoted to the extension of the datasets used to increase the generalizability of these findings, especially in OSS projects where results and structure of datasets were very different from industrial ones.

5. Factors affecting open source software reliability growth[4]

Herein focusing on the second research gap, we discuss the factor that affects the reliability growth of OSS projects. In OSS projects defects detection and fixing time for a defect is quite different from each other, which may affect the reliability growth. This difference in defect detecting and fixing time may be a reason of unclear results reported in literature regarding the applicability of SRGM for reliability characterization of OSS. We empirically analyze the reliability growth of OSS with respect to defect detecting time versus defect fixing time through SRGM. We therefore investigate the research question:

5.1 Goal

5.1.1 R.Q: Does defect detection and fixing time affect the reliability of an OSS project?

In order to achieve the goal we focus on these research questions, which are presented in detail:

5.1.1.1 R.Q1: Does defect detection and fixing time affect the SRGMs' fitting capabilities for an OSS project?

Or, in operational terms, the OSS defect occurrence trend can be represented by SRGM in a similar way such like OSS defect fixing trend? Models are fitted to the defects datasets collected with respect to defect detection date (DD DS) and to the defects datasets collected with respect to defect fixing date (DF DS), and their R^2 are analysed and compared by adopting visual analysis and statistical hypothesis testing. Model fitting is required to estimate the parameters of the models and produce a prediction of failures. We use NLR for model fitting due to the nature of data. We use goodness of fit (GOF) test, R2 to determine how well curve fit to the data. R^2 takes a value between 0 and 1, inclusive. The closer the R^2 value is to one, the better the fit. The R^2-value is used for its simplicity and is motivated by the work of Gaudoin, O. et al [52], who evaluated the power of several statistical tests for GOF for a variety of reliability models. Their evaluation showed that this measure was as least as powerful as the other GOF tests analyzed.

For the purpose of visual representation, we use box plots: as they allow for an immediate comparison. We consider a good fit when $R^2 > 0.90$ because the model fit

[4] **The content of this chapter have been modelled according to N. Ullah, M. Morisio .An Empirical analysis of Open Source Software Defect data through Software Reliability Growth Models. In: IEEE EUROCON 2013 in Zagreb, Croatia, 1-4 July 2013.**

might be considered good having R^2 = 0.90. This threshold categorizes the models as good and bad in term of fitting capability. We also do hypothesis testing on the R^2 of fitted models in order to determine statistical significant difference in models fitting values for both types of datasets. Therefore we formulate null and alternative hypothesis as follows.

$H0_0$: The SRGM models' fitting capabilities for OSS are not affected with defect detecting and fixing time (i.e. R^2 of models fitted to DD DS is not different from R^2 of models fitted to DF DS).

H0a: The SRGM models' fitting capabilities for OSS are affected with defect detecting and fixing time (i.e. R^2 of models fitted to DD DS is better than R^2 of models fitted to DF DS).

According to the recommendations in [53] we use the Mann-Whitney test in order to evaluate practical differences in models' fitting capabilities for both types of datasets. The assumption to select was the not normal distribution of datasets comprising of R^2 values of fitted models. In the statistical testing, the significance level is checked by the given p-value. For rejecting or accepting the null hypothesis, we used the significance value α=5%.

5.1.1.2 R.Q2: Does defect detection and fixing time affect SRGMs' prediction qualities for an OSS project?

Or in operational terms, the models prediction accuracy and correctness do not change with respect to defect detection and fixing time. We use the partial failure history (i.e. first portion of the collected defect datasets are used for model fitting and remaining portion of the datasets are used for prediction) of the products to accomplish the prediction as [22]. The first two thirds data points of the each datasets following [50], is used to estimate the parameters. These estimated values of the parameters are then applied to the entire time span for which failure data is collected in each dataset in order to compare the prediction qualities of the models for both types of defect datasets.

Prediction capability can be evaluated under two points of view, accuracy and correctness. Accuracy deals with the difference between estimated and actual over a time period. Correctness deals with the difference between predicted and actual at a specific point in time (e.g. release date). A model can be accurate but not correct and vice versa. For this reason we use the Theil's Statistic (TS) for accuracy and Predicted Relative Error (PRE) for correctness.

Similar to models fitting (i.e. R^2), models prediction accuracy and correctness are visually represented through boxplots. We consider a prediction as good if TS is below 10% and PRE is within the range [-10%, +10%] of total number of actual defects because 10% range might be acceptable. These thresholds of TS and PRE categorize the models as good and bad in term of prediction accuracy and correctness. We also do hypothesis testing on the TS and PRE of fitted models in order to determine statistical significant difference in models prediction qualities for both types of datasets. Therefore we formulate null and alternative hypotheses as follows.

$H1_0$: The SRGM models' prediction qualities for OSS are not affected with defect detecting and fixing time (i.e. TS and PRE of models prediction for DD DS is not different from TS and PRE of models prediction for DF DS).

H1a: The SRGM models' prediction qualities for OSS are affected with defect detecting and fixing time (i.e. TS and PRE of models prediction for DD DS is better than TS and PRE of models prediction for DF DS).

Similar to methodology adopted for RQ1, we use the Mann-Whitney test in order to evaluate practical differences in models' prediction qualities for both types of datasets. The assumption to select was the not normal distribution of datasets comprising of TS and PRE values of fitted models. In the statistical testing, the significance level is checked by the given p-value. For rejecting or accepting the null hypothesis, we used the significance value α=5%.

5.2 Data collection

In OSS projects defects detection and fixing time for a defect is quite different from each other. We therefore investigate the reliability growth of OSS with respect to defect detection time versus defect fixing time. We identified five notable and active open source projects from apache.org (https://issues.apache.org/). These projects are C++ Standard Library, JUDDI, HTTP Server, XML Beans, and Enterprise Social Messaging Environment (ESME). The Apache C++ Standard Library provides a free implementation of the ISO/IEC 14882 international standard for C++ that enables source code portability and consistent behavior of programs across all major hardware implementations, operating systems, and compilers, open source and commercial alike. JUDDI is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. The Apache HTTP Server is an open-source HTTP server for modern operating systems including UNIX, Microsoft Windows, Mac OS/X and Netware. XML Beans is a tool that allows you to access the full power of XML in a Java friendly way. ESME (Enterprise Social Messaging Environment) is a secure and highly scalable micro sharing and micro messaging platform that allows people to discover and meet one another and get controlled access to other sources of

information, all in a business process context. All these projects are considered stable in production. The 66%, 95%, 68%, 64% and 82% of the reported issues in these projects respectively, have been fixed and closed. We collected defect data of the selected projects from apache.org using JIRA. JIRA is a commercial issue tracker. Issues can be bugs, feature requests, improvements, or tasks. JIRA track bugs and tasks, link issues to related source code, plan agile development, monitor activity, report on project status.

For each release of the selected projects we have collected all the issues reported at our date of observation. For each project, we have considered all the major releases until October 2012. We were able to get eight (8) versions for C++ Standard Library, seven (7) versions for JUDDI, two (2) versions for HTTP Server, five (5) versions for XML Beans and three (3) versions for ESME. Hence defects data of 25 different releases of 5 projects were collected. Table 4.4 lists the information of the projects along with the selected releases and their time windows for each release.

The tracking software records all the information regarding each issue, such as *issue type*, *status*, *created date*, *updated date*, *affected version*. After a deep inspection of the repositories and of their documentation, we have decided to focus on those issues that were declared "bug" or "defect" excluding "enhancement," "feature-request," "task" or "patch". For the same reason, we have considered only those issues that were reported as closed or resolved after the release date of each version. Further, we excluded issues closed before the release date. These issues are typically found in the candidate (or testing) releases of projects. We filtered all the issues in order to collect only issues that have declared "defect" or "bug" as in [21, 23]. For the filtration of the collected issue from the online repository we used the aforementioned attributes. After refining the data we grouped the defects into cumulative defects by week.

We developed two types of datasets for each release of each project. In first type of datasets (i.e. created date DS) we grouped the defects into cumulative defects by weeks with respect to created date of the defects while in second type of dataset (i.e. updated date DS) we grouped the defects into cumulative defects by weeks with respect to updated date of the defects. We divided the entire time span of each release into weeks and then counted detected defects in each week. For each release in first type of dataset we counted defects for each week with respect to created date (after this will call created date DS) and in second type of dataset we counted defects for each week with respect to updated date (after this will call updated date DS). In this way we developed 25 created date DS and 25 updated date DS for total of 25 selected releases of the five OSS projects.

Table 5-1: Selected Projects Details

Project	Version	Release Date
C++ Standard Library	V4.1.2	18/07/2005
	V4.1.3	30/01/2006
	V4.1.4	03/07/2006
	V4.2.0	29/10/2007
	V4.2.1	01/05/2008
	V4.2.2	30/06/2008
	V4.2.3	01/09/2008
	V5.0.0	31/05/2009
JUDDI	V2.0	02/08/2009
	V3.0	26/10/2009
	V3.0.1	01/02/2010
	V3.0.2	17/05/2010
	V3.0.3	22/07/2010
	V3.0.4	06/11/2010
	V3.1.0	27/06/2011
HTTP Server	V3.1.4	13/02/2005
	V3.2.7	13/02/2006
XMLBeans	V2.0	30/06/2005
	V2.1	16/11/2005
	V2.2	23/03/2006
	V2.3	01/06/2007
	V2.4	08/07/2008
ESME	V1.1	09/10/2010
	V1.2	14/03/2011
	V1.3	29/08/2011

5.3 Results

5.3.1 Models Fitting Results: (RQ1)

In Figure 4.4 we report the boxplots of R^2 (i.e. Goodness of Fit values) per model for both types of datasets of each release of the selected projects. For RQ1, observing the box plots in Figure 4.4 it appears that there is clear difference. Medians of all the models are above the threshold in case of created date DS and all the models have also narrow boxplot (always better than 0.9, the threshold depicted as a red horizontal line) but some outliers. On contrary in case of updated date DS the boxplots of R^2 values show clear variation. It is clear from the Figure 4.4 that models fitting capabilities increase in case of created date DS. Hence it is suggested that for the reliability characterization of OSS through SRGMs defects created date should be considered.

We also test the hypothesis H00 with Mann-Whitney test for differences. The test reports a p-value = 0.0006344 which is below the threshold, α. Therefore, we reject the null hypothesis, indicating that there is significant difference of models fitting between the

defects created and updated date DS, which is also visually represented through boxplots in Figure 4.4.

In summary:

- All the models have very good fit (better than 0.9), but with outliers in the case of created date DS while in updated date DS only the median of Inflection, Logistic, Gompertz and Generalized are above the threshold.

- There is practical significant difference in models fitting capabilities when defects created date is used in developing OSS defects datasets for the reliability characterization.

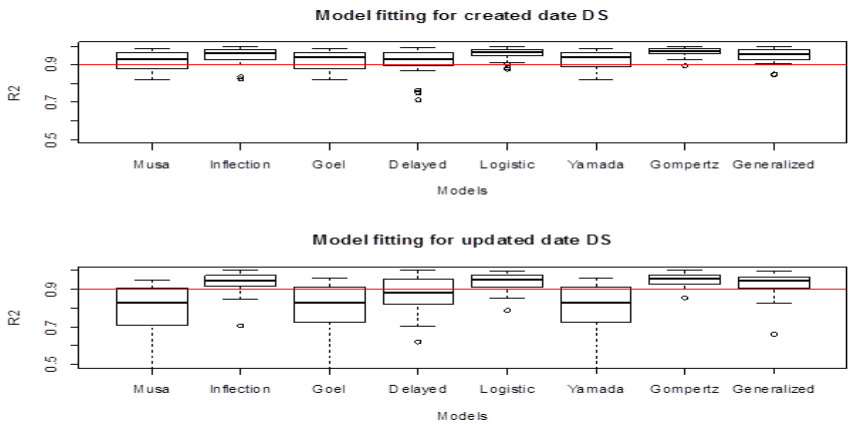

Figure 5-1: Box Plots of fitting (R^2) values

5.3.2 Models Prediction Results: (RQ2)

In order to analyze the models prediction qualities we used the first two-third data points of the data sets to train the model, and predicted the last third. The choice of two-third data points was motivated with the wood's suggestion for model stability [2]. We analyse the models prediction qualities in terms of prediction accuracy and correctness.

Accuracy

Figure 4.5 reports the TS values for all datasets of both types. The red line represents the 0.1 threshold, usually considered indicator of good accuracy.

In created date DS, all the models have very good prediction accuracy and have narrow boxplot (always the medians lie on the threshold 0.1, the threshold depicted as a red horizontal line). In updated date DS all the models have not good prediction accuracy and the boxplots show the variations in their prediction accuracy. It is clear from the Figure 4.5 that models prediction accuracy increase in case of created date DS. Hence it is suggested that for the reliability characterization of OSS through SRGMs defects created date should be considered.

We test the hypothesis $H1_0$ with Mann-Whitney test for differences. The test reports a p-value < 2.2e-16 for TS values of both types of datasets, which is below the threshold, α. Therefore, we reject the null hypothesis, indicating that there is significant difference of models prediction qualities in term of prediction accuracy between the defects created and updated date DS, which is also visually represented through boxplots in Figure 4.5.

In summary:

- In created date DS all models are close to the threshold while in updated date DS all other models have variations.
- There is practical significant difference in models prediction accuracy when defects created date is used in developing OSS defects dataset for the reliability characterization.

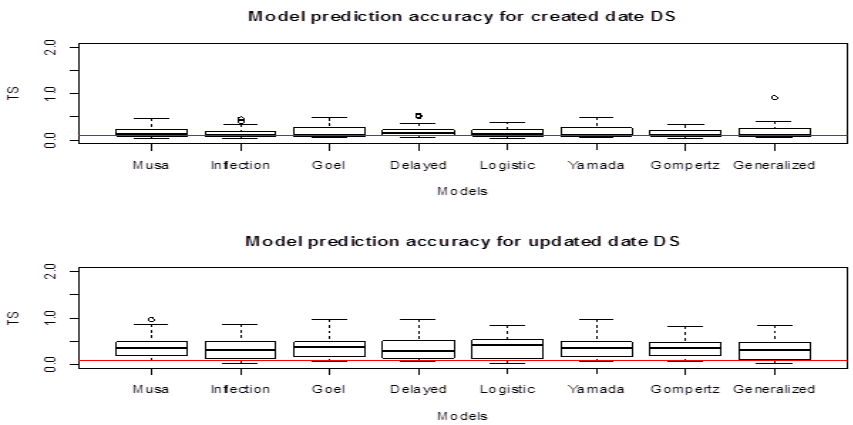

Figure 5-2: Box Plots of Prediction Accuracy (TS) values

Correctness

Correctness results are shown in the boxplots of Figure 4.6. The red lines represent the range ±10% of total number of actual defects.

In created date DS all models have narrow boxplots and their medians lie within the range ±10% of selected threshold. On contrary in updated date DS all the models tend to underestimate the actual number of defects. Only inflection S-Shaped has median lies in the selected range. It is clear from the Figure 4.6 that models prediction correctness increase in case of created date DS. Hence it is suggested that for the reliability characterization of OSS through SRGMs defects created date should be considered.

We test the hypothesis $H1_0$ with Mann-Whitney test for differences. The test reports a p-value = 9.709e-05 for PRE values of both types of datasets, which is below the threshold, α. Therefore, we reject the null hypothesis, indicating that there is significant difference of models prediction qualities in term of prediction correctness between the defects created and updated date DS, which is also visually represented through boxplots in Figure 4.6.

In summary:

- On created date DS all the models provide good accuracy and prediction while for updated date DS all the models behave inversely.
- There is practical significant difference in models prediction accuracy when defects created date is used in developing OSS defects dataset for the reliability characterization.

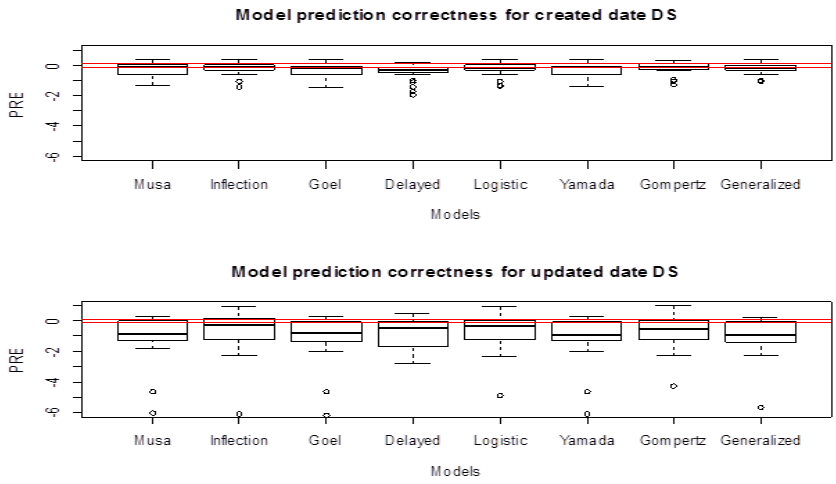

Figure 5-3: Box Plots of Prediction Correctness (PRE) values

5.4 Threats to validity

We recognize a first conclusion threat is the choice of threshold is not grounded in the literature. However we provided boxplots to show to the readers that certain models got good fitting\prediction performances in several datasets. Although the high number of datasets used (50) might make our findings generalizable, we strongly suggest the reader to define her own thresholds for fitting, accuracy and correctness of predictions and re elaborate the results according to those thresholds, using the boxplot provided. We notice another conclusion threat in the choice of not performing cross validation in prediction. However we grounded our choice in the literature.

The number of release and the time windows of the observations are different in the five OSS. This was due to some time constraints and the availability of the data in the repositories. As we do not compare the five OSS, but we rather want to understand whether there is a pattern of reliability in each OSS, this difference is not crucial.

We used open on-line repository to collect data of five different projects. We intensively cleaned the data we collected to limit the bias associated with the open nature of these repositories.

5.5 Discussion and conclusion

We have attempted to derive general conclusion about the reliability growth of OSS applying eight different SRGM models to a wide range of OSS defects datasets. We evaluate the reliability growth pattern of OSS using SRGMs. The performance of models differs between created date and updated date datasets. The results show a huge

difference between failures occurrence patterns of created date DS and updated date DS, which indicates a clear cut difference in the reliability growth of the OSS with respect to defect creating date and defect fixing date. From the results of this study it is suggested that for reliability characterization of OSS defects created date should be considered because the reliability of OSS directly increases with defects created date. The results also show that SRGM can be used for the reliability characterization OSS and their fitting capabilities and prediction qualities directly related to defects creating date instead of defects updating/fixing date. This study makes the unclear results reported in the literature regarding the applicability of SRGMs for OSS reliability characterization, clearer.

In our previous studies [54, 55] we have observed different behavior of the best models for OSS as compared to CSS (Closed Source Software). The best performer models were Musa Okumoto and Inflection for industrial datasets, while Gompertz and Inflection were the best for OSS datasets. We therefore deeply investigate the models fitting and prediction results focusing on this observation. We observed that all the S-Shaped models fitting and prediction qualities for OSS is better than concave shaped that is why Musa is best former for CSS but not for OSS because of its concave nature. While Gompertz and Inflection belong to S-Shaped category and as such it indicates an initial learning phase in which the community of end-users and reviewers of the open source project does not react promptly to new release. So because of this S-Shaped nature Inflection S-Shaped and Gompertz outperformed for OSS than Musa Okumoto Model.

These results of this study show that SRGM models can characterize the OSS reliability growth. For reliability modelling of OSS the defect creating date should be used for developing defect data sets of OSS in order to characterize their reliability growth through software reliability models.

6. Method for predicting OSS residual defects[5]

Herein this chapter we focus on *third research gap*: No method available that can be used to evaluate the reliability of an OSS before its adoption. In order to support practitioners in characterizing the reliability (in terms of remaining or residual defects) of an OSS component or application an empirical method is needed to be developed that might be used for reliability evaluation of an OSS. The reliability of the OSS component/application is one of the factors for the decision of a project manager about using the OSS or not in a project. SRGMs can be used at this purpose.

6.1 Goal

We detail here the goal using the GQM template [51].

Object of the study	A method for selecting best SRGM
Purpose	to support practitioners in decision about the adoption of an OSS
Focus	characterizing OSS reliability in terms of remaining defects
Stakeholder	from the point of view of project managers
Context factors	in the context of OSS

We derive a research question (RQ) on the object of the study that completes the GQM.

RQ: Does the proposed method select the best (i.e. that predicts more precisely the number of residual defects) SRGM?

We describe our proposed method in detail in section 0. Section 0 shows the application of the method to twenty one different releases of seven OSS projects using eight SRGMs. Section 5 presents the validation of the method. Here in next section we give a quick refresher on reliability modeling and brief description about SRGMs that are used in this study.

[5] **The content of this chapter have been modelled according to N. Ullah, Morisio M, Vetro'.A. A method for selecting software reliability growth model to predict Open Source Software residual defect. In: IEEE Computer Journal. (In press)**

6.2 The proposed method

The method proposed aims to select the SRGM, which best predicts the residual defects of an OSS. The idea to develop the method has been inspired by the work of Stringfellow et al [34]. They have developed a method that selects the best model to help in decision making when to stop testing and deploy the software. The following points indicate the main risks while applying SRGMs for predicting the residual defects of an OSS:

- All model assumptions listed in the background section may not apply exactly to OSS development. This will result in models fitting capabilities that may not fit or may have low goodness of fit (GOF) value.
- There is a limited amount of defects data from OSSs. The smaller the amount of data the longer the time may take the models to stabilize, or even not to fit the data.

To handle these risks, our method uses several SRGMs and selects the models which best fit the data.

The method is defined by the following steps (see also **Error! Reference source not ound.**and **Error! Reference source not found.**)

1. The first step is to select the release of the OSS project of interest and collect the issues from the online repositories. There are many online repositories, such as sourceforge, apache, bugzilla etc, which contain issues and defects data of OSS projects.
2. The second step is to extract defects from the issues collected in step 1. Issues can be bugs, feature requests, improvements, or tasks. The issues need to be filtered in order to include only those issues that have been declared as a "bug" or a "defect" and exclude "enhancements," "feature-requests," "tasks" or "patches". Further, only defects that were reported as closed or resolved are considered, open or reopen defects are excluded. Finally, duplicate defects must be excluded too. The defects data of the whole release interval [0, T] are grouped into cumulative defects by weeks.
3. The third step is to apply the SRGMs (i.e. Musa, Inflection, Goel, Delayed, Logistic, Yamada, Gompertz and Generalized) to the defects data obtained from step 2. The models are fitted to the defect data of 3/4T; that is represented as 'model fitting window' in **Error! Reference source not found.**. Fitting is done sing Non Linear Regression (NLR) techniques. NLR is a general technique to fit a curve through the data. The parameters are estimated by minimizing the sum of the squares of the distances between the data points and the regression curve. NLR is an iterative process that starts with an initial estimated value for each

parameter. The iterative algorithm then gradually adjusts these parameters until they converge on the best fit so that the adjustments make virtually no difference in the sum-of-squares. If the model's parameters do not converge to the best fit, then the model cannot fit the data. If a curve can be fitted to the data for a model, the goodness of fit (GOF) is evaluated using the R^2-value. The R^2-value is used for its simplicity and is motivated by the work of Gaudoin, O. et al [52], who evaluated the power of several statistical tests for GOF for a variety of reliability models. Their evaluation showed that this measure was at least as powerful as the other GOF tests analyzed. The larger the R^2 value, the better the fitted model explains the variation in the data.

Model fitting estimates best fitted values for all parameters of the model. The value of parameter 'a' represents the expected total number of defects. For model fitting we used the commercial program *PRISM*.

4. In this step models are passed through model rejection criteria. First criterion is about GOF, $R^2 > 0.95$. Second criteria is estimated defects >= actual defects. Models that do not satisfy both criteria are rejected. Our choice for setting the GOF value threshold at 0.95 is motivated by the work of Stringfellow, et al [34]. If a model does not have a high GOF then the collected defects data is insufficient for reliability modeling and additional data is required. The second criterion is added to keep only 'safer' models.
5. In this step models are evaluated in term of prediction stability. A prediction is stable if the prediction for week *j* is within ±10% of the prediction for week *j-1*. Setting a threshold for stability is based on subjective judgment. One may require higher or lower values for this threshold. Our rationale for setting this threshold at 10% is motivated by Wood's suggestion of using 10% as a stability threshold [2]. If no model has a stable prediction that is within the stability threshold defined, this means that the collected defects data is insufficient for reliability modeling and additional data is required.

   To check model stability the time window from 3/4 T to T is used; that is represented as 'model stability check window' in **Error! Reference source not ound.**. For instance, the model stability is checked as follows: in cumulative defects of the 3/4T, add one week defect, i.e. 3/4T+1week, then fit all the models that have passed the rejection step, to cumulative defects of 3/4T+1weeks, the value of parameter 'a' is the prediction of the model for the expected total number of defects. Similarly another week is added, i.e. 3/4T+2weeks, and so on till T. A model is stable if its prediction for week *j* is within ±10% of the prediction for week *j-1*, for all weeks.

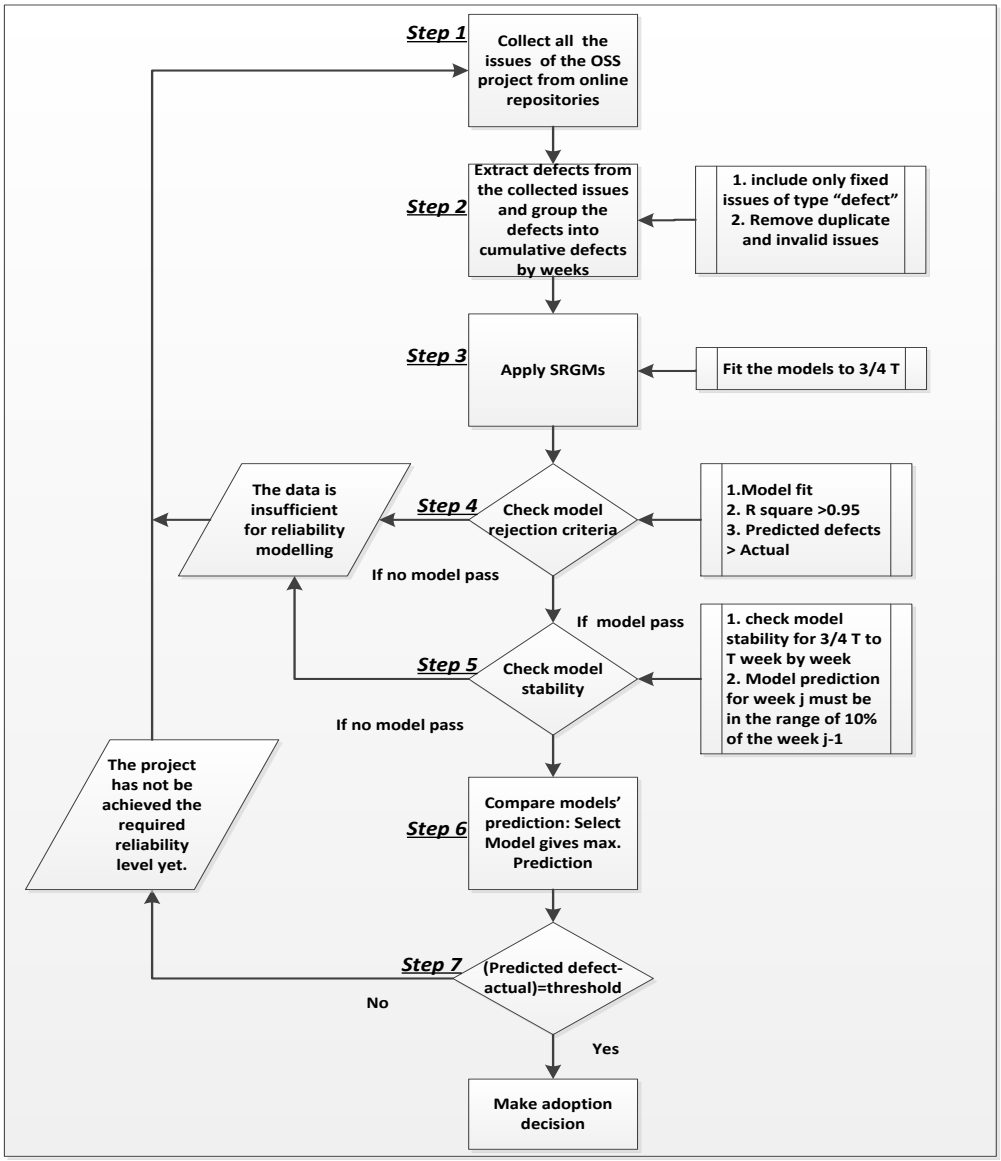

Figure 6-1: The proposed method: steps

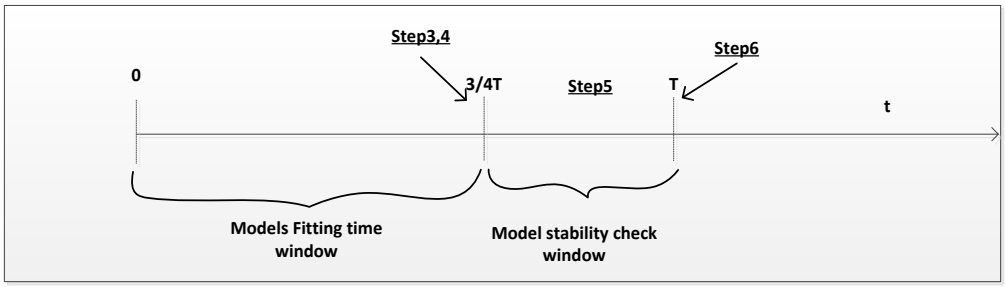

Figure 6-2: The proposed method: time frames

6. The sixth step is to select the best SRGM. The model which gives the highest number of predicted defects among all stable models is selected. It is a conservative choice, but the rationale is to select a safer model. In practice that model is considered suitable which overestimate the actual number of defects because defect fixing cost before adoption of an OSS would be lesser than the defects fixing cost after the adoption of the OSS. If there is a large difference in values for the prediction between different models, one may want to either augment this analysis with other quality assessment methods, as shown in [56], or choose a subset of models based on the GOF.
7. In this step using the selected SRGM the residual defects of the OSS are computed. In function of the number of residual defects the project manager may decide to adopt the OSS, wait some more time (i.e. wait for more defects to be found and fixed), adopt another OSS or closed source component.

6.3 Application of the Method

Here we apply our method to the selected OSS projects to show in practice how it works. Section 2.1 describes the OSS projects selected, and section 2.2 gives the results.

6.3.1 OSS projects selected

Thousands of OSS projects are undergoing development, so, it is important to select the most representative ones. We selected seven projects of a different nature having large and well organized communities. Among our selected seven OSS projects, the defect data of two projects, i.e. Apache and GNOME, were available in open literature. We took defect data about three different releases of Apache and three different releases of GNOME projects published by Li et al [24]. Both Apache and Gnome have a great number of developers having the right to update and change files freely. The large sizes of these two projects make them the state-of-the-art in terms of management of OSS projects. Apache 2.0.35 has been available to the public since 2002 and this is the first release of Apache's major release 2.0. As for the GNOME project, GNOME 2.0 is a

major upgrade which includes the introduction of the human interface guidelines. The first two steps of our method do not apply because defect data were published [24].

Besides these two projects, we used the issues of the five notable and active open source projects that have been downloaded from apache.org (https://issues.apache.org/) for our pervious study (see chapter 4 table 4.4). These projects are C++ Standard Library, JUDDI, HTTP Server, XML Beans, and Enterprise Social Messaging Environment (ESME). All these projects are considered stable in production. The 66%, 95%, 68%, 64% and 82% of the reported issues in these projects respectively, have been fixed and closed. All the issues about the selected projects have been collected from apache.org using JIRA. JIRA is a commercial issue tracker. Issues can be bugs, feature requests, improvements, or tasks. JIRA tracks bugs and tasks, links issues to related source code, plans agile development, monitors activity and reports on project status.

We selected three releases of C++ Standard Library, three releases of JUDDI, two releases of HTTP Server, four releases of XML Beans and three releases of ESME. In the downloaded issue the tracking software (i.e. JIRA) records all the information regarding each issue, among which the following are the more useful attributes that we used for selecting defects from issues:

- *Project:* Contains the project name;
- *Key:* The unique identity of each issue.
- *Summary:* Gives a comprehensive description of the issue.
- *Issue Type:* Describes the type of issue, which may be bug, task, improvement, or new feature request.
- *Status:* Describes the current status of the issue. It may be resolved, closed, open or reopened.
- *Resolution:* Shows the resolution of the issue, which may be fixed, duplicate, or invalid.
- *Created:* Shows the created date and time of the issue.
- *Updated:* Shows the updated/fixing date and time of the issue.
- *Affects Version:* Gives the release/version of the project that contains the issue.

All the issues were filtered to eliminate duplicate and invalid issues. We focused only on those issues that were declared "bug" or "defect" excluding "improvement," "feature-request," or "task". We considered only those issues that were reported as closed or resolved after the release date of each version and removed issues that were reported as open or reopened because their status is not clear: they are either defects/bugs or improvement/feature-request/task. We collected defects according to their creation date (i.e. date of opening) and then grouped cumulative defects by week. We filtered all the

issues in order to collect only issues that have been declared "defect" or "bug" as in [21, 37]. Due to space limitation the full defect data sets of each release are available online (http://softeng.polito.it/najeeb/IEEE/DataSets.pdf).

6.3.2 Results

Due to space limitations we show here (Table 5.1) the results of the application of the method to one release of a project. The results for all releases of all projects are available here: http://softeng.polito.it/najeeb/IEEE/Remaining_results.pdf. Table 5.2 shows the summary of all results. The method has been applied using, for each release of each project, 2/3rds of the time window available, and of the corresponding defects. The remaining 1/3 is used for validation, as explained in section 3.

Let's start by discussing the application of the method to GNOME release 2.0 (Table 5.1). The time interval available is 24 weeks. As said we use 2/3 of this time frame for model selection, or 16 weeks. Of these, we use the last 1 /4 for model stability check, or weeks 12 to 16. Table 5.1 contains a row for each week, from 12 to 16. For each week the table shows, in the columns, from left to right: the cumulative number of actual defect found in that week, and, for each of the eight SRGMs, the number of total defects predicted, and the R^2 value.

Each of the eight models is fitted each week. When a model fails the stability check it is rejected (this is indicated by letter 'R'). The week a model stabilizes is indicated by an 'S'. If a model becomes unstable after being stable, this is indicated by a 'D'. The model selected by the method is underlined.

Table 6-1: Model fitting for stability check, for release 2.0 of GNOME project

Weeks after release	Actual Defects	Musa		Inflection		Goel		Delayed		Logistic		Yamada		Gompertz		Generalized	
		Pred.	R^2	Pred.	R^2	Pred.	R^2	Pred.	R^2	Pred.	R^2	Pred.	R^2	Pred.	R^2	Pred.	R^2
12	58	945	0.9806	66	0.9859	926	0.9806	68	0.974	59	0.9937	1743	0.9806	73	0.9889	105	0.9819
13	58	446**D**	0.9836	791**D**	0.9836	455**D**	0.9836	71**S**	0.9781	63**S**	0.9937	883**D**	0.9836	74	0.9909	100	0.9852
14	66							78	0.9764	69	0.9879			84**D**	0.9891	202**D**	0.9866
15	72							86	0.9747	77	0.9844						
16	74							90	0.9772	83	0.986						

In Table 5.1 GOF values of all the models show that all the models perform very well in terms of fitting and pass the rejection criteria but their predictions are different. Musa, Inflection, Goel, and Yamada destabilize at week 13 whereas Gompertz and Generalized destabilize at week 14. All these models overestimate by a very large amount.

From Table 5.1 it is also clear that the Delayed S-shaped and Logistic models stabilize first at week 13 and remain stable up to week 16 (i.e. throughout the whole stability check window). The Delayed S-Shaped and Logistic models predict the number of

defects at week 16 as 90 and 83 respectively. Since the Delayed S-shaped model predicts more residual defects than Logistic, it is selected.

Table 5.2 summarizes the selected models for all releases of all projects. The table shows also for each model selected the GOF value and the total number of defects estimated.

Table 6-2: Summary of selected Models

Project	Release	Selected Model	Model's estimated total number of defects	R^2 Value
GNOME	V2.0	Delayed	90	0.9772
	V2.2	Goel	57	0.9776
	V2.4	Inflection	59	0.9876
Apache	2.0.35	Gompertz	78	0.9929
	2.0.36	Generalized	56	0.9623
	2.0.39	Goel	60	0.9896
C++ Stand. Lib	4.1.3	Inflection	54	0.9887
	4.2.3	Gompertz	15	0.9526
	5.0.0	Yamada	15	0.9580
JUDDI	3.0	Delayed	40	0.9553
	3.0.1	Delayed	41	0.9570
	3.0.4	Goel	57	0.9532
HTTP Server	3.2.7	Goel	46	0.9709
	3.2.10	Logistic	19	0.9651
XML Beans	2.0	Delayed & Gompertz	36	0.9871, 0.9882
	2.2	Logistic	34	0.9696
	2.3	Logistic	23	0.9690
	2.4	Gompertz	20	0.9828
ESME	1.1	Goel	48	0.9767
	1.2	Gompertz	12	0.9569
	1.3	Inflection, Generalized & Goel	11	0.9543, 0.9523

The proposed method uses R^2 based GOF to filter out unsuitable model (i.e. step 4) in terms of fitness. A useful model is required to be stable in prediction, because if a model says there are 50 residual defects one week and 200 the next, no one is going to believe either prediction. For a model to be accepted by management, its prediction should not vary significantly. Therefore, in stability check (i.e. step 5) the method filters out unsuitable models in terms of their prediction. Among all the stable models, instead of best fitting (the model with best R^2) conservative choice (the model with maximum prediction) is used for selecting the best model. The rationale behind this choice is to select safer model. From our previous study [54] we know that the best fitted model is not necessarily the best predictor model. This is also clear from the Table 5.1, the Logistic is best fitted (i.e. R^2=0.986) compared to Delayed S-Shaped (i.e. R^2=0.9772), but

Delayed S-Shaped predicts best. We underline here some points regarding the application of the method.

- Most of the time models are rejected due to prediction instability instead of GOF value. This could be explained by the fact that not enough defect data is available. However the method usually overcomes this problem with the wide number of models available.
- The stability evaluation is a heuristic way; a stable model can become unstable, and vice versa and there is some likelihood that the choice will lead to bad predictions. However, the method usually overcomes this problem with the wide number of models available, and applying the models to whole stability check window (3/4T to T).
- S-Shaped models outperform concave ones in 14 out of 21 cases, which also confirm the results of our previous studies [54, 55]. S-Shaped models are better probably because initially the community of end-users and reviewers of the open source projects do not react promptly to a new release. This is modeled by the learning phase included in the S-Shaped models.
- Time unit (i.e. day or week) has no effect on the application of the method because for SRGM the test time can be measured using calendar time, the number of tests run, or execution time. However, it is suggested to group defect data by weeks because the smaller the amount of data the longer the time may take the models to stabilize, or even not to fit the data.
- There is at least one model that fits the defect data after 5 weeks (in terms of the method, at least a model passes step 4 after 5 weeks). So 5 weeks could be the suggested delay from release before doing any analysis about reliability.

6.4 Validation

We apply our method to 2/3 of each release interval defect data to select the best model. The remaining defect data of each release are used for validation. The choice of 2/3 release interval for the estimation of model parameters was motivated by Wood's suggestion that the model parameters do not become stable until about 60% of the way through the test [2].

As a measure to validate the prediction capability of a model we use the PRE (Prediction Relative Error) indicator.

$$PRE = \frac{Predicted - Actual\ number\ of\ defects}{Predicted}$$

Where *Predicted* is the total number of defects predicted by a model, as fitted at 2/3 of the time interval available, and *Actual number of defects* is the number of defects at the end of the time interval.

For each release and each project we compute PRE for each model, and rank the models accordingly. The model with minimum PRE is considered the best predictor model for that release.

Table 5.3 shows the validation results: for each project and each release the best model on PRE, and the best model according to the method. We observe that 17 out of 21 times, the model selected by the method corresponds to the best model. In the remaining four cases (see underlined model names in the table) the best model has a negative PRE, and for this reason was rejected by the method. However, in these four cases the model selected by the method is the one with the lowest positive PRE (the lowest negative for C++ 4.2.3). So in these four cases the method selects the second best model.

Table 6-3: Best predictor model selected by our method vs. best predictor selected on prediction PRE

Project	Release	PRE of the model								Best model on PRE	Best model selected by proposed method
		Musa	Inf.	Goel	Delay.	Log.	Yama.	Gomp.	General.		
GNOME	V2.0	0.19	0.08	0.18	**0.01**	-0.02	0.18	0.05	-0.85	Delayed	Delayed
	V2.2	0.09	-0.07	**0.00**	-0.11	-0.10	**0.00**	-0.10	-0.47	Goel, Yamada	Goel
	V2.4	0.14	**0.01**	0.04	-0.03	-0.02	0.03	**-0.01**	-0.28	Inflection, Gompertz	Inflection
Apache	2.0.35	0.17	0.07	0.12	**-0.01**	**-0.01**	0.12	0.03	-0.53	**Delayed, Logistic**	**Gompertz**
	2.0.36	0.18	0.03	0.09	**0.01**	**0.01**	0.09	**0.01**	**0.01**	Delay, Log, Gompertz, General.	Generalized
	2.0.39	0.15	**0.00**	**0.00**	-0.02	-0.02	0.02	-0.01	**0.00**	Inflection, Goel, General.	Goel
C++ Stand. Lib	4.1.3	**-0.22**	-0.25	-0.39	-0.52	-0.45	-0.34	-0.41	-0.31	**Musa**	**Inflection**
	4.2.3	**-0.01**	-0.05	-0.12	-0.20	-0.07	-0.09	-0.03	0.06	**Musa**	**Gompertz**
	5.0.0	-0.28	**-0.16**	-0.66	-0.98	-0.92	**-0.16**	-0.83	**-0.16**	Inflection, Yamada, General.	Yamada
JUDDI	3.0	**0.00**	0.24	**0.00**	**0.00**	0.26	**0.00**	**0.00**	0.13	Musa, Goel, Delay, Yama, Gomp	Delayed
	3.0.1	**-0.02**	-0.32	-0.10	**-0.02**	-0.05	-0.10	-0.27	-0.29	Musa, Delayed	Delayed
	3.0.4	-0.30	-0.33	**-0.27**	-0.72	-0.67	-0.34	-0.59	-0.33	Goel	Goel
HTTP Server	3.2.7	0.04	-0.10	**0.02**	-0.18	-0.16	-0.05	-0.11	-0.05	Goel	Goel
	3.2.10	0.23	0.22	0.23	**0.01**	**0.01**	0.23	0.13	0.23	Delayed, Logistic	Logistic
XML Beans	2.0	0.17	0.02	0.17	-0.02	-0.14	0.17	**0.01**	0.07	Gompertz	Delayed & Gompertz
	2.2	-0.04	0.04	-0.04	-0.29	**0.01**	-0.03	0.11	-0.02	Logistic	Logistic
	2.3	**0.00**	0.05	**0.00**	-0.25	**0.00**	**0.00**	0.01	**0.00**	Musa, Goel, Log, Yama, General.	Logistic
	2.4	0.24	-0.12	0.24	**0.06**	-0.16	0.24	**0.06**	-0.11	Delayed, Gompertz	Gompertz
ESME	1.1	**-0.03**	-0.23	**-0.03**	-0.32	-0.34	-0.14	-0.29	-0.23	Musa, Goel	Goel
	1.2	-0.26	-0.33	-0.41	-0.57	**-0. 09**	-0.37	**-0.09**	-0.16	Logistic, Gompertz	Gompertz
	1.3	0.18	0.07	0.07	**-0.02**	**-0.02**	0.08	**-0.02**	0.07	**Delayed, Logistic, Gompertz**	**Inflection, Goel, Generalized**

6.5 Threats to validity

We discuss here validity threats for this work, using the classification suggested by [57].

The first construct threat comes from the issues data sets used. Our work is second level, so we have no control on the quality of the issues collected and reported. Issues could be

missing; others could have been miss-reported, either on time or on content. The subset of issues classified as defects by the project community could be imprecise too, both for false positives and false negatives. Overall we have tried to reduce this threat by selecting established Open Source projects and communities. Most of the data sets have also been used in other similar works.

The second construct threat comes from the defect data sets used. While developing defect data sets (in step 2 of the method) discarding not closed issues may affect the models fitting and prediction. However, we selected the projects considered stable in production and more than 64% (i.e. 64%, 66%, 68%, 95%, and 82%) of the reported issues in each project have been fixed and closed.

Still on construct validity, we have considered each release of a project as a separate project, independent of others. This choice is in line with [24, 26, 34, 54, 56] and corresponds to the goal of the method: users of OSS select and use a specific release. As a cross validation of this independence it should be noted that different releases of the same project are best fitted by different SRGMs.

The time span covered by the datasets of projects, and project versions, is quite different. We assume this is not critical, especially because we do not compare project versions, but we consider each project version independently.

We recognize one external threat to validity. We have evaluated our method on 21 projects. This is one of the largest datasets in literature (see chapter 2) but we cannot generalize the results to all projects. In particular the method could just be not applicable to a project because curve fitting, satisfaction of GOF or stability thresholds could fail. In these cases the thresholds can be changed, or more time should pass so that more defect data is available.

6.6 Conclusion

The goal of this work was to support practitioners in characterizing the reliability (in terms of residual defects) of an OSS component or application, to help a project manager in deciding whether using the OSS or not in a project. To achieve the goal we proposed a pragmatic approach, which selects best model both on its fitting capability, and on the stability of its prediction over time. The model selected with our proposed method, among several alternative models predicts very precisely the residual defects of an OSS.

Empirical evaluation has shown that the research question (RQ) derived on the object of the study has been satisfied. The results show that the method selects the best model (i.e. the model with the better precision in the estimation of the residuals), or second best model, in all cases.

We believe that the key contribution of our work lies in the systematic approach (eight SRGMs have been considered) and in the extent of validation (21 releases of seven projects). As it is clear from chapter 2 no other study has achieved this level of coverage.

There are also some side contributions to be underlined.

- The method tested in such a generalized way that the unclear results regarding the applicability of SRGM for reliability characterization of OSS reported in the literature (see chapter 2) are made clear.
- No model is clearly superior to others. In fact, in the 21 data sets no model ranks as the best one in more than a few cases, and each of the eight models is the best one at least once.
- Even more surprisingly, different releases of the same project (see for instance XML Beans in Table 5.2) are each fitted by different models. This result seems to suggest that only the history of defects (and not any other project characteristic) counts for the model selection.
- S-Shaped models outperform concave ones in OSS projects, probably because of the learning curve, or the time needed by the OSS community to report defects.
- Five weeks of defect data from the release is the time frame needed to fit a model. This could be used as an initial rule of thumb by managers in decisions about reliability.

The method we propose sets some parameters: the GOF minimal threshold for fitting (0.95), the stability threshold (10%), the time frames (last 1/4 of defect data for fitting and stability check). These parameters were set using suggestions from the literature and seem to perform well.

There are three directions of future research work. First future work could perform some kind of sensitivity analysis on them to further improve the results. Second future work is the development of a tool to automate the method. However, filtering out issues to isolate defects (step 2 in the method) is a time consuming task that cannot be automated. The method we developed helps project manager in deciding whether using the OSS or not in a project. For that purpose we selected all the OSS projects and versions that have been released. Another possible line of future research is to investigate the research question, is the proposed method also applicable to early OSS projects in predicting the residual defects to decide the first stable release?

Bibliography:

[1] C. Smidts, R. W. Stoddard, and M. Stutzke, "Software reliability models: An approach to early reliability prediction," *IEEE Trans. Reliability*, vol. 47, no. 3, pp. 268–278, 1998.

[2] Wood, Alan, "Software-reliability growth model: primary-failures generate secondary-faults under imperfect debugging and. IEEE Transactions on Reliability, 43, 3, 408 (September 1994)."

[3] S. S. Gokhale, P. N. Marinos, and K. S. Trivedi, "Important milestones in software reliability modeling," in *Proc. of Softw. Engineering and Knowledge Engineering (SEKE '96)*, Lake Tahoe, NV, 1996, pp. 345–352.

[4] S. Gokhale, W. E.Wong, K. S. Trivedi, and J. R. Horgan, "An analytical approach to architecture-based software reliability prediction," in *IEEE International Computer Performance and Dependability Symposium*, September 1998, pp. 13–22.

[5] C. Y. Huang and S. Y. Kuo, "Analysis of incorporating logistic testing effort function into software reliability modeling," *IEEE Trans. Reliability*, vol. 51, no. 3, pp. 261–270, 2002.

[6] K. Pillai and V. S. S. Nair, "A model for software development effort and cost estimation," *IEEE Trans. Softw. Engineering*, vol. 23, no. 8, pp. 485–497, 1997.

[7] S. Hwang and H. Pham, "Quasi-renewal time-delay fault-removal consideration in software reliability modelling," *IEEE Trans. Systems, Man and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 1, January 2009.

[8] K. C. Chiu, Y. S. Huang, and T. Z. Lee, "A study of software reliability growth from the perspective of learning effects," *Reliability Engineering and System Safety*, pp. 1410–1421, 2008.

[9] X. Zhang, X. Teng, and H. Pham, "Considering fault removal efficiency in software reliability assessment," *IEEE Trans. Systems, Man, and Cybernetics— Part A*, vol. 33, no. 1, pp. 114–120, 2003.

[10] P. L. Li, J. Herbsleb, and M. Shaw, "Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of OpenBSD," in *Proceedings of the 16th IEEE International Symposium on Softw. Reliability Engineering*, Chicago, IL, 2005, pp. 193–202.

[11] S. Brocklehurst, P. Y. Chan, B. Littlewood, and J. Snell, "Recalibrating software reliability models," IEEE Trans. Softw. Engineering, vol. SE-16, no. 4, pp. 458–470, April 1990.

[12] A. L Goel, "Software reliability models: assumption, limitations, and applicability," IEEE Transaction on Software Engineering, pp. 1411–1423, December 1985.

[13] Abdel-Ghaly, P.Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," IEEE Trans. Softw. Engineering, vol. SE-12, no. 12, pp. 950–967, September 1986.

[14] T. M. Khoshgoftaar, "On model selection in software reliability," in 8th Symposium in Computational Statistics, August 1988, pp. 13–14, (Compstat '88).

[15] T. M. Khoshgoftaar and T. G. Woodcock, "A simulation study of the performance of the Akaike information criterion for the selection of software reliability growth models," in Proc. of the 27th Annual South East Region ACM Conf., April 1989, pp. 419–423.

[16] T. M. Khoshgoftaar and T. Woodcock, "Software reliability model selection: A case study," in Proc. of the 2nd International Symposium on Software Reliability Engineering. Austin, TX: IEEE Computer Society Press, 1991, pp. 183–191.

[17] M. Lyu and A. Nikora, "CASREA—A computer-aided software reliability estimation tool", in Proc. of the Fifth International Workshop on Computer-Aided Softw. Engineering, Montreal, CA, 1992, pp. 264–275.

[18] J. Zeng et al, "A Prototype System of Software Reliability Prediction and Estimation", Third International Symposium on Intelligent Information Technology and Security Informatics, 2010, Page No. 558-561.

[19] Syed-Mohamad et al, “Reliability Growth of Open Source Software Using Defect Analysis”, International Conference on Computer Science and Software Engineering, 2008.
[20] Ying Zhou et al, “Open source software reliability model: an empirical method”, ACM SIGSOFT Software Engineering Notes, 2005
[21] Bruno Rossi et al, “Modelling Failures Occurrences of Open Source Software with Reliability Growth”, journal of Open Source Software: New Horizons, page 268-280, 2010.
[22] Cobra Rahmani et al, “A Comparative Analysis of Open Source Software Reliability”, Journal of Software, page 1384-1394, 2010.
[23] Fenhzong et al, “Analyzing and Modeling Open Source Software Bug Report Data”, 19th Australian Conference on Software Engineering.
[24] Xiang Li et al. 2011. Reliability analysis and optimal release-updating for open source software. Information and Software Technology 53 (2011) 929–936.
[25] IEEE Std 1633-2008 IEEE Recommended Practice in Software Reliability.
[26] Kapil Sharma, et al, “Selection of Optimal Software Reliability Growth Models Using a Distance Based Method”, IEEE Transactions on Reliability, Vol. 59, No. 2, June 2010.
[27] J.D. Musa, A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction and Application. McGraw-Hill, 1987.
[28] Wen-Li Wanga, et al. “Architecture-based software reliability modeling”, Journal of system and software, Volume 79, Issue 1, January 2006, Pages 132-146.
[29] M.R. Lyu, Handbook of Software Reliability Engineering. McGraw Hill, 1996.
[30] J. D. Musa and K. Okumoto, “A logarithmic Poisson execution time model for software reliability measurement,” in Conf. Proc. 7th International Conf. on Softw. Engineering, 1983, pp. 230–237.
[31] M. Xie, Software Reliability Modeling. World Scientific Publishing, 1991.
[32] G. H. Schick and R. W. Wolverton, “An analysis of competing software reliability models,” IEEE Trans. Software Engineering, pp. 104–120, March 1978.
[33] A. N. Sukert, “Empirical validation of three software errors predictions models,” IEEE Transaction on Reliability, pp. 199–205, August 1979.
[34] Stringfellow, et al. An empirical method for selecting software reliability growth models. Empirical Software Engineering, Journal. Page 319-343, 2002.
[35] Arora, S. et al, “Software Reliability Improvement through Operational Profile Driven Testing”, Proceedings Reliability and Maintainability Symposium, 2005.
[36] Xuemei Zhang, et al, “Considering Fault Removal Efficiency in Software Reliability Assessment”, IEEE Transactions on Systems, Man, and Cybernetics—PART A: System and Human, Vol. 33, No. 1, January 2003
[37] P. Li et al. Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources : a Case Study of OpenBSD CMU-ISRI-05-12, 2005.
[38] Yongqiang Zhang, et al, “Improved Genetic Programming Algorithm Applied to Symbolic Regression and Software Reliability Modeling”, Journal of Software Engineering and Applications, pages 354-360, 2009
[39] Mullen, Robert E, “The Lognormal Distribution of Software Failure Rates: Application to Software Reliability Growth Modeling Cisco Systems 250 Apollo Road”.
[40] Jeske, Daniel R., et al, “Adjusting Software Failure Rates That Are Estimated from Test Data”, IEEE Transaction on Reliability, Vol. 54, No. 1, March 2005.

[41] Jeske, D.R., et al, “Accounting for Realities When Estimating the Field Failure Rate of Software”, Proceedings of 12th International Symposium on Software Reliability Engineering, 2001.
[42] Jeske, D.R., et al, “Estimating the failure rate of evolving software systems”, Proceedings 11th International Symposium on Software Reliability Engineering, 2000.
[43] Derriennic, H., Le Gall, G., “Use of failure-intensity models in Software-validation Phase for Telecommunications”, IEEE Transaction on Reliability, Vol. 44, No. 4, 1995 December.
[44] Cao, Yong. , Zhu, Qingxin, “The Software Failure Prediction Based on Fractal”, Journal Advanced Software Engineering and Its Applications, Pages 85-90, 2008.
[45] Jalote, Pankaj, et al, “Post-release reliability growth in software products”, ACM Transactions on Software Engineering and Methodology, Vol. 17, Pages 1-20, 2008.
[46] Lo, J., Huang, C., “An integration of fault detection and correction processes in software reliability analysis”, Journal of Systems and Software, Vol.79, Pages 1312-1323, 2006.
[47] Li, Xiang, et al, “Reliability analysis and optimal version-updating for open source software”, Journal of Information and Software Technology, Vol. 53, Pages 929-936, 2011.
[48] Jeske, D, “Some successful approaches to software reliability modeling in industry”, Journal of Systems and Software, Vol. 74, Pages 85-99, 2005.
[49] Kelani Bandara, K.B.P.L.M. et al, “Optimal selection of failure data for reliability estimation based on a standard deviation method”, Proceedings of International Conference on Industrial and Information Systems, 2007.
[50] Carina Andersson , “A replicated empirical study of a selection method for software reliability growth models”, journal of Empirical Software Engineering, pages 161-182, year 2006
[51] Basili, et al, "The Goal Question Metric Approach", in Encyclopedia of Software Engineering, Wiley, 1994.
[52] O. Gaudoin et al, “A simple goodness-of- fit test for the power law process based on the Duane plot”, IEEE Transactions on Reliability.
[53] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, Experimentation in software engineering: an introduction. Norwell, Massachusetts. USA: Kluwer Academic Publishers, 2000.
[54] Najeeb Ullah, Maurizio Morisio, Antonio Vetro, “A Comparative Analysis of Software Reliability Growth Models using defect data of closed and open source software”, proceedings of SEW-35, 2012.
[55] Najeeb Ullah, Maurizio Morisio, “An Empirical Study of reliability growth of open versus closed source software through software reliability growth models”, proceedings of APSEC 2012.
[56] Stringfellow, C. 2000. An integrated method for improving testing effectiveness and efficiency. PhD Dissertation, Colorado State University.
[57] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA.
[58] Goel, A. L., and Okumoto, K. 1979. “A time dependent error detection model for software reliability and other performance measures”, IEEE Transactions on Reliability 28(3): 206–211.
[59] Kececioglu, D. 1991. Reliability Engineering Handbook, Vol. 2, Englewood Cliffs, NJ: Prentice-Hall.
[60] Musa, J., Iannino, A., and Okumoto, K. 1987. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill.
[61] Trachtenberg, M. 1990, “A general theory of software-reliability modeling”, IEEE Transactions on Reliability 39(1): 92–96.
[62] Yamada, S., Ohba, M., and Osaki, S. 1983. “S-shaped reliability growth modeling for software error detection”, IEEE Transactions on Reliability 32(5): 475–478.

[63] Musa, J., and Ackerman, A. 1989, Quantifying software validation: When to stop testing. IEEE Software. 19–27
[64] M. Ohba, “Software reliability analysis models”, IBM. J. Research Development, vol. 21, no. 4, 1984.
[65] H. Pham, L. Nordmann, “A generalized NHPP software reliability model”, 3rd Int’l Conference on Reliability and Quality in Design, 1997.
[66] Najeeb ullah, Maurizio Morisio, “An Empirical Study of Open Source Software Reliability Models”, Proceedings of International conference on computational intelligence and computing research, 2011.
[67] R.C. Cheung, “A user-oriented software reliability model”, IEEE Trans Software Eng., vol. 6, no. 2,1980, pp. 118-125.
[68] David, N. C. (2002) Managing Software Quality with Defects, In Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Redevelopment IEEE Computer Society.
[69] T.R. Moss, The Reliability Data Handbook, ASME 2005.
[70] B. Littlewood, A. Ghaly, P.Y. Chan, “Tools for the Analysis of the Accuracy of Software Reliability Predictions”, in Soft. Sys. Design Methods, J.K. Skwirznski, Ed., NATO ASI Series, vol. F22, 1986, pp.
299-335.
[71] Farber D. Six barriers to open source adoption, ZDNet Tech Update, March, 2004 [WWW document].
[72] Yamada, S., Ohtera, H., and Narihisa, H. 1986. Software reliability growth models with testing effort. IEEE Transactions on Reliability 35(1): 19–23.
[73] Wood, A. 1996, Predicting software reliability. IEEE Computer 29(11): 69–78.
[74] Wood, A. 1997. Software reliability growth models: Assumptions vs. reality. Proceedings of the International Symposium on Software Reliability Engineering 23(11): 136–141.
[75] A. Abdel-Ghaly, P. Chan, B. Littlewood, “Evaluation of Competing Software Reliability Predictions";
IEEE Transactions on Software Engineering, SE-12(9), Sept. 1986, pp. 950-967.
[76] C. Y. Huang, M. R. Lyu, and S. Y. Kuo, “A unified scheme of some non-homogenous Poisson process models for software reliability estimation,” IEEE Trans. Softw. Engineering, vol. 29, no. 3, pp. 261–269, March 2003.
[77] K. Pillai and V. S. S. Nair, “A model for software development effort and cost estimation,” IEEE Transaction on Software Engineering, vol. 23, no. 8, pp. 485–497, 1997.
[78] M.W. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study,” Proc. Int’l Conf. Software Metrics, 2000.
[79] J.W. Paulson, “An Empirical Study on the Growth of Open Source and Commercial Software Products,” master’s thesis, Dept. of Electrical and Computer Eng., Univ. of Calgary, Alberta, 2001.
[80] E.S. Raymond, “The Cathedral and the Bazaar,” http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/, Aug. 2003.
[81] D. Wheeler, “Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!,” http://www.dwheeler.com/oss_fs_why.html, Aug. 2003.
[82] Wangbong Lee, Boo-Geum Jung, and Jongmoon Baik, “Early Reliability Prediction: An Method to Software Reliability Assessment in Open Software Adoption Stage,” in Secure System Integration and Reliability Improvement, 2008. SSIRI '08. Second International Conference on, pp. 226-227, 2008.
[83] Y. Zhou and J. Davis, “Open source software reliability model: an empirical method,” SIGSOFT Softw. Eng. Notes, vol. 30, no. 4, pp. 1-6, 2005.

[84] S. Brown, “What exactly are the merits of open-source software? Microsoft and Novell go head to head to talk out the issues - View #2,” IEE Review, vol. 50, no. 10, pp. 48-50, 2004.
[85] A. Mockus, R.T. Fielding, and J. Herbsleb, “A Case Study of Open Source Software Development: The Apache Server,” Proc. 22nd Int’l Conf. Software Eng., 2000.
[86] AIAA SBOS/COS Software ReliabilityWorking Group, “AIAA Software Reliability Engineering Recommended Practice", R-013-1992, American Institute of Aeronautics and Astronautics.
[87] K. Cai, C. Wen, M. Zhang, “A Critical Review on Software Reliability Modeling", Reliability Engineering and System Safety, vol. 32, 1991, pp. 357-371.
[88] S. Conte, H. Dunsmore, V. Shen, Software Engineering Metrics and Models, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1986.
[89] H. Pham, “Software reliability and cost models: perspectives, comparison and practice,” European J. of Operational Research, vol. 149, pp. 475–489, 2003.
[90] S. Eick, C. Loader, M. Long, L. Votta, S. VanderWeil, “Estimating Software Fault Content Before Coding," Procs. International Conference on Software Engineering, (1992), Melbourne, Australia, pp. 59-65.
[91] A.L. Goel, “Software Reliability Models: Assumptions, Limitations, and Appicability," IEEE Transaction on Reliability, vol. 11, no. 12, (December 1985), pp. 1411- 1421.
[92] R Project, http://www.r-project.org/.
[93] SMERFS, http://www.slingcode.com/smerfs/.
[94] SPSS, http://www.spss.com/statistics.
[95] https://issues.apache.org/
[96] PRISM, http:// www.graphpad.com