Master Thesis
Software Engineering
Thesis no: MSE-2004-17
June 2004

Software Reliability Prediction

– An Evaluation of a Novel Technique

Björn Andersson **Marie Persson**

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Björn Andersson

Marie Persson
Email: marie.persson@bth.se

University advisor(s):
Mikael Svahnberg
School of Engineering

Håkan Lennerstad
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.tek.bth.se
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

Along with continuously increasing computerization, our expectations on software and hardware reliability increase considerably. Therefore, software reliability has become one of the most important software quality attributes.

Software reliability modeling based on test data is done to estimate whether the current reliability level meets the requirements for the product. Software reliability modeling also provides possibilities to predict reliability. Costs of software developing and tests together with profit issues in relation to software reliability are one of the main objectives to software reliability prediction.

Software reliability prediction currently uses different models for this purpose. Parameters have to be set in order to tune the model to fit the test data. A slightly different prediction model, *Time Invariance Estimation*, TIE is developed to challenge the models used today. An experiment is set up to investigate whether TIE could be found useful in a software reliability prediction context. The experiment is based on a comparison between the ordinary reliability prediction models and TIE.

Keywords: Reliability, Prediction, Time Invariance Estimation, Experiment

CONTENTS

ABSTRACT 1

1 INTRODUCTION 3

1.1 SOFTWARE RELIABILITY 3
1.1.1 Software Reliability Definition 3
1.1.2 Reliability Estimation 3
1.1.3 Software Reliability Modeling and Prediction 5
1.2 AN ATTEMPT TO IMPROVE PREDICTION MODELING 6
1.3 THESIS CONTRIBUTION 6
1.4 OUTLINE OF THESIS 6

2 BACKGROUND & RELATED WORK 7

2.1 WHY SOFTWARE RELIABILITY MODELING 7
2.1.1 Trade-off, cost of development verses cost of failures in released software 7
2.2 RELIABLE SYSTEM APPROACHES 8
2.2.1 Model types 9
2.3 EVALUATIONS OF FC-MODELS 12
2.3.1 FC-models 12
2.4 PARAMETER ESTIMATION 13

3 TIME INVARIANCE ESTIMATION 14

3.1 INTRODUCTION 14
3.1.1 Modification techniques 14
3.1.2 Pattern matching 16
3.1.3 Compromise Techniques 17
3.1.4 Summary of TIE 18
3.2 SOFTWARE OF TIE 19
3.2.1 Architecture and Design 19
3.2.2 Implementation 19
3.2.3 Current version of TIE 20

4 EXPERIMENT 21

4.1 INTRODUCTION AND PROBLEM STATEMENT 21
4.1.1 Experiment Definition 21
4.2 PLANNING 21
4.2.1 Hypothesis 22
4.2.2 Experiment Design 23
4.2.3 Validity 26
4.3 EXPERIMENT OPERATION 27
4.3.1 Preparation 27
4.3.2 Execution 27
4.4 ANALYSIS 28
4.4.1 Discussion 28
4.4.2 Data Results 28
4.4.3 Hypothesis Testing 30

5 CONCLUSIONS 31

5.1 FUTURE WORK 31

6 REFERENCES 32

APPENDICES

1 competition_results.pdf, 7 pages

1 INTRODUCTION

1.1 Software Reliability

Today, more and more of the world around us is computerized. Along with this, more and more sensitive operations are computer controlled, for example medical monitoring, economical transactions etc. Concurrently we increase our confidence in that the software responds correctly. Accordingly, our expectations on software reliability are assumed to be obvious and are therefore one of the most important software quality attributes.
To assess a certain reliability level we have to determine software reliability. Software usage varies depending on several factors like type of software, type of environment and so forth. Still there is a need of capturing a usage that, as good as possible, represents the main purpose of the software. How to ensure that software reliability meet the user needs is therefore of great interest.
A simulation or a complete specification of the usage is thus required so that the software can be tested based on the simulation in order to provide data to estimate the reliability.
To what extent we can perform this test activity is a relevant question. It could lead to an exhaustive testing with enormous costs before product market. This indicates that the software engineering is forced to include calculations of costs, i.e. costs for product developing and testing, when estimating software reliability. We need to find the moment when the software reliability is of satisfactory and the costs are reasonable.
Due to this, the activities within the area of software reliability consist of techniques of how to estimate but also predict the reliability. Predict, in order to manage software developing and testing in relation to the cost. First, we have to *estimate* the current reliability before we can *predict* any future reliability.
In summary, the following coherent activities are related to the aim of software reliability;

- ◊ Simulation/mapping of usage in order to specify the usage
- ◊ Statistical testing based on the specified usage
- ◊ Modeling the test result to estimate current level of reliability and to be able to certify the reliability.
- ◊ Prediction, using reliability modeling. The main purpose of prediction activity is related to restrictions concerning software development.

1.1.1 Software Reliability Definition

Before we can estimate the software reliability, we must define software reliability. Software reliability can be defined as follows;
"*The probability of failure-free software operation for a specific period of time in a specified environment*". (7)

1.1.2 Reliability Estimation

We estimate the reliability in order to certify the reliability level to meet the requirements set for the product. When the reliability is estimated, we have a possibility, using the modeled data, to predict the future reliability. With help from prediction, we might be able to find the crucial point when the software reliability is of a satisfactory degree and the costs are reasonable. Reasonable, in relation to costs but also in relation to time to market with profit issues. This is actually the main objective of reliability prediction, and is further discussed in section 2.1.1.

When estimating software reliability you sample data to model the current level of reliability. The sampled data are different characteristics of the failure process. Depending on what model you use, the data could, for example, be expression of:

1. The average number of failures experienced at any point in time.
2. The average number of failures in a time interval.
3. The failure intensity at any point in time.
4. The probability distribution of failures intervals.

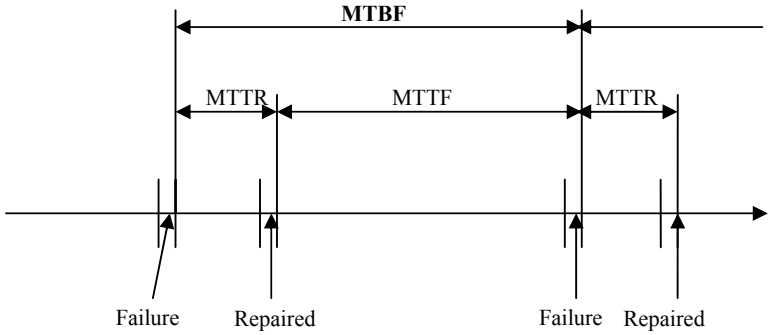
(9, page 37)
According to these statements and the reliability definition stated in section 1.1.1, we have to decide what a *failure* is. By inference, we need to introduce concepts like *error, fault* and *failure.*
Error; An error is an action taken by a human being that can result in a fault in the software.
Fault; A fault is a defect in software, also called a bug, which can lead to a failure if it is being executed.
Failure; A failure occurs when a user consider the software to fail in responding correctly which implies the software to be *executed* during failure occurrence.
With reference to the failure definition, it is obvious that the usage of the software is a very important constituent element in software reliability. The *expected usage* must be taken into consideration when estimating the reliability. Tests have to be performed to represent the usage of the system. There exists several techniques to specify the usage in order to perform usage based testing, e.g. Operational profile, Algorithm model, Markov model etc (3, page 5-12). These techniques are different approaches to model the usage in order to specify the same.
Consider the following data of software experience (4, slide 14), where;

- ◊ 61% of faults account for only 2.8% of failures, and
- ◊ 1.6% of faults account for 58% of failures.

If we go back to the reliability definition again (1.1.1), we realize the importance of representative data from the usage specification in order to estimate of the software reliability. However, it should not only be the quantity of the usage that has to be mirrored, but also the importance of a specific usage. Therefore, if there is some usage that is more important not to fail, it has to be weighted. However, the intention to sample test data reflecting a simulated usage, with considerations taken to both ordinary usage and importance of specific usage of software, is not an easy task. We will not go any further into this area in this thesis other than to emphasize its significance and difficulty when estimating and predicting software reliability.
Statistical testing samples data based on a characterization of the expected usage to be able to test from a usage perspective. These test results will constitute the foundation to what we estimate our reliability upon. In order to estimate reliability we must illustrate the test results by modeling.

1.1.3 Software Reliability Modeling and Prediction

We have brought up the importance of capturing the expected usage, now we also introduce the importance of homogenous and measurable data from the test result in order to model something sensible, (remember the four statements of different test data collections for failure measuring from above). Terms like *failure intensity*, MTTF- *mean time to failure,* MTTR- *meantime to repair,* MTBF- *meantime between failures* are important to understand since these are standard definitions that are used in software reliability (3).



When sampling failures you calculate the average number cumulatively at time *t* and derive thus acquired curve in order to extract the failure intensity for that moment. Failure intensity explains how the average number of found failures per time unit changes over time. Failure intensity is expressed as failures in a time interval and MTBF is the inverted relation.
In order to draw any conclusions from these different terms that later on are modeled, we pinpoint the importance of data analysis. How the data is measured. Is the time measured as calendar time, CPU-time or system in use and so forth? For example, the number of testers will make an influence on the test result if test data is expressed in calendar time. Questions and decisions like these have to be put into a context chosen for the specific intention of the measurement, which the reliability modeler must be aware of. Otherwise, the modeling is not able to say anything!
It is thus important to maintain a holistic thinking when model for software reliability.
The software reliability field offers different types of models to use for estimations. Regardless of what model you choose, (see section 2.3 for further information about the different models), you must set a variety of parameters in order to fit the model into the data. The reliability models use the, up until now, available data for this task. The most generally used methods for doing this, is the *maximum likelihood* and *least square* methods. See section 2.4 for a more thorough exposition of these techniques.
We have notified a distinction between the general usage and reliability engineering field usage of *qualitative forecasting* (see section 3.1). The software reliability engineering field uses, as mentioned, data currently available for parameter estimation but with the commonly usage, it is the expertise that is steering the curve shape (5). This means that software reliability engineering uses a mixture of expertise and historical data analysis for modeling reliability.
As the test activity is processed, test result-data is gathered for modeling. Data is modeled into a curve for visualization and we are able to draw conclusions from this curve of how well the reliability level meets the requirements. With a representative model, it is possible to foretell the extension of the data, i.e. the end part of the curve of your chosen model and by this; predictions can be made.
Problems related to data analysis such as knowledge and understanding of what the data and predictions actually mean is not considered in this experiment. Instead, we concentrate on technical aspects of problems related to how different prediction models perform.

1.2 An attempt to improve prediction modeling

TIE, *Time Invariance Estimation*, is a prediction model developed by Håkan Lennerstad and is still a research material. Our intention is to implement TIE (3) into a software, i.e. develop an application to use for software reliability, and find out if TIE could match the prediction models used in software reliability today.
The difficulty of previously mentioned characteristics in the field of reliability prediction, made us interested in investigating this area to see if a slightly different reliability modeling approach, like TIE, could be found useful. As earlier mentioned, the modeling most used to day, uses both expertise and available data for modeling reliability.

(Statement 1.2.a) *Expertise,* when deciding what type of model to use for a representative shape of model curve. (sec. 2.3)

(Statement 1.2.b) *Available data,* in order to map the chosen model with the data choice of parameter estimation for this purpose is needed.
Different techniques of parameter estimation is discussed in section 2.

Decisions taken within these two activities could lead to incorrect modeling and are therefore considered risky. With TIE, we assume that no decisions of these characteristics are necessary. Instead, TIE learns the historical behavior both in short term and long term. Moreover, since TIE also provides an estimation of the reliability corresponding to the predicted suggestion, i.e. for every prediction, an estimation of how reliable it is, is provided, we believe that utilizing this method could be very useful.
Our *research question* is;
Since TIE does not require decisions of earlier mentioned characterizations (statement 1.2.a, 1.2.b), we want to know if TIE is able to perform predictions that are more trustworthy than general reliability modeling and prediction.

1.3 Thesis contribution

The activities related to modeling data to fit a curve such as the decision of what model to use and parameter estimation are regarded as risky. These could jeopardize the modeling accuracy. By using TIE we can perform predictions that are not influenced by these activities.
An experiment is set up to test if TIE could be found useful as a software reliability prediction model by a comparison with the traditionally used prediction models.

1.4 Outline of thesis

To perform this experiment, we deepen our knowledge within the field of software reliability by literature research of different reliability methods and models used today and present which models to use for a comparison with TIE (section 2). A presentation of the theory of TIE is given (section 3) before the experiment is described (section 4). Finally, a conclusion of the experiment is presented (section 5).

2 BACKGROUND & RELATED WORK

After having a discussion of why to model software reliability, we introduce four separate approaches for accepted usage when achieving high reliability. We continue with an examination of the *failure forecasting* – approach, which include prediction modeling. After having presented some of the model types used in this area, we present *failure count models* which also are the types of models that we are focusing on in this thesis.

2.1 Why software reliability modeling

Along with continuously increasing usage of computers, the expected hardware and software response additionally increases our demand in general. One of the most important software attribute for this matter is software reliability. Different techniques of how to tackle this issue are developed with different approaches, se section 2.2.
First, we want to be able to *estimate* the reliability in order to *certify* the reliability level to meet the requirements. To estimate, we model the reliability. When the reliability is modeled, we are given an opportunity to *predict* the reliability using the model for prediction. This is a very urgent task since a prediction of a software's reliability can act as direct input to management decisions, see section 2.1.1.
As mentioned in chapter 1, the software reliability is connected to the expected usage. The expected usage must be captured in order to set up tests for the software and then test results can be modeled for estimation and prediction.

2.1.1 Trade-off, cost of development verses cost of failures in released software

One of the main issues concerning the cost of developing a product is associated with testing. The goal with testing is to make sure that the user is satisfied with the software. But testing costs money, developers that are bound with debugging cannot start working on new projects and software outside the market does not produce any income (8).
At the same time, the cost of repairing a failure is generally less during testing than operational use. In other words, it is important to conduct trade-off studies to aid the managing decisions of when to release the product. It is possible to use a cost model where different contra dictionary objectives are weighted to minimize cost (24). The most common way to model failures during a products operational life is: $f(x)=kx$, where $f(x)$ is the total cost of field failures, k is a constant and x is the failure intensity. In **Figure 2** you can see how the Total cost for failures varies depending on system testing cost and cost of failure in the field.

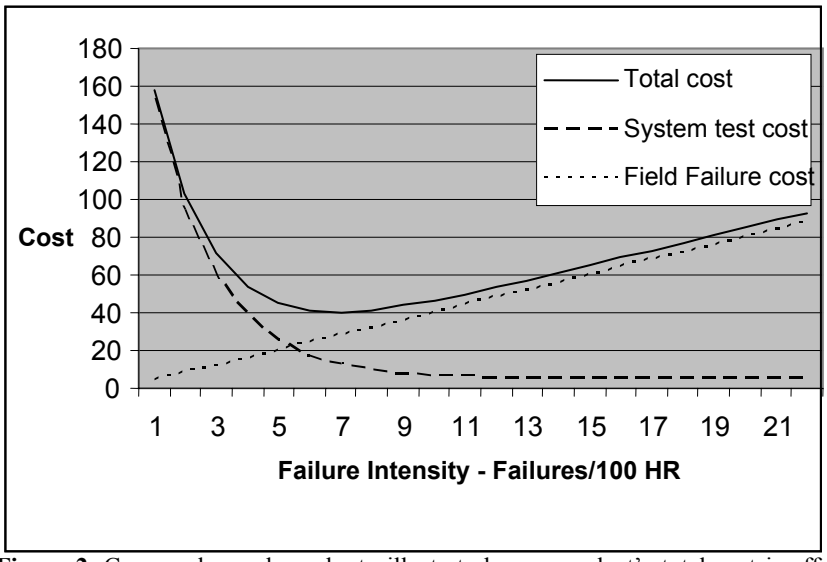


Figure 2. Commonly used graphs to illustrate how a product's total cost is affected by the cost of testing, system test cost and by the cost of failures occurring during the systems usage in the field, field failure cost.

2.2 Reliable system approaches

In the development process of a software product the literature outline four methods to achieve high reliability (8). Here is a short presentation of the methods:

Fault prevention – This field's goal is to prevent that the system developers introduces faults in the system during the development. It includes several different techniques. For example: writing clear code according to a standard, verification of system architecture and system design. During the 1990's new and more complete systems engineering processes have been invented; one is the *cleanroom model process,* with the focus of getting the code right from the start by quality certificate increments that accumulate into a system (11); another technique is *software reuse* with the idea to simply reuse whole, or parts of, working systems in new systems (8).

Fault removal – The goal is to find and eliminate faults in the system. This is a vast field that is included in all phases of the software development process. The general idea is that faults costs more and more the later in the software engineering process it is found. It is in other words best to find the faults during the requirement engineering phase and worst to find them during field usage. Fault removal incorporates some more or less sophisticated techniques such as *formal inspections* (14) in which a small group focuses on finding and correcting faults and then verifying the correction. Other techniques are blackbox- and whitebox-testing (26).

Fault tolerance – The system should never fail to deliver its service to the user even in the presence of faults. There are two separately, but not mutually exclusive main courses to ensure fault tolerance. The first is the single-version software environment, which include "monitoring techniques, atomicity of actions, decision verification, and exception handling" (8). The other course is the multiple-version software environment, in which separately developed versions of either all of, or parts of the software are used contemporaneous to ensure fault tolerance. These approaches include (8): the recovery blocks technique, the N-version programming technique (18), and the N self-checking programming technique.

Fault/failure forecasting – To estimate how much faults that exists in the software and their probability to cause failures. The main concepts is fault/failure relationship, understanding of the operational environment, collection of failure data, reliability models, the selection of the same and understanding of their results and finally the idea to use the results to make management decisions (8).

The field of fault/failure forecasting is the main focus of this thesis and some of its aspects will be described in more detail below.

2.2.1 Model types

The models that are applied in software reliability engineering can be separated into four classes that represent what they focus on (3). The first two are Time between failures models (sec. 2.2.1.1) and failure count models (sec. 2.2.1.2) which are both used in the fault/failure forecasting field and are closely related. The third model class is fault seeding models (sec. 2.2.1.3) which could also be used in fault/failure forecasting, but also in the field of fault removal. The fourth and last class is input-domain based models (sec. 2.2.1.4).

2.2.1.1 Time between failures models

Time between failures models are used in order to see how the reliability changes over time. The models are used to determine whether the failure intensity increase (reliability drops and the time between failures decrease), or decrease (reliability growth and the time between failures increase). Z. Jelinski and P. Moranda developed the first time between failures model in 1972, called Jelinski-Moranda model. The main idea is that when the total number of remaining faults decreases as the errors are eliminated, the program should be able to run longer before a new failure occurs. In other words, the failure intensity should decreases as time goes by (as illustrated in **Figure 3**).

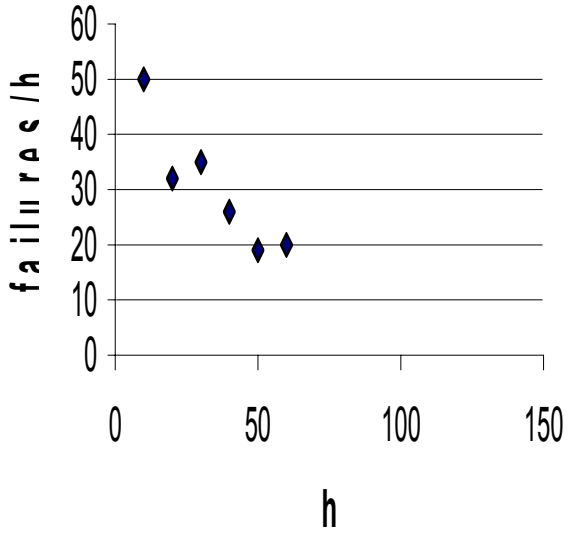


Figure 3. The dots represent the measured failure intensity at different times. It is possible to see that the trend of failure intensity decreases over time.

The procedure continues by model selections: one or several models are selected (for example the Jelinski-Moranda model). This is followed by parameter estimation (see sec. 2.4), that is done by using the, up to the current time, observed values. Finally, it is possible to plot how the failure intensity is believed to decrease (as exemplified in **Figure 4**).

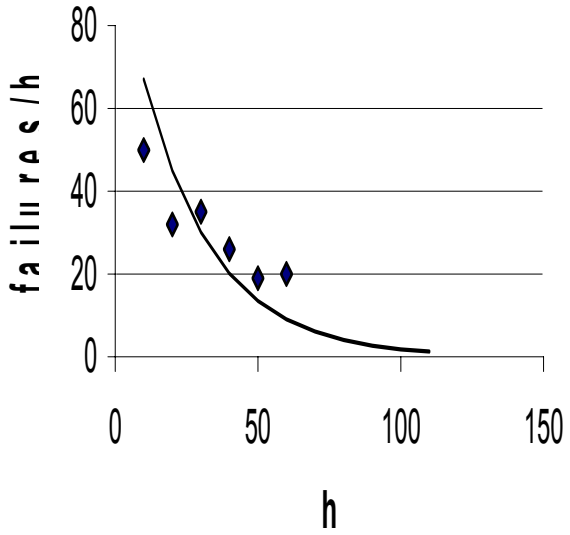


Figure 4. A model is selected and fitted to the dots. Now it is possible to predict how the failure intensity will change over time.

The Jelinski-Moranda model has some important assumptions (8 and 20):

- ◊ The initial number of faults is constant but unknown.
- ◊ The faults are independent and equally hazardous.
- ◊ Failures are independent of each other.
- ◊ One fault is removed when a failure occurs.
- ◊ Faults are removed instantaneously without inserting new failures.
- ◊ The failure rate is constant during a failure interval and it is proportional to the remaining faults.

In the years after the Jelinsik-Moranda model was introduced new variants have been proposed. These models include aspects such as faults do not need to be repaired until a major failure occurs, failure intensity varies between failures and finally imperfect debugging are accounted for (3).

2.2.1.2 Failure count models

Models that are based on the number of failures that occur in each time interval are considered as Failure count models. According to (3) the number of failures is modeled as a stochastic process, where $\mu(t)$ is the number of failures at time t. One subgroup of these models is the NonHomogeneous Poisson process (often referred to as NHPP) models first proposed by Goel and Okumoto in 1979 which has formed the basis for the failure count models (8). The idea is that the number of observed faults at time t is nonlinear because the failure intensity decreases when faults are found and corrected (3). In this thesis, we focus on the following failure count models. They are all included in the software reliability modeling tool used during the experiment described in sec. 4.

NHPP or Goel Okumoto model – This is as described above the first failure count model. The number of failures at time t is $\mu(t) = N(1 - e^{-bt})$ for some constant $b > 0$ and $N > 0$. N is the expected total number of faults and b is the "shape"-factor. The failure intensity is $\lambda(t) = Nbe^{-bt}$ which is equal to the derivate of $\mu(t)$. The NHPP model is sorted as a concave model (see **Figure 5**).

Schneidewind – This model incorporates the idea that the current fault rate might be of higher importance than the distant past. The number of failures at time t is $\mu(t_i) = \frac{\alpha}{\beta}(1 - \exp(-\beta i)$ for some constant α , β >0. And the failure intensity is $\lambda(t) = \alpha \exp(-\beta t)$.

Yamada S-Shaped – This model is a modification of the NHPP model that include the assumption of the start up problem. The idea is that early testing is not as efficient as later on. This because of the testers need time to familiarize them self with the test environment and so on. The error detection rate will after the start-up phase increase and then decrease as the faults remaining gets harder to find (see **Figure 5** for an example curve).

The number of failures at time t is $\mu(t) = \alpha(1 - (1 + \beta t)e^{-\beta t})$ for some constants $\alpha, \beta > 0$. And the failure intensity is $\lambda(t) = \alpha B^2 te^{-\beta t}$.

Generalized Poisson – This model is included in the software which we use in the experiment, but we have not found any information about its characteristics.

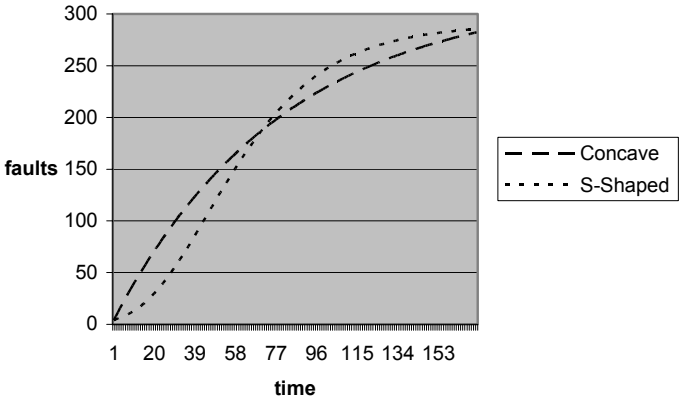


Figure 5. The two major classes of failure count models; the concave ones and the S-Shaped ones. Both assume that the defect detection rate decreases as the number of detected defects increase. The S-Shaped class tries to model start-up problems in testing.

2.2.1.3 Fault seeding models

By seeding errors to a document (for example: source code, requirements specification) and then let the document undergo testing of some kind it is possible to calculate how many "real" errors that exists. Some variant of this technique exists; Mills' error seeding model is the first method proposed in 1970. It founded this type of models and uses seeded, induced, errors to calculate the number of errors in the document (20); Cai's Model modified Mills' model by instead of seeding errors divide the software (or document) into two parts to *estimate* the number of defects remaining (20); Hypergeometric Distribution model is also an estimation model to find out how many faults initially exists in a program. By not removing the errors when found and keeping track of the already found errors and newly found errors for each *test instance* it is possible to estimate the number of faults in the program (20).

2.2.1.4 Input domain-based

By finding all unique paths through the program and then execute each and everyone it is possible to guarantee that everything is tested. This has in reality shown to be very hard to accomplish except for very small programs (8).

2.3 Evaluations of FC-models

Many articles have evaluated the use of Failure count models together with related issues. They often identify the problem of model selection and try to overcome this by the usage of a combination of models, or by using a technique called *recalibration*. Some of these articles are very interesting and some short summaries will be presented in the following subsection.

2.3.1 FC-models

"*Investigating a Specific Class of Software Reliability Growth Models*", by: Peter A. Keiller and Thomas A. Mazzuchi (27).
This article is a thorough analysis of several NHPP models and it also includes a new technique called trend determination, or reliability growth "window" technique. The evaluation uses 41 datasets which originates from several journals, technical reports and some books. They use six models in four procedures in order to test their idea. They conclude that it is not possible to select one of the procedures over the other, but they did find out that it was successful to use "…data analysis trend techniques to improve the predictability performance of the models on test." They also illuminate some important issues such as that it is better when a model overestimates than when it underestimates. A model can at most underestimate 100%, but it can overestimate without limit. They think that an underestimation is probably a more serious mishap than an overestimation.

"*Applying Reliability Measurement: A Case Study*", by Willa K. Ehrlich, S. Keith Lee and Rex H. Molisani (28).
The authors perform an experiment to see if it is possible to apply the reliability-measurement approach in real time. They use an existing data series consisting of test-failures found during a load test driven by an operational profile (sec. 1.1.2). They try to fit a NHPP model at several points (25%, 50%, 60%, 70%, 80%, and 100% of the data series) and validate the results. Their conclusion is that the NHPP model has predictive validity already at 60% into system test. Moreover, it is possible to "apply the reliability-measurement approach in real time if you test in a controlled environment". They also note that if a system is tested against a specific operational profile environment the measured reliability will differ in another, alternative, environment.
"*Model Validation Using Simulated Data*", by Swapna S. Gokhale, Michael R. Lyu and Kishor S. Trivedi (23).
The aim for this article is to validate a selection of models of software reliability. The authors believe this is possible by simulate the datasets instead of using existing real sets that are influenced by process and product characteristics. They construct 20 datasets from five software reliability models using rate-based simulation. Then they use the same five software reliability models on the simulated datasets and obtained by this model validation. They have also compared this validation with validation results obtained when using the datasets found in (8).
From the simulation datasets validation, they conclude that the parameter estimation is a very important aspect. Depending on what parameter estimation is used, i.e. *the least square* or *the maximum likelihood* (sec. 2.4), the performance of the models varies. After this, they conclude that they can observe similar behavior when using real data. Their final contribution to the field is that it is possible to describe a failure data set more accurately by simulate a profile using a combination of reliability models.

2.4 Parameter estimation

The most important thing after selecting the right model in software reliability modeling is the parameter estimation. This can be done in several ways and we have found that two methods are most represented in the literature including software reliability modeling tools (25). The first one is called *maximum likelihood* and the second is called *least square*.
The maximum likelihood estimates the parameters by solving a set of equations. According to (2) maximum likelihood is most useful for large sample sizes, but the equations that need to be solved are very complex.
The least square method estimates the parameters by trying to minimize the difference between the model and the sample data. According to (2) least square is best for small and medium sized samples. For more information about parameter estimation please see (2, 20).

3 TIME INVARIANCE ESTIMATION

3.1 Introduction

The Time Invariance Estimation (TIE) method is what is called a "technical prediction analysis", which is, in contrast to domain specific analysis where you put great importance to analyze the domain where the prediction data is taken, a pure mathematical analysis of the data.
Literature also describes these two contradictory methods as *Qualitative forecasting* and *Quantitative forecasting (*5, page 8-12*)*
Qualitative forecasting usually uses experts in order to predict future events. It also involves subjective choices of what forecasting method to use. The methods could for example be based on an s-curve or a concave curve (section 2, **Figure 5**). Then the technique of *subjective curve fitting* must be done, which means that parameters have to be set so that the data can be modeled with the chosen curve. These two tasks require expertise.
Unlike qualitative forecasting, quantitative forecasting involves analysis of historical data in attempt to predict future values.
TIE is basically constituted by three main activities;

- ◊ the *Modification* techniques
- ◊ a *Pattern Match* technique
- ◊ and the *Compromise* techniques.

Mainly, TIE searches for matching parts in a data series (sec. 3.1.2). The original series is modified into several series according to a variety of different techniques (sec. 3.1.1). This is done in order to find matches that are hidden and could be impossible or difficult to find by looking at or calculate from the original series in a graph. See **Figure 6** to **Figure 9** in section 3.1.1 for an illustrative example.
All matches that are found are saved and used for later predictions. How good a match is, is also depending on how well earlier similar matches have resulted. Thus, the TIE method learns as time goes by.
The main idea with TIE is to capture similarities in the past to build future predictions upon. And by building empirical functions of the similarities, TIE is able to learn how well the similarities are for predictions.
Below we give a brief presentation of the three pillars underlying TIE, i.e. modification-, pattern match- and compromise-technique with a final summary.
Note, that the complexity of TIE is rather high and there is no need for the reader of this thesis to have a complete understanding since the functionalities are wrapped up and concealed inside the TIE software.
As stated earlier, TIE is based on research material that is not yet published and we are therefore not able to refer to any specific source.

3.1.1 Modification techniques

In order to find similarities hiding in a series you can modify the series with a transformation technique. The TIE-method uses three techniques to perform this modification, the mapping-modification, the difference-modification and the moving average-modification.
Mapping-modification This modification technique uses one or several mathematical formulas like:

$$g(x_n) = \ln(x_n - x_{\min} + 1)$$

called mapping modifications, to modify each data value of the series into a new series. This modification is able to pick up the curves with shapes of for example exponential characterization.

Difference-modification calculates the difference between the data values in the data series to create a new series. Difference-modification can be calculated on data series already modified by this technique, into several "difference-levels". A series s = $s_1,...,s_n$ with a first level of difference-modification calculated (i.e. only one difference modification is calculated).

$$Ds = (s_2 - s_1,...,s_n - s_{n-1}) .$$

With difference modification, for example the increasing or decreasing tendency is captured. *Moving average-modification* is a frequency filtering method, often used in the work of signal processing. Different levels of "noise" that affects the series are eliminated in this technique. For illustration we exemplify how the *moving* value in a series is calculated into a new "average-value" in an"average-series".

$$[Ax]_j = (x_j + x_{j+1})/2 = 2$$

Whit help from these "average series", different frequency bands can be created and investigated according to the following principle;

$$x = \underbrace{x - A^{m_1}x}_{y_1} + \underbrace{A^{m_1}x - A^{m_2}x}_{y_2} + \underbrace{A^{m_k}x}_{y_3}$$

Where,

m = Number of how many calculated *moving average* are done on one specific series.

y_n = Elicited (created) seasons, trends and noise series originated from one series.

If there exist any regularity of different frequencies like season variations in the series, we are able to extract them with this technique.

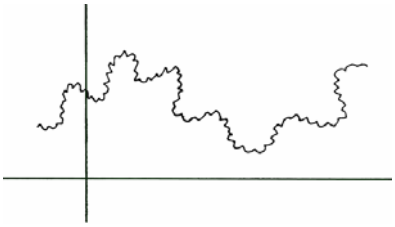


Figure 6. Example of a curve, A. Represented by x from formula above.

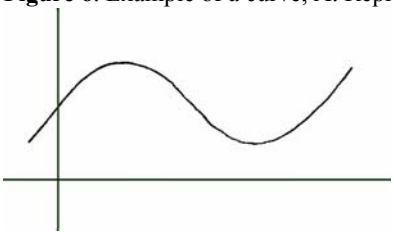


Figure 7. Season variations of curve A, when noise and trend is excluded. Represents y_3 from formula above.

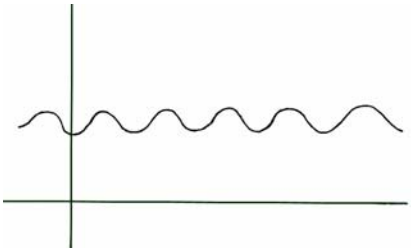


Figure 8. Trend variation (higher frequency than season) when season and noise are withdrawn from curve A. Represents y_2 from formula above.

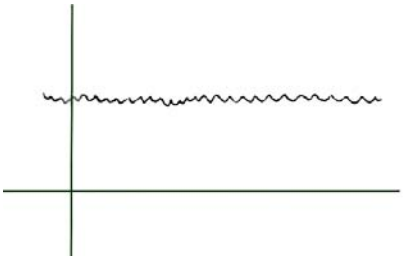


Figure 9. Noise extracted from curve A. Represents y_1 from formula above.

3.1.2 Pattern matching

The TIE method tries to find earlier parts in the series that matches a recent part of the series. These parts are tested in different sizes, i.e. the compared parts could be 1...n numbers long depending on how TIE is set up. Several different sizes, lengths, of the compared parts are investigated within one computation. Also, the *size of translation* in relation to the recent part is investigated.
If

$$\sum_{j=1}^{l} \left| x_{i+j} - x_{n-l+j} \right| = 0$$

TIE has found a perfect match.
l = the size of the part that are investigated.
i = translation, i.e. the i steps to the matching part.
n = the n'th data available, i.e. the last number.
This quantity is the *pattern match error*. The variation of a series s is measured by the quantity *MAD(x)*:

$$MAD(x) = \frac{1}{n-1} \sum_{j=1}^{n-1} \left| x_j - x_{j+1} \right|$$

MAD stands for *Mean Absolute Deviation.*
And by dividing the pattern match error with MAD(x), we normalize the pattern match error into, *e*. We have a match if $e < 1$.
The data values that follow directly after the matched part are considered as prediction values:

$$\tilde{x}_{n+1} = x_{i+l+1}$$

And the quantity:

$$\frac{1}{MAD(x)} \left| \tilde{x}_{n+1} - x_{n+1} \right|$$

is the normalized prediction error, pe when having access to real observed value, x_{n+1}.
With f (future) step prediction, we have the normalized prediction error;

$$pe = \frac{1}{MAD_f(x)} \left| \tilde{x}_{n+f} - x_{n+f} \right|$$

The prediction error is used for reliability estimation according to the following principle.

3.1.2.1 Prediction errors and reliability

The TIE-method increases the probability of the predicted values according to how well the part is matched and also by how well similar matches historically have performed. To do this, we build empirical functions *E*(p*e*) and *N*(p*e*) of the normalized prediction errors together with corresponding information from the matching part.
The empirical function represents the characteristics related to a specific prediction and its prediction error. The division of the prediction error, e, in separated intervals of $0 \le pe \le 1$, for example $0 \le pe < 0.1$, $0.1 \le pe < 0.2$ and so on together with the

information of what modification, translation, future etc is corresponding to a particular prediction error is how the empirical functions are created.
The information is;
d = differenced series,
m = moving average-series,
g = mapped series,
f = future steps
t = translation steps between the matching parts
l = length of matching part
pe = prediction error
And then we calculate the *reliability* of the related prediction by:

$$q = 1 - \frac{E(d,m,g,f,t,l,pe)}{N(d,m,g,f,t,l,pe)}$$

p =prediction, corresponds to $\tilde{x}$.
q = reliability, corresponding to an empirical function with its prediction suggestion.
N is the total number of prediction errors corresponding to set of parameters corresponding to the information above.
E is the sum of the same prediction error.
$0 \le q \le 1$ is only accepted.

3.1.3 Compromise Techniques

The compromise part of the TIE-method is composed of several compromise techniques. First, all the suggested predictions from each modified series are compromised by several stages into one suggestion per modified series. Similar to when the pattern match part saves prediction errors in empirical functions *E*(p*e*) and *N*(p*e*), the compromise part saves compromised prediction errors to evaluate the level of its correctness. We have a set of prediction suggestions (p_i, q_i) for each combination of *d, m*, *g* and other parameters.

3.1.3.1 Compromise over translation and length

Here we build empirical functions like:
$C(d,m,g,f,t,l,Q_0,Q_1,Q_{\max,}\alpha)$ and $N(d,m,g,f,t,l,Q_0,Q_1,Q_{\max,}\alpha)$
Where α acts as an instrument with weighted characterizations to consider how much the different reliabilities differs in the set of prediction suggestions. We calculate three different strategies of α , and for every α , we measure the number of high reliabilities by Q_0 . The contradictory degree of the predictions is measured by Q_1 and finally the presence of any good reliability is measured by $Q_{\max}$. The "Q – calculations" are considered to be outside the scope of this thesis and are for simplicity excluded and not further described here.
The reliability is as before, calculated with the build up empirical functions;

$$q = 1 - \frac{C(d,m,g,f,t,l,Q_0,Q_1,Q_{\max,}\alpha)}{N(d,m,g,f,t,l,Q_0,Q_1,Q_{\max,}\alpha)}$$

3.1.3.2 Compromise over moving average

After having compromised over translation and length, we are now left with *one* prediction for *every* modified series. The different combinations of *moving average* series, i.e. frequency band-combinations, is now overviewed and compromised before a final prediction suggestion. Using the frequency bands from section 3.1.1 above, we obtain the following predictions with related reliabilities;

$$p = \underbrace{pq^{\beta}}_{y_1} + \underbrace{pq^{\beta}}_{y_2} + \underbrace{pq^{\beta}}_{y_3}$$

β = is the factor that weights the prediction suggestions from the separated moving average frequencies by adopting different values like, 0, 1 and ∞.
We calculate three different strategies of β, and for every β, we save the reliabilities from the three different frequency bands in separate Q.

This compromise is calculated according to the same strategy as before;

$$q = 1 - \frac{C(d, m, g, f, Q_0, Q_1, Q_{\max,}\beta)}{N(d, m, g, f, Q_0, Q_1, Q_{\max,}\beta)}$$

3.1.3.3 Reconstruct modified prediction suggestions

The suggested predictions are related to the different modified series and are therefore not relevant to represent predictions to the original series. Before presentation of the suggested predictions, a reconstruction of the modified prediction must be done.
The *moving average* series predictions are assembled and reconstructed in the compromise over moving average stage, described in section 3.1.3.2 above, and no more reconstruction is needed.
Mapped series just inverts the mapping formula.
Reconstruction from *difference* series uses "earlier" difference series of which this difference series is built upon for reconstruction, see formula below. The difference reconstruction stage is done accordingly to the following formula;

$$\tilde{x}_{n+f} = g^{-1}\left(p_f^d\right) + \sum_{i=1}^{f-1}(-1)^{i+1}\binom{d}{i}\tilde{x}_{n+f-i} - \left(D^d x\right)_{n-d+f}, \text{ for } 2 \le f \le d$$

$$\tilde{x}_{n+f} = g^{-1}\left(p_f^d\right) + \sum_{i=1}^{d}(-1)^{i+1}\binom{d}{i}\tilde{x}_{n+f-i}, \text{ for } f > d.$$

d = difference levels
The formulas are quite complex and are included just to show how the difference modified predictions relate to each other.

3.1.3.4 Reliability of the final compromise

Now we are left with one prediction for each modification, (modified series). The final compromise is to extract the best prediction found from all the modified series into one final prediction suggestion.
We simply choose the prediction with the corresponding best reliability from the competing series.

3.1.4 Summary of TIE

First, we modify the series in several stages with three different techniques in order to perceive hidden patterns.
Then we trace similarities from the past in all the modified series which is what the predictions basically are constructed from. We save facts related to all the pattern matches that are good enough until the "real values" arrives. Then we build empirical functions based on the comparison of the prediction suggestions from the pattern match and let the empirical

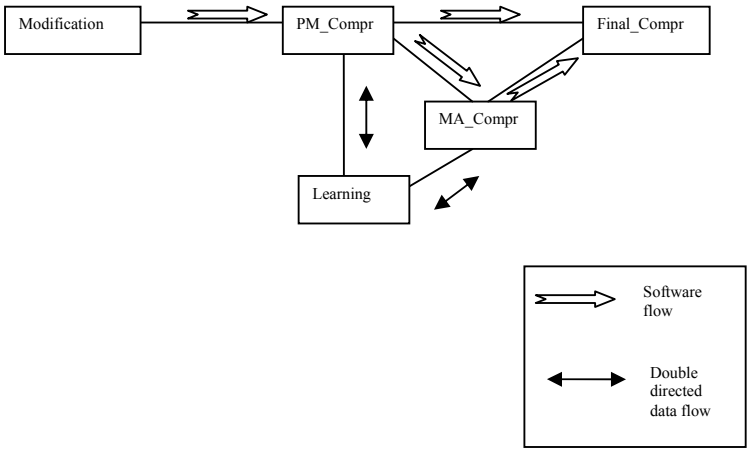
function represent the reliability of the prediction suggestion. In this way, we steer the compromise process based on reliability, i.e. results of earlier occurrence.
After having compromised the competing prediction suggestions according to the earlier described techniques, we again build empirical functions of how the compromise appears. Before the final compromise, the suggested predictions are reconstructed to represent predictions to the original series.
Finally, we just choose the prediction suggestion given with the highest reliability.

3.2 Software of TIE

In order to work with TIE we have developed a software program that implements the TIE algorithm. Below, we will describe some of our design decisions and experiences of this implementation.

3.2.1 Architecture and Design

We have chosen to delimit the abstractions that we found when learning the TIE-method out of the research material into a couple of components. The most important ones are the *modification-, pattern-match-, compromise-, and learning- components*. Unlike the theory of TIE, where the empirical functions are built-in to the pattern match and compromise techniques, we have chosen to extract these functionalities into one component - *learning*.



Throughout the design, we have tried to create those abstractions that we have used when learning and performing the TIE method in a non-computerized way.

3.2.2 Implementation

The theory of TIE has been challenged by a computerization of the same. The expected behavior did on several occasions fail to correspond with the theory. Therefore, the mathematical research material has been updated, as a consequence of this computerization.

3.2.3 Current version of TIE

The theory of TIE is not fully supported by this implementation.
For example, the order in which the different modification techniques can be executed is limited. As we believe, this limitation removes some of its possibilities and is further discussed in the analysis of the experiment and conclusion section.

3.2.3.1 TIE Settings

The user of the software can specify several setting alternatives order to achieve specific behavior;
Modification – In what order and which modification techniques to use; *mapping, moving average and difference.*
Window length – The maximum and minimum sizes, length, of the compared parts used in the *pattern match technique.*
Translation of the window – Intervals in steps between the compared parts used in the *pattern match technique.*

4 EXPERIMENT

4.1 Introduction and problem statement

Because of our generally increasing dependency of computers, software reliability has become one of the most important software quality attribute. To ensure software reliability, measurable data have to be sampled and used for certifying a reliability level meeting the requirements. Along with this, generally accepted terms and definitions are developed together with different approaches and techniques for usage specification and reliability estimation.
Software reliability modeling is used for reliability estimation and prediction. When modeling software reliability, a certain level of insecurity is related to how earlier measurements of expected usage are specified for test purpose, but also decisions taken when model the test results. The decisions made could lead to models that are defective.
As stated earlier, reliability predictions are of great management interests since they help us decide (sec. 1.2);

- ◊ When to stop testing.
- ◊ When to put to market.

These are questions that are strongly related to software reliability prediction.
Today's software reliability modeling requires a decision taken what model to use. This decision is often related to earlier experiences drawn from previous and similar projects. The models are mathematical formulas using parameters for mapping the model with the data (test result), i.e. the data available is used for parameter setting.
Both *choice of model* and *parameter setting* are activities that could expose the estimation and prediction with errors.
In the subsequent section we will define the experiment and its goal.

4.1.1 Experiment Definition

Object of study – The objects studied are Software reliability growth modeling and predictions.
Purpose – The purpose is to investigate if the TIE-model can serve as a software reliability prediction model with reduced risks of earlier mentioned characteristics (statement 1.2a and 1.2b, sec. 1.2) and if so, can it be proved to be better than the commonly used software reliability prediction growths models.
Quality focus – The quality focus is to improve the software reliability prediction phase in the software engineering field.
Perspective- The perspective is from the researcher's point of view.
Context – The experiment is run using ourselves as subjects, i.e. two undergraduate students and the experiment acts as the empirical study that our master thesis is based upon. The study is conducted as a multi-object study, with the reliability growth models described in section 2.2.1.2 and TIE (sec. 3) as treatments of chosen data sets to model reliability and perform predictions.

4.2 Planning

The experiment is taking place in a software engineering master thesis at Blekinge Institute of Technology. Two undergraduate students are conducting an experiment within the field of software reliability modeling and prediction. A literature study serves as an investigation method of which models to use in the experiment but also how previously experiments of the same have been conducted in research before. A new prediction model, TIE, still a research material, is developed to challenge the reliability models chosen.

4.2.1 Hypothesis

We conduct this experiment with the intention to give an answer to our research questions stated in section 1.2 and investigate whether the TIE-method can serve as a prediction model for this purpose. The hypothesis is stated as TIE is able to perform competitive reliability predictions with reduced earlier mentioned risks related to the experimentally compared reliability growth models.
This is formally described as the following formulas where H_0 is the null hypothesis, H_1 is the hypothesis, where μ_{Nold} = the "goodness" of the compared growth models and μ_{Nnew} = the "goodness" of the TIE – model:

$H_o : \mu_{Nold} > \mu_{Nnew}$

$H_1 : \mu_{Nold} \leq \mu_{Nnew}$.

The goodness of the predictions is measured with a technique known as: *mean absolute error* (MAE).

$$MAE = \sum_{t=1}^{T} \frac{|e_t|}{T}$$

e = error = observed number faults found at time t - predicted number faults found at time t,
T = time
The prediction data suggested from the different prediction models used in the experiment are the input data for this calculation.
This measure, MAE, is used as direct input for a competition, further described in section 4.2.2 below. The results from the competition constitute the material that we build our conclusion on.

4.2.2 Experiment Design

The experiment is set up as a competition between the chosen reliability prediction models (section 2.2.1.2) and TIE. A brief overview of the planned experiment is illustrated here;

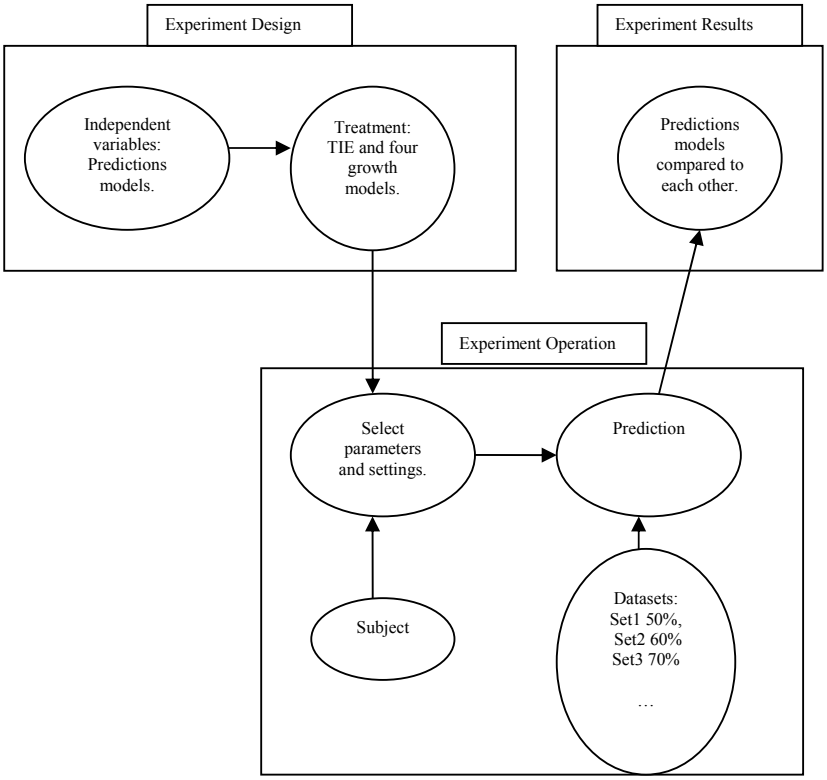


Illustration of the experiment.

Experiment Design

Independent variables – Are represented by the different prediction methods. These prediction methods act as treatments in the experiment.

Experiment Operation

The chosen reliability predictions are further divided in choice of parameter estimation. The two parameter estimation used in this experiment is discussed in section 2.4. TIE will be tuned according to two different settings choices. This is controlled by the subjects.

Subject – Two undergraduate students at Blekinge Institute of Technology.

We use three data sets provided by (8) which also are used in similar evaluations. Additionally, we use three test data sets provided by Ericsson AB.

Data sets - The three data sets provided by (8) are system test results and the three data sets provided from Ericsson AB are function test results.

All data sets are treated by all the different models with the different parameter estimations and settings (TIE). All data sets are further divided according to how much of the data that is available for prediction, i.e. 50%, 60% and 70% of the total data series. This division represents different stages in test phase.

This implies 10 different treatments (four prediction models with two parameter estimations and TIE with two settings) of 18 different data sets.

Setting 1: Mapping Modifications: Goel And Okumoto simple version.
Difference Modification: Level 1
Minimum, Maximum, Jump: 5, 40, 6
Translation: 4

Setting 2: Mapping Modification: Goel And Okumoto simple
Difference Modification: Level 1
Minimum, Maximum, Jump: 5, 40, 6
Translation: 3

Both settings alternatives are very similar, but they will find different matching parts since the Translation setting differs. In most cases the Setting 2 is more effective since it investigates more (see 3.2.3.1).

Experiment Results

The competition is divided into three installments according to how much of the data sets are available. For example, one competition with all treatments on all data sets when 50% of the data sets are available.

The treatment (model with chosen parameter estimation) with the lowest mean absolute error is, in this experiment, considered the best prediction model among the tested models (6). Some model may be better than others for early predictions and very bad for later ones. This is an important issue which the MAE calculation does not take into consideration.

Example: If two different models are used and these two modeled predictions together with the "real values" look like those in **Figure 10**, the plotted graphs of the errors from Prediction 1 and Prediction 2 compared to the "real values" will look like those in **Figure 11**. It is easy to see that the model that corresponds to Prediction 2 gives suggested predictions that are more accurate than Prediction 3 at first, but then seven steps into the future, the situation is reversed.

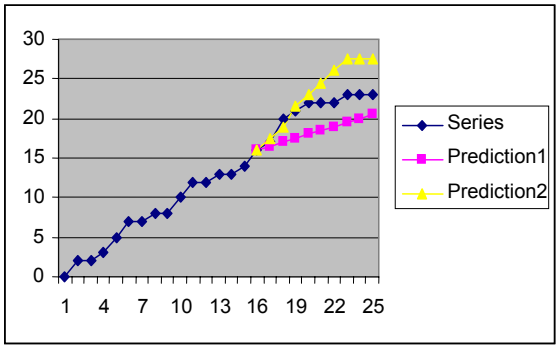


Figure 10 Here, predictions from Prediction1 and Prediction2 together with the "real" values are plotted.

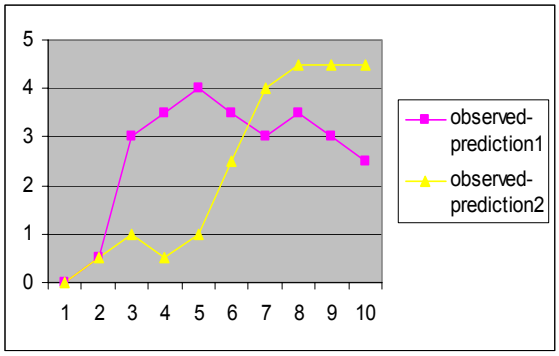


Figure 11 In this graph, the differences between the prediction suggestions and the "real" value are plotted.

Figure 11 shows that Prediction1 is closer to the observed values in the beginning than in the end and vice versa for Prediction2. Therefore, the results of the competitions for 50%, 60% and 70% data available are graded by four separate calculations.

4. A. *MAE - calculation for all prediction data*

(Two calculations after dividing the prediction data into two *equal* parts)

4. B. *MAE, one for first part of the prediction data.*

4. C. *MAE, one for the last part of the prediction data.*

4. D. *MAE, one for first prediction value available.*

The score is based on the median MAE - prediction error according to the following technique;
Example;
One model, *A1,* presents the median MAE of 10 for one data series. Another model, *A2*, presents a MAE of 30 for the same data series. A third model, *A3* presents a MAE of 5 for the same data series. Model *A1* will form the standard model since its MAE of 10 is the median value of the three models MAE. The best model of these three gets the score 5/10 = 0.5, the second best, gets the score 10/10 = 1 and the third best gets the score 30/10 = 3. This means that the median result serves as standard and the individual results from each treatment are taken into consideration when given score. The winner is of course the model that has the lowest total score.
The three competitions are further divided according to the four separated calculations, described above.
Dependent variables – These variables represent the output from each model. The output data is the predicted total number of faults found at a specific time. These output data is compared to how the real faults evolved.
The experiment result is presented in tables like this;

	DATA 1 when 50% data available			
Prediction model	100% pred.	first 50% pred.	last 50% pred.	one step pred.
....				
....				
....				

Figure 12. One table for each data series that is processed.
For each model used in this experiment the four MAE – calculations stated above is presented.
Finally, three tables of the total score from each part of the competition are presented.

4.2.3 Validity

There exist several threats to the experiment, some are general and will be found in most experiments, independent of the field, and some are specific for this experiment. We have listed those we have found most important and relevant to our experiment.
Conclusion validity – Threats to the ability to draw correct conclusions of the experiment.
Fishing – If the subjects are fishing for specific results. This risk may be extra high in this experiment since the subjects are the same persons as the designers of the experiment. Our single solution to this is to make sure that the experiment is well designed with strict control of how the experiment should be performed.
Data collections

- ◊ The fault data series used in the experiment would preferably originate from different types of projects and companies. But this is hard to accomplish since software companies generally treat their test results with a certain integrity level and are not very keen on sharing this kind of information. To lower the impact of this threat we include three extra fault data series from the book Handbook of Software Reliability Engineering (8).
- ◊ As stated in section 1.1.3, the importance of analyzing the data in order to have the modeling and prediction to make any sense, otherwise this could influence the experiments generalization ability. In this thesis we do not analyze the meaning of the prediction data we only compare the different models with each other.
- ◊ The results of the predictions made by TIE and by any chosen prediction model will only be as good as the data collection is (3). This is something we cannot control nor test.

Construct validity – Threats that are related to the design of the experiment.
Result calculation model compared to reality
The calculation of how well a model performs compared to the real observed values can be a serious threat. To meet this threat we investigate several ways to measure this comparing. We select the *Mean Absolute Error* since it is simple and it serves its purpose.
Result calculation model compared to model
The calculation of how the models perform compared to each other is very important, especially since this is the most important task with this thesis. There is several ways to do this:

- ◊ Simply give the best model 1 point, the second best 2 points and so one. The problem with this technique is that the models are exclusively scored after mutual order and no considerations are taken of mutual performance.
- ◊ Select the best model as standard and compare all the models with this one. The problem with this technique is that if one model is extremely accurate compared to the real values and thereby gets a MAE very close to 0 all the other models will get very high points.
- ◊ Select the mean value from all models MAE as standard and compare the models with this value. One possible problem with this technique is that if one or two models are very inaccurate and the others are accurate, the inaccurate will weight too much.
- ◊ Select the median MAE as standard and compare the models with this value. This technique is the one that we select. A model that is extremely inaccurate will get a high score, but it will not influence on the standard value and a model that is extremely accurate will get a low score.

Internal validity – Threats that are related to the operational part of the experiment.
Low experience – The subjects have no professional experience of using growth models in a software engineering context.

4.3 Experiment Operation

In this chapter we present the activities related to the operational part of the experiment.

4.3.1 Preparation

In order to calculate and keep track of every data constituted of prediction data, measured MAE-data, the score and median score, several excel-sheets is developed. The material is presented and attached according to the table in figure 7 in section 4.2.2 above.

4.3.2 Execution

The experiment is conducted during a period of seven days.
The prediction modeling by the challenged reliability growth models is performed using CASRE (25). All of these computed predictions are performed within less than 8 hours.
TIE software is the most time consuming instrument for this experiment. The execution of all the data series requires approximately 72 hours.
Parameter setting for TIE is chosen by experience from testing the software. Our intention is to include as much functionalities as possible, with respect to time limitation.

4.3.2.1 Data validity

One data series, provided by (8) is not interpretable by some of the challenged prediction model and is therefore decided to be excluded from this experiment.

4.4 Analysis

4.4.1 Discussion

The chosen score strategy to represent the capacity of the reliability models and TIE does not cover all perspectives. In this experiment the median value acts as standard. To enhance distinctions of other perspectives, the average value could for example act as standard.
TIE predictions are very time consuming. In principle, TIE is restricted to memory usage and time for execution. Together these factors are the main limits for how TIE is setup. The reliability prediction models used for comparison are done in a few seconds. *TIE could have performed better results if no restrictions concerning time existed,* which is a possibility that cannot be ruled out.
As stated in the introduction, TIE offers an estimation of the reliability corresponding to the predicted suggestion. This is not considered in this experiment.
The issue presented by (27), (sec. 2.3.1) concerning the differences of underestimates and overestimates is not taken into account when calculating for accuracy in this experiment.

4.4.2 Data Results

Here we present the result from the competitions that constitute our experiment in three different figures. Each figure shows the total sum for each model in four different views; *MAE* which is how the models performed over the complete prediction interval; *first half MAE 50/50* is how the models performed over the first half of the prediction interval; *last half MAE 50/50* is how the models performed over the last half of the prediction interval; *One step* is how the models performed in predicting one step into the prediction interval. (As stated before the experiment only compares the different models with each other.)
Figure 13 describes the score summary from the four competitions described in 4.2.2 when having 50% data available. Here we can see that TIE result is not very close to the rest of the start field, exception though for one step prediction where TIE shows very competitive results. It is possible to see if a model conduct similar results on all five data sets by reviewing the test data (appendix1) focusing on distinctions between the mean, median and maximum values for one model at a time. In appendix1 we can see that TIE setting1 has large distinctions especially on the last half MAE and on one step prediction. TIE setting2 has one large distinction on the first half MAE. Both Yamada and Scneidewind independent of parameter estimation have large distinctions on one step prediction.

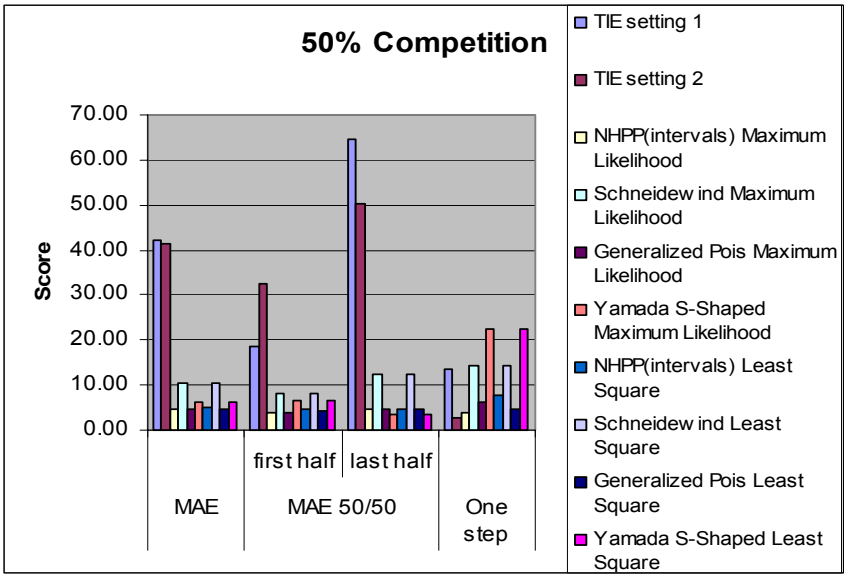


Figure 13, graph representing competition result from 50% data available.

After 60% data available you can see that TIE is approaching the rest of the start field (**Figure 14**). The long term predictions are better but still no competitive results. One step predictions are even better than when 50% data available. When searching for distinctions in the test data (appendix XX) it is possible to see the following: TIE setting2 has one large distinction on the first half MAE. Independent of parameter estimation both Schneidewind and Yamada has large distinctions, Schneidewind on MAE, last half MAE and on one step prediction, Yamada on first half MAE and on one step.

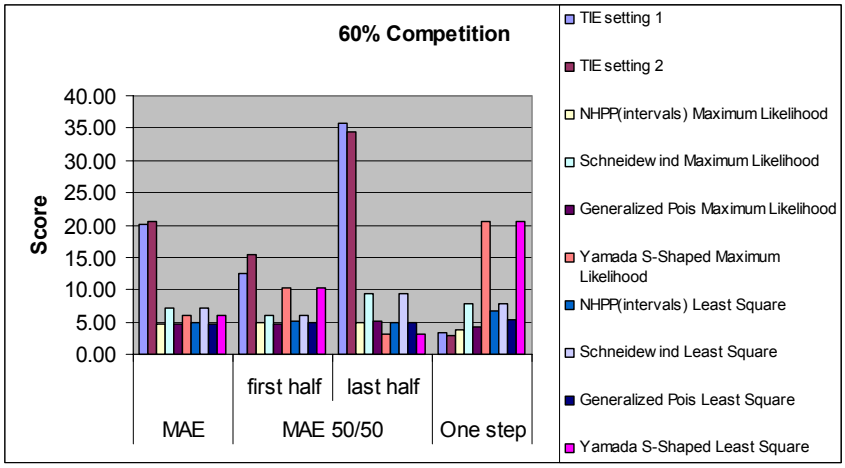


Figure 14, graph representing competition result from 60% data available.

After 70% data available TIE is closer to the other prediction models (**Figure 15**). The first half of the prediction has comparable score. When searching for distinctions in the test data (appendix XX) it is possible to see the following: TIE setting1 has large distinctions on the MAE and on the first half MAE. Independent of parameter estimation both Schneidewind and Yamada has large distinctions, Schneidewind on MAE, first and last half MAE and Yamada on one step.

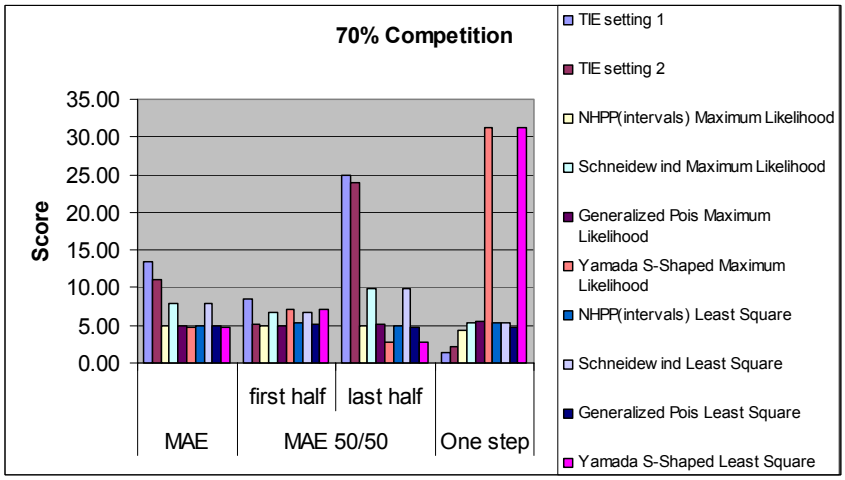


Figure 15, graph representing competition result from 70% data available.

To summarize the result, we can see how TIE is approaching better results when more data becomes available. This is also reflected by the percentage prediction size related to the data available. This indicates that TIE performance is independent of the data series. When looking at the other models it is observed that Schneidewind and Yamada both have large distinctions independent of parameter estimation and also which is more noticeable, the distinctions remain also when more data is used for parameter estimation. Of course, this could be caused by the fact that the other models including TIE setting1 and 2 becomes better faster…

Even though one step predictions from TIE generally are very good and comparable, TIE is still not quite adequate for software reliability prediction at this stage.

4.4.3 Hypothesis Testing

The H_0, null hypothesis cannot be rejected and thereby our assumptions cannot be proved.

5 CONCLUSIONS

This experiment has proven that TIE is not yet prepared for challenges within the reliability prediction context. But we noticed an improvement of the results from TIE along the experiment as more data became available which confirms that TIE cannot be rejected as an idea.
Therefore, with knowledge of the present limitations of the implemented TIE, assumptions of future success (when reduced present limitation) is not excluded.
The NHPP-model and Generalized Poisson model (2.3.1), independent of parameter estimation, have performed noticeable good results through out the experiment. This will serve as an additional conclusion to the experiment.

5.1 Future work

- ◊ Further development of TIE is required in order to provide functionalities that are excluded in the version used in this experiment. These functionalities are considered essential for reliability prediction (section 3.2.3).
- ◊ After a newer version of TIE is implemented it would be interesting to investigate the possibility of TIE to be found useful within other areas connected to software metrics and engineering.
- ◊ Due to our confidence in the general usability of TIE; in the future, secondary and additional efforts also are to test TIE within contexts other than reliability prediction and software metrics and engineering. The environment could be for example, weather forecast or medical informatics, such as children growth metrics, among others. However, this is not investigated in this thesis and left for future work.

6 REFERENCES

1. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, "Experimentaion In Software Engineering – An Introduction", Kluwer Academic Publishers, 2000, ISBN 0-7923-8682-5.
2. A. Wood, "Predicting Software Reliability", IEEE Computer, November 1996, pp. 69-77.
3. C. Wohlin, M. Höst, P. Runeson and A. Wesslén, "Software Reliability", Encyclopedia of Physical Sciences and Technology (third edition), Academic Press, to appear.
4. C. Wholin, lecture slides from the course: "Verification and Validation" at BTH (Blekinge Institute of Technology), 2002.
5. Bowerman and O'Connell, "Forecasting and time series an applied approach", Third edition, 1993.
6. R. Yaffee, "Introduction to Time Series Analysis and Forecasting", Academic Press INC., 2000, ISBN 0-12-767870-0
7. ANSI/IEEE, "Standard glossary of Software Engineering Terminology," STD-729-1991, ANSI/IEEE, 1991.
8. Michael R. Lyu, editor. "Handbook of Software Reliability Engineering" ISBN 0-07-039400-8 ,1996.
9. J.-C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Teminology" Proceedings of the 15th International Symposium on Fault-Tolerant Computing, 1985.
10. A. Avižienis and J.-C. Laprie, "Dependable Computing: From Concepts to Design Diversity", Proceedings of the IEEE, vol 74, May 1986.
11. R. C. Linger, "Cleanroom Process Model", IEEE Software, March 1994, pp50-58.
12. M. Dyer, "The Cleanroom Approach to Software Quality", John Wiley & Sons, New York, 1992.
13. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, 15(3), 182-211, 1976.
14. a. Aurum, H. Petersson and C. Wohlin, "State-of-the-Art: Software Inspections after 25 Years", Software Testing Verification and Reliability, 2002 (to appear).
15. A. Avižienis, "Fault-Tolerance: The Survival Attribute of Digital Systems", Proceedings of the IEEE, vol. 66, no. 10, October 1978.
16. B. Randell, "System Structure for Fault Tolerance", IEEE Transactions on Software Engineering, June 1975.
17. A. Avižienis, and L. Chen, "On the Implementation of N-version Programming for Software Fault-Tolerance during Program Execution", in Proceedings COMPSAC 77, 1977.
18. L. Hatton, "N-version Design Versus One Good Design", IEEE Software, November/December 1997.
19. J.-C. Laprie, J. Arlat, C. Béounes, K. Kanoun, C. Hourtolle, "Hardware and Software Fault-Tolerance: Definition and Analysis of Architectural Solutions", Proceedings of the 17th International Symposium on Fault-Tolerant Computing (FTCS17), 1987.
20. H. Pham, "Software Reliability", ISBN 981-3083-84-0, 2000.
21. Musa J, "Software Reliability Engineering", ISBN 0-07-913271-5, 1999.
22. Lennerstad H. "Prediction of Time Series by Time Invariance Estimation", to appear.
23. S. S. Gokhale, M. R. Lyu and K. S. Trivedi, "Model Validation Using Simulated Data", 1998 IEEE Workshop on Application-Specific Software Engineering Technology, 1998.
24. K. Okumoto and A. L. Goel, "Optimum release time for software systems", Computer Software and Applications Conference, IEEE, 1979
25. CASRE, Computer-Aided Software Reliability Estimation, NASA Cosmic, Jet Propulsion Laboratory.
26. T. Yamura, "How to Design Practical Test Cases", IEEE Software, November/December 1998, pp. 30-36.
27. P. A. Keiller and T. A. Mazzuchi, "Investigating a Specific Class of Software Reliability Growth Models", IEEE Reliability and Maintainability Symposium, 2002.
28. Willa K. Ehrlich, S. Keith Lee and Rex H. Molisani, "Applying Reliability Measurement: A Case Study", IEEE Software, 1990.