Staff Prediction Analysis
- Effort estimation in system test

Divna Vukovic
Cecilia Wester
This thesis is submitted to the Department of Software Engineering and Computer Science 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 10 weeks of full time studies.

Contact Information:
Authors:
Divna Vukovic
E-mail: divna@swipnet.se

Cecilia Wester
E-mail: cicci.wester@swipnet.se

External advisor:
Christopher Carlander
Symbian AB
Soft Center VIII
SE - 372 25 Ronneby
Sweden
Phone: +46 457 38 64 38

University advisor:
Claes Wohlin
Department of
Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
SE - 372 25 Ronneby
Sweden

Internet : www.ipd.bth.se
Phone : +46 457 38 50 00
Fax : +46 457 271 25
This master thesis is made in 2001 at Blekinge Institute of Technology and Symbian, which is a software company in Ronneby, Sweden.

The purpose of the thesis is to find a suitable prediction and estimation model for the test effort. To do this, we have studied the State of the Art in cost/effort estimation and fault prediction.

The conclusion of this thesis is that it is hard to make a general proposal, which is applicable for all organisations. For Symbian we have proposed a model based on use and test cases to predict the test effort.

**Keywords:** effort estimation, fault prediction, test time estimation
During the work with this master thesis we have come in contact with some persons we want to thank. These are:

*Christopher Carlander*, which has been our external advisor at Symbian.

*Symbian*, which gave us the opportunity to do this master thesis.

*Claes Wohlin*, which has been our advisor and have given us valued opinions during the thesis work.

Ronneby 01-06-08

Divna Vukovic

Cecilia Wester
1 INTRODUCTION

1.1 BACKGROUND ................................................................................................................ 7
1.2 PURPOSE AND GOALS OF THE THESIS ................................................................. 7
1.3 CONTEXT OF THE THESIS ....................................................................................... 7
1.4 READING GUIDELINES ............................................................................................. 8

2 SURVEY OF PAST WORK IN COST ESTIMATION ................................................. 10

2.1 1960’S .......................................................................................................................... 11
2.2 1970’S .......................................................................................................................... 12
2.3 1980’S .......................................................................................................................... 14
2.4 1990’S .......................................................................................................................... 15
2.5 2000 AND THE FUTURE .......................................................................................... 16

3 SURVEY OF PAST WORK IN FAULT PREDICTION ............................................. 18

3.1 WHAT IS A FAULT? .................................................................................................... 18
3.2 MODEL CATEGORISATION ...................................................................................... 18
3.2.1 Objective models ............................................................................................... 18
3.2.1.1 Within release models ....................................................................................... 18
   Design metrics ........................................................................................................ 19
   Inspection data ....................................................................................................... 20
   Test case data ........................................................................................................ 22
3.2.1.2 Between release models ................................................................................. 23
   SRET ...................................................................................................................... 24
   BBN ...................................................................................................................... 24
3.2.2 Subjective models ............................................................................................. 25
3.2.2.1 Direct models ............................................................................................... 25
3.2.2.2 Indirect models ............................................................................................ 25

4 DESCRIPTION OF SYSTEM TEST ........................................................................ 27

4.1 GENERAL DESCRIPTION ....................................................................................... 27
4.2 SYMBIAN’S SYSTEM TEST PROCESS ............................................................... 27
   4.2.1 Symbian’s defect categories ............................................................................. 28
5 PROPOSAL FOR ESTIMATION IMPROVEMENT

5.1 FAULT PREDICTION WITH USE OF USE CASES ................................................................. 29
  5.1.1 How use cases can be used to predict the number of faults ........................................ 30
  5.1.2 How long time does it take to correct a fault? ............................................................. 30
    5.1.2.1 Statistics to use ........................................................................................................ 31
    5.1.2.2 Direct estimation ..................................................................................................... 31
  5.1.3 Reused code VS. new or changed code ........................................................................ 31
  5.1.4 Example of the proposed model ................................................................................... 31
    5.1.4.1 Example 1 ............................................................................................................... 32
      Correlation between use and test cases ........................................................................... 32
      Correlation between test cases and number of faults ................................................... 32
      Defect categories .......................................................................................................... 32
      Time estimation .............................................................................................................. 33
    5.1.4.2 Example 2 ............................................................................................................... 33
      Correlation between use and test cases ........................................................................... 33
      Correlation between test cases and number of faults ................................................... 33
      Defect categories .......................................................................................................... 33
      Time estimation .............................................................................................................. 33
    5.1.4.3 Example 3 ............................................................................................................... 34
      Correlation between use and test cases ........................................................................... 34
      Correlation between test cases and number of faults ................................................... 34
      Defect categories .......................................................................................................... 34
      Time estimation .............................................................................................................. 34
    5.1.4.4 Example 4 ............................................................................................................... 35
      Correlation between use and test cases ........................................................................... 35
      Correlation between test cases and number of faults ................................................... 35
      Defect categories .......................................................................................................... 35
      Time estimation .............................................................................................................. 35
  5.2 CAPTURE – RE-CAPTURE ............................................................................................... 36

6 CONCLUSION ......................................................................................................................... 38

7 RECOMMENDATIONS ............................................................................................................. 39

8 REFERENCES ........................................................................................................................... 40
1 Introduction

1.1 Background
Resource estimation is an important activity of the project manager each time a new project is started and during the whole lifecycle of the project. The most important resource in software development is the staff. To be able to know how many employees to devote to a task, the project manager must be able to estimate how much effort it takes to complete the task and in what time the task has to be finished.

Many models to predict the effort have been proposed during the years. Most of them are concentrated on the whole development process, some using part estimation. The need to achieve a good prediction on phase level is necessary and can be achieved in different ways.

1.2 Purpose and goals of the thesis
To improve the quality of the developed product almost all development processes have support for a test phase, which is described in chapter 4. Test is an important part of the development, where developers have a chance to find and correct faults before the product is released. This phase must be planned as all other phases within the process. This can be hard since you do not know how many faults you will encounter and how hard they will be to correct.

The purpose of the master thesis is twofold:

a) Find a suitable prediction and estimation model for the system test phase.

b) Describe the actions Symbian needs to take to be able to use it properly.

1.3 Context of the thesis
This is a master thesis in Software Engineering at Blekinge Institute of Technology. The work has taken place at Symbian, Ronneby. Symbian is a software company with offices in London, Cambridge, Tokyo, Ronneby and San Francisco. They develop an operating system, called Symbian OS, formerly known as EPOC, for wireless communication.

Symbian’s last project was delivered within a timeframe of two weeks after the agreed delivery date. However, to reach that date they had to defer defects from the development project to maintenance projects.
If they had a model for estimating the number of defects they could expect during system test, they could more easily plan resources needed for test and defect prevention and correction, which in turn would lead to a more correct estimate of the "real" end date.

Nevertheless there are actions Symbian needs to take to improve processes and documentation needed for such a prediction model.

1.4 Reading guidelines

The first chapter of this master thesis has provided an introduction to this thesis and should be read first to give the reader an idea of what the thesis is about.

In the next two chapters we try to describe the State of the Art within the field of cost/effort estimation and fault prediction. In the fourth chapter we give a short overview of system test and the system test procedure at Symbian. These three chapters, 2-4, can be read in any order, but should be read before the remaining chapters.

In chapter 5 we propose a model to solve the estimation problem formulated in chapter 1. This chapter also includes some examples to illustrate the use of the proposed model.

Chapter 6 and 7 is the final chapters and give a general conclusion and specific recommendations to Symbian.

An outline of the chapters is presented in Figure 1.1.
Introduction

Chapter 1
Introduction
The background and purpose with this master thesis is presented.
Page 7-9

Chapter 2
Survey of past work in cost estimation
A presentation of different cost estimation models
Page 10-17

Chapter 3
Survey of past work in fault prediction
A presentation of different fault prediction models
Page 18-26

Chapter 4
Description of system test
Gives a description of system test (general & Symbian)
Page 27-28

Chapter 5
Proposal for estimation improvement
A proposal of 2 different test effort estimation models
Page 29-37

Chapter 6
Conclusion
The conclusion of this master thesis
Page 38

Chapter 7
Recommendation
Our specific recommendations to Symbian
Page 39

Figure 1.1 Outline of chapters
2 Survey of past work in cost estimation

The cost of a project depends on many factors where software, hardware and human resources are the main parts. Since human resources are the largest cost and hardware and software only are minor parts, the effort estimation is the base for most cost estimation models.

Cost estimation models have been around since the 60’s and can be split into 7 different types. These types are [Boehm81]:

- **Algorithmic models**
  algorithmic models are a group of models, which provide one or more algorithms to calculate the cost estimate as a function of major cost drivers. One example of such a model is COCOMO, see section 2.2.

- **Expert judgement**
  expert judgement is a method where experts are consulted. This can be done with help of the Delphi technique, see section 2.1.

- **Analogy estimation**
  analogy estimation is a model, which uses data from past projects to make an estimation of the actual costs, see section 2.4.

- **Parkinson’s principle**
  “Work expands to fill the available volume”, which means that all resources will be used by the project if they are available. For example: if you have a project, where phase 1 and 2 are estimated to 100 hours each, see Figure 2.1 A. To complete the work in phase 1, 80 hours are required. This means that the remaining 20 hours could be used in phase 2, but according to Parkinson’s principle these 20 hours will be consumed in phase 1. To finish phase 2, 120 hours are required, 20 hours more than estimated. This means that the project will be delayed by 20 hours, which had not been necessary if the time in phase 1 was spent better, see Figure 2.1.

- **Price to win**
  the price to win estimation is fitted to the assumed cost of winning the contract.

- **Top-Down Estimation**
  in top-down estimation an estimate for the whole project is made and this estimation is split up between the different subparts.

- **Bottom-Up Estimation**
  in bottom-up estimation is each subpart estimated and then added to the total estimation.
No model is in general superior to the others [Boehm81]. Anyway, Parkinson’s principle and Price to win should be avoided since they do not produce any good cost estimates. All of the other mentioned models have their strengths and weaknesses [Boehm81].

The algorithmic model and the expert judgement method are complementary, so are the bottom-up and top-down models.

Boehm [Boehm81] mentions that it is important to combine different models, and evaluate and iterate the estimates to gain the best result.

Here follows State of the Art within cost estimation and it is presented in chronological order.

2.1 1960’s

Significant research on software cost modelling began with the extensive 1965 study of 104 attributes of 169 software projects performed by System Development Corporation, SDC, for the U.S. Air Force. This led to some useful models in the late 60’s and early 70’s [Boehm et al. 00].

In the 1940’s the Delphi technique was developed [Boehm et al. 00]. This expert judgement method was originally used for making predictions about future events. The technique is now used as a guideline for a group of experts to come to an agreement. This technique is based on that participants formulate some estimation regarding an issue, without consulting the other participants. The results are then collected, put into a table, and then returned to each participant for a second round. During the second round the participants are again asked to make estimations regarding the same issue, but this time with knowledge of what the other participants did in the first round. The second round usually results in a reduction of the range in estimations by the group, pointing to some reasonable middle ground regarding the issue of concern.

Frank Freiman developed the PRICE model [Stutzke96]. Freiman saw hardware development and production costs as a process controlled by logical interrelationships.
between some cost variables. Freiman derived a set of algorithms that modelled these relationships. The algorithms were not released in the public domain, although a few where published in [Park88].

### 2.2 1970’s

In 1970’s the need for better cost and schedule predictions became important and the research effort increased. The models in the 70’s concentrated on new development, since the programming languages did not support reuse.

In 1977 F. Freiman and Dr. R. Park present the first commercially available computerised cost estimation model [Stutzke96]. This model is called PRICE-S and is a modification of PRI CE to suite software. The model was modified and re-validated in 1987. It is an algorithmic model. PRICE-S can be used to estimate selected parts of a software project, or to in detail estimate the entire project, including all development, modification, and life-cycle costs. It also provides sizing applications that makes it easier to determine the size of the project to be estimated.

In the end of 1970 Barry W. Bohem developed the COnstructive COst MOdel, COCOMO, an algorithmic model. The COCOMO model is based on inputs related to the size of the resulting system and different cost drivers [Fenton&Pfleeger96]. The development effort, E, in person months, is calculated by:

\[
E = aS^bF
\]  

where

- \(S\) is the size measured in thousands of delivered source instructions.
- \(F\) is an adjustment factor.
- \(a\) and \(b\) are constants.

There exist three different modes of COCOMO:

- **Organic system**
  - an organic system involves data processing, for example banking systems.

- **Embedded system**
  - an embedded system contains real-time software, for example water temperature sensing system.

- **Semi-detached system**
  - a semi-detached system is between the organic system and the embedded system.

More about COCOMO can be found in [Boehm81].

The most common models at this time were the algorithmic models, like COCOMO, which is mainly based on size metrics.

At the end of 1970’s A. Albrecht and J. Gaffney developed the Function Point Analysis (FPA), also an algorithmic model, but mainly based on Function Points (FP).
Function Points are a size metric, which lists and counts the number of user input, enquiries, outputs and masterfiles to be included in the resulting system. Technical Complexity Adjustment, TCA, is determined by estimating the number of influence of fourteen ‘general application characteristics’, for example data communication, performance and re-usability [Symons91], see upper part of Figure 2.2.

Function Points Analysis differs from COCOMO primarily in the way the size of the system is estimated. COCOMO uses lines of code while FPA uses Function Points, which is based upon the various ways users interact with computerised systems.

Function Points are calculated by multiplying the size with the technical complexity, see lower part of Figure 2.2.

Two models based on theoretical grounds were also developed at this time.

One was developed by L. H. Putnam called the Software Lifecycle Model (SLIM) [Stutzke96]. The SLIM model is based on the Rayleigh curve shown in Figure 2.3, which is a way to model project personnel level versus time, and empirical results from 50 U.S. Army projects. This original version can not handle for example incremental development. A clear description of this model is not available in the literature.

M. H. Halstead [Stutzke96] developed the other model. He defined size in terms of the number of operators and operands identified in a program, for example the FORTRAN statement: A(I) = A(J) has one operator (=) and two operands (A(I) and A(J)). From these operators and operands Halstead defined program level and difficulty. The program level and difficulty are then used to estimate the length of the program and the development effort. This size metric is nearly impossible to obtain before the project starts, since a good understanding of the design is needed.
2.3 1980’s

The models created in the 1970’s were improved and many became computerised. Among these was COCOMO, which became one of the most used models in the latter half of 1980’s.

When DoD, Defence of Department, introduced the ADA programming language, COCOMO was adapted to it and an ADA COCOMO was developed [Boehm&Royce89]. This model is also able to handle incremental development. ADA COCOMO is only mentioned as a variant of COCOMO and information about its development and use is hard to find.

Jensen made a software development schedule/effort estimation model, which integrates the effects of any of the environmental factors, such as resources, strategies, tools and methodologies and personnel capability and experience, impacting the software development cost and schedule [Jensen84]. This model is similar to Putnam’s SLIM. Jensen’s software equation relates the effective size of the system and the technology used in the implementation of the system. This model is now sold as the Software Estimation Model, SEM [Stutzke96].

In 1988 Charles Symons introduced Mark II Function Points. This was an attempt to improve the correctness of the Function Point metric mainly for projects with high technical complexity. Symons found that Albrecht’s and Gaffney’s Technical Complexity Adjustment (TCA) was sound and only had to be modified in two ways, by adding five complexity factors and allow for the possibility to vary the relative weights of the complexity adjustments [Symons91]. The five technical complexity factors may be extended by the organisation employing Mk II function point analysis [UKSMA98].
Mark II Function Points are calculated by multiplying the size with the technical complexity, see Figure 2.4. The difference from Function Points is that 19 or more general application characteristics are used, instead of the 14 proposed by Albrecht and Gaffney.

\[
\text{Function Points} = \left( \text{“Information Processing Size”} \right) \times \left( \text{“Technical Complexity Adjustment”} \right)
\]

Figure 2.4 Components of the Mark II Function Points

2.4 1990’s

A diversity of different software development processes is used and new or improved estimations models are needed.

Among the efforts for updating old models is an attempt to update COCOMO. The COCOMO model is extended to cope with the new type of programming languages and the reuse of software. This new model is called COCOMO 2.0 [Boehm et al. 95]. COCOMO 2.0 has three submodels:

- **Applications Composition**
  this submodel is used to estimate effort and schedule.

- **Early-Design**
  this submodel involves the exploration of alternative system architectures and concepts of operation. It is based on Function Points.

- **Post-Architecture**
  this submodel estimates the entire development life-cycle, when the top level design is completed.

A new form of models has also been developed; these can be classed in a group called non-algorithmic models. Analogy is one of the earliest non-algorithmic models; others are based on neural networks, see below [Shepperd&Schofield97].

Estimation by analogy is a form of Case Based Reasoning, CBR [Shepperd&Schofield97]. Cases are abstractions of events limited in time and space. The main activities in analogy are to identify new cases (problems), find similar cases, use knowledge from previous cases...
and find a solution for the new case. Analogy tries to avoid the use of expert judgement in form of human resources.

Neural networks are estimation models that can be trained using historical data, see Figure 2.5, to produce better results by automatically adjusting their algorithmic parameter values [Boehm et al. 00]. Neural networks is inspired by biological neural networks. A neurone generates an output from numerous inputs with help of thresholds. This output is then input to other neurones and this continues until an output is found.

![Neural network estimation model](image)

Figure 2.5 *A neural network estimation model*

### 2.5 2000 and the future

A lot of research in the estimation area has been done, but in our opinion there is still no maturity. There is no standard for which measures to use and more research must be done in the area of validation of metrics.

All estimation models must stay up to date since the software engineering discipline is evolving, because technology changes fast.

COCOMO II has for example been updated with the use of Bayesian analysis, see section 3.2.1.2, [Boehm et al. 00]. The Bayesian approach permits the use of both data and expert judgement in a logically consistent manner in making inferences. Bayesian analysis has all the advantages of standard regression and it includes expert judgement. It also attempts to reduce the risks associated with imperfect data gathering.
Research also continues in the non-algorithmic area with case based reasoning, see section 2.4 for description, neural networks, see above, fuzzy logic, regression trees and rule induction, for more information see literature [Schofield98].

To be able to make a suggestion for improvements in effort estimation and fault prediction we had to look at the State of the Art in these areas. In this chapter we have described effort and cost estimation models and in the following we describe some fault prediction models.
3 Survey of past work in fault prediction

The prediction of faults has proven to be a huge area and we try here to categorise different kinds of fault prediction models. We divide the models into two major groups with subgroups. But before we do that, a definition of error, fault and failure will be presented [Fenton&Pfleeger96], since we have noticed that Symbian does not follow the common terminology.

3.1 What is a fault?

Error
An error is a human mistake. An error, if encountered or detected, results in a fault.

Fault
A fault is the result of a human error. The fault can be present in requirement specification, design or code. A fault can be synonymous with "bug".

Failure
A failure is a lack of correct performance of the software. The cause of a failure is a fault. For example: if the design fault is implemented and first discovered when the code is executed.

3.2 Model categorisation

The two major groups of fault prediction models are objective models and subjective models. The objective models give an independent view of the prediction, while the subjective model is more subjective and relies on human intuition.

3.2.1 Objective models

The objective models can be divided into two subgroups, the models which makes a prediction within the same release and the models which makes a prediction between two or more different releases of the same product.

3.2.1.1 Within release models

A software development process can often be defined in different phases, such as analysis, design, coding and testing [Sommerville01]. To be able to predict the number of faults within a release, different documents from the various phases are used. From the analysis phase, the requirement specification can be used, from the design phase, the design documents can be used and from the code, different code metrics can be used.
Often the fault prediction is needed early in the lifecycle, so the predictions from requirement specification and design are most useful.

*Design metrics*

There exist two kinds of design metrics. The first is the metrics developed for the traditional functional decomposition and data flow development methods. The other is metrics developed for object-oriented (OO) development. The OO metrics can not be applied on the traditional development methods, but some of the traditional metrics can actually be used on OO development [Rosenberg & Hyatt97].

Many of the design metrics are used for predicting the quality of the software, but none of them directly addresses the fault prediction. The design metrics that predicts the general quality of the software often shows on classes or methods that will be hard to test. These classes or methods does often limit the reuse possibility, and are likely to be error prone or hard to maintain.

Design metrics can be used for predicting which modules or classes that will be fault prone. Some OO design metrics are presented in Table 1 [Rosenberg & Hyatt97].

**Table 1 Object Oriented Design Metrics**

<table>
<thead>
<tr>
<th>Metrics Name</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted Methods per Class (WMC)</td>
<td>WMC measures the complexity of an individual class, or if the complexity for all methods is equal, the number of methods in a class. The complexity of a method is measured by cyclomatic complexity.</td>
<td>This metric is a predictor of how much time and effort that is required to develop and maintain the class.</td>
</tr>
<tr>
<td>Depth of Inheritance Tree (DIT)</td>
<td>The depth of a class is measured by the number of ancestors of a class.</td>
<td>The higher the DIT is the more complex is the design, but the more code can be reused.</td>
</tr>
<tr>
<td>Number of Children (NOC)</td>
<td>This is the number of direct children of a class.</td>
<td>The more children, the more the class can influence the design. A high number of the NOC value predicts that the code requires more testing, but increases the reuse.</td>
</tr>
</tbody>
</table>
### Metrics Name | Description | Use
--- | --- | ---
**Coupling Between Object (CBO)** | A class is coupled to another class if it uses its member functions and/or instance variables. | The larger CBO, the higher the sensitivity to changes in other parts of the design. This gives a system that is harder to maintain. |
**Response For a Class (RFC)** | RFC is the number of methods that can be invoked in response to a message sent to an object of the class or by some method in the class. | The larger RFC, the more complex is the class. This affects testing, understandability and maintainability. |
**Lack of Cohesion on Methods (LCOM)** | LCOM uses data input variables or attributes to measure the degree of similarity between methods. | High cohesion indicates good class subdivision. Low cohesion indicates high complexity. |

A validation of these metrics [Basili *et al.* 96] has shown that all except the lack of cohesion on methods (LCOM) is significant to the fault proneness of a class. Among the metrics, especially DIT and RFC are very significant. The NOC metric has also shown very significant, except in cases of user interface, UI, classes. CBO is highly significant for UI classes.

**Inspection data**

A description of Inspections in general is outside the scope of this report and the interested reader is referred to Gilb’s description [Gilb&Graham93]. Inspections are only discussed in the light of fault prediction.

Inspections are used to ensure that the requirement, design and even code are of satisfactory quality. Studies show that faults often originate in the software design phase [Eick *et al.* 92]. A way to solve this problem is to apply an Inspection to the documents and collect and analyse data from the Inspection. This collection and analysis can be done in two ways, either by Capture – Re-capture [Eick *et al.* 92] or curve fitting [Wohlin&Runeson98].

**Capture - Re-capture**

Capture - Re-capture [Briand *et al.* 00] has been used in biology to be able to estimate the animal population. Animals are captured, marked and released on several occasions. If a marked animal is captured once more it is said to be re-captured. Based on the number of re-captured animals the total population can be estimated.
This method can be used on Inspections. There exists many different ways to calculate prediction. Below we present one of them.

Each inspector finds a number of faults in a document. A fault discovered by one inspector and re-discovered by another is said to be re-captured. Based on this the total number of faults can be estimated according to the formula below (2) [Eick et al. 92].

\[
\left(1 - \frac{n}{N}\right) = \prod_{j=1}^{m} \left(1 - \frac{n_j}{N}\right)
\]

(2)

where

\[N\] is the number of faults in a document being reviewed.
\[n_j\] is the number of faults that the \(j\):th reviewer has found in the preparation.
\[m\] is the number of reviewers.

For the case when \(m=2\), which means that there are two inspectors, the formula will be like (3). \(n_1\) is the faults found by inspector one, \(n_2\) the faults found by inspector 2 and \(n_{12}\) the \('re-captured\' faults found by both inspector 1 and 2. \(\tilde{N}\) is the number of predicted faults in one inspection.

\[
\tilde{N} = \frac{n_1 n_2}{n_{12}}
\]

(3)

According to Briand et al. [Briand et al. 00] Capture – Re-capture models tend to underestimate the number of faults in a document. This is especially true when using less than four inspectors.

\textbf{Curve Fitting Models}

Curve fitting models [Wohlin&Runeson98] use Inspection data to estimate the remaining number of faults. The data used is the information of how many inspectors that find a particular fault. The data is sorted and plotted, according to a criterion, and a mathematical function is used to predict the total number of faults. The function differs depending on which curve fitting model that is used.

Wohlin and Runeson [Wohlin&Runeson98] suggest two exponential models to fit the inspection data, one decreasing and one increasing. The decreasing model gives the lowest mean error and provides a stable estimate, while the increasing model can be used for predicting the worst case.

According to Thelin and Runeson [Thelin&Runeson00] curve fitting models were introduced as a complement to Capture – Re-capture models.
Test case data

The testing effort can be calculated using Test Points [Broekman98]. This is a measure of how large the system is. Test Points are based on Function Points, see section 2.2, which means that the use of Function Points are necessary if you want to use Test Points. Test Points are only applicable to black box testing, such as system test and acceptance test. The literature does not mention why and no additional literature can be found about this model.

Test Points are calculated by direct and indirect calculations and cover size, risk and strategy of the system, see Figure 3.1. Direct Test Points are related to test activities performed while testing the system, while indirect Test Points are related to documents and processes.

The direct Test Points are calculated with

\[ \Sigma (FP_f * T_f * Q_d) \]

where

- \( FP_f \) is the number of function points per function and is calculated according to Function Points, see section 2.2.
- \( T_f \) is the test impact per function based on four characteristics: importance, influence, complexity and uniformity. The calculation of these characteristics is based on the method of counting Function Points in FPA.
- \( Q_d \) is the direct Quality attributes factor, which contains 4 explicit attributes and 4 implicit. The explicit attributes are functionality, security, suitability and performance, while the implicit are user friendliness, resource usage, performance and maintainability. Each explicit attribute is scored and weighted. For the implicit attribute, defined values are provided, which are added to the explicit sum.

The indirect Test Points are calculated with

\[ (Qi * FP)/500 \]

where

- \( Qi \) is the indirect quality attributes factor; these are flexibility, testability, security, continuity and traceability. If any of these attributes will be tested, the weight will be 8 for each attribute tested. \( Qi \) is the sum of the tested attributes. For example: You want to test flexibility and security. This will result in \( Qi \) being equal to 16.
- \( FP \) is the total number of Function Points of the system.

In the literature [Broekman98] this formula is not explained further.

When the Test Points are calculated, conditions and productivity factors are taken into account to calculate the primary test hours, which is the time it would take to test the
system without management overhead. The productivity factor is the time it takes for a tester to perform an activity, which depends on his skills and experience. The conditions factor is external conditions which influence the tester, for example test tools, quality of system documentation and the test environment.

Finally the management claims are taken into account and added to the primary test hours to estimate the actual time the test phase will take. The management claims can for example be size of the test team and tools support for problem reporting.

The whole procedure is illustrated in Figure 3.1.

![Schematic overview of Test Point Analysis](image)

**Figure 3.1 Schematic overview of Test Point Analysis**

### 3.2.1.2 Between release models

We think that most of the research in fault prediction is concentrated on the reliability of software, with the development of different reliability models. Most of the models are based on operational profiles [Musa93] and logged execution time (actual time used by a processor in executing a program's instructions) [Musa96] for the test cases. There also exist models of Bayesian Belief Networks, see BBN below.
**SRET**

Software-Reliability-Engineered Testing, SRET, combines the use of quantitative reliability objectives and operational profiles [Musa96]. The operational profiles are profiles made from user surveys of how the system is used or supposed to be used in reality. The profiles act as guides for the testers, so they know that the software test is performed according to its use or intended use. For example, some functions are used more often by the user and should be covered by more test cases.

According to Musa you can apply SRET to any software system and for most kinds of testing. The only requirement is that testing should be spread broadly across system functions, for example with help of operational profiles. You can apply SRET to feature, load, performance, regression, certification, or acceptance testing.

There are two types of SRET:
- Development testing, in which you find and remove faults. Here you estimate and track failure intensity, which are the failures per unit execution time.
- Certification testing, in which you either accept or reject the software. Here, there is no attempt to "resolve" the failures you identify.

In many cases, the two types of testing are applied sequentially.

**BBN**

Bayesian Belief Networks, BBN, is a graphical network and represents probabilistic relationships among variables [Fenton&Neil99]. The model is also known as Causal Probabilistic Networks, Probabilistic Cause-Effect Models, and Probabilistic Influence Diagrams.

BBN is a graph, see Figure 3.2, where the nodes represent uncertain variables and the arcs the causal relationships between the variables. The graph is combined with an associated set of probability tables, not shown here, which capture the conditional probabilities of a node.

The ability to use BBNs to predict defects, according to Fenton and Neil, will depend largely on the stability and maturity of the development process. They state that organisations, which do not collect metrics data, do not follow a defined life cycle or do not perform any forms of systematic testing, will find it hard to build or apply such models as BBNs.
3.2.2 Subjective models

The subjective models are built on personal opinion and intuition. There exist direct and indirect subjective models. The difference between them is that indirect models take various attributes into account.

3.2.2.1 Direct models

Direct models are one of the most basic ways to make a prediction. The prediction is made directly by a person, with knowledge about the system, who estimates the size or number of faults from their own view. No attributes except the person’s intuition are used. This can give very diverse results, since the estimate is based on experience.

3.2.2.2 Indirect models

In indirect models the prediction is based on human decisions, but with help of additional attributes, such as number of test cases, number of use cases or different size metrics. The attributes used should correlate to the attribute you want to predict or estimate. This is done by a statistical calculation.

When correlation exists, a mathematical model can be created. The model can be used in the following ways [Tze-jie et al. 88]:

1. description - the models identify those factors which are correlated with software defects
2. control - the models suggest ways to manipulate and track these factors
3. prediction - the models predict the number of defects that remain in the software product

A defect model generally takes the following form:

\[ D = f(M, ED, T, \text{others}) \]

where

- \( D \) = number of software defects found during a certain phase of the software lifecycle
- \( M \) = program metrics such as program size, number of decisions, number of variables, etc.
- \( ED \) = number of defects detected during an earlier phase of the software lifecycle
- \( T \) = testing time measured in CPU time, calendar time, or some other effort measure
- Others = other factors may include hardware facilities, types of software, development effort, programmers' experience, design methodologies, etc.

In chapters 2 and 3 we have presented the State of the Art within cost/effort estimation and fault prediction. In the following chapter we give a description of system test, to be able to propose a test effort model in chapter 5.
In this chapter we give a short general view of a system test phase and a more specific description of Symbian’s system test process.

4.1 General description
System test [Marciniak94] is often performed after the integration test is completed. In the system test the entire system is tested. Often is the requirement specification used to derive the test case selection. System testing looks for errors in the end functionality of the system and also for errors in non-functional quality attributes. These could for example be reliability, performance, stress tolerance, security and usability. System testing is often carried out by independent testers (not the ones who developed the system).

4.2 Symbian’s system test process
Symbian’s system test process, see Figure 4.1, is similar to the general description given above. Symbian uses functional specifications to describe the functionality of the system. In these functional specifications use cases are described, which are used to derive test cases.

The test cases are collected in test suites, which purpose is to focus on particular areas of the system during test. Each test case consists of one or more tests. A test is a collection of actions that has to be performed in order to test certain functionality. Each test has a test condition, which has an expected result that defines the result of these actions so it is possible to identify a simple pass or fail when executed.

Symbian brakes the system test down into a number of categories of tests. This helps with the prioritisation of testing schedule, which enables the focus of testing to be on the areas most used and important to the end user. Example of these categories is functionality, user interface, performance and security.
4.2.1 Symbian’s defect categories

The faults, found during system test, are categorised into different defect\(^1\) categories. Symbian divides the faults in four defect categories: critical, high, medium and low.

- A **critical category defect** is a defect that causes severe problems with the operating system or applications, which makes all or parts of the application/operating system unusable or untestable.
- A **high category defect** is a defect that causes major problems with the operating system or applications, which makes certain functions or features unusable or untestable.
- A **medium category defect** is a defect that causes noticeable problems with the operating system or applications, which makes certain functions or features less easy to use and would affect an end-users perception of the system.
- A **low category defect** is a defect that causes minor problems with the operating system or applications, which makes certain functions or features unusable or untestable.

In chapter 5 we try to connect effort/cost estimation models with fault prediction models, to propose a model for test effort estimation.

\(^1\) In this case we have used Symbian’s terminology where a defect is the same as the definition of a fault (See chapter 3). In the rest of the chapters we use defect and fault interchangeably.
5 Proposal for estimation improvement

As we have seen in chapters 2 and 3 a lot of different estimation models for cost/effort estimation and fault prediction exists. The models cover different development phases and some prerequisite is often needed. The prerequisite can be different documents or data collected, or that a specific task is done in a certain way.

5.1 Fault prediction with use of use cases

To create a fault prediction model many different sources can be used. It is important that the company, which will use the model, finds the one that is most suitable. Every company has something to improve and all models do not fit all companies. We have studied the development process of Symbian and we have tried to find the most suitable model from the State of the Art.

None of the studied models seems to fit Symbian perfectly. Therefore, our final conclusion is that the use of use cases to predict the number of faults is most suitable for Symbian. We base this on the fact that Symbian already creates uses cases and use this for the creation of test cases. Our proposed model is described in Figure 5.1 and will be described further in sections below.

![Diagram of fault prediction model using use cases](image-url)
5.1.1 How use cases can be used to predict the number of faults

Our basic assumption in this study is that use cases has a high correlation with test cases, since the test cases are developed from the use cases, see 1 and 2 in Figure 5.1. This correlation has to be proven before it can be widely used. We have tried to get access to this data, but have not succeeded.

The test cases can be used to predict the number of faults in the system, but also here is it important that the number of test cases correlates with the number of faults, see 2 and 3 in Figure 5.1. There might also be the situation that test cases are created by not using use cases. If this is the situation, these test cases also has to be considered, since additional test case also can reveal a number of faults. The later the test cases are created, the more the time-estimation will be delayed.

The logical assumption is that the number of faults will stabilise after a certain amount of test cases. If the assumptions of our model, illustrated by the dotted diagonal line in Figure 5.2, are correct it will be valid before this stabilisation occurs, see the vertical line in Figure 5.2.

![Correlation graph between faults and test cases](No of Faults vs No of Test Cases)

Figure 5.2 Correlation graph between faults and test cases

5.1.2 How long time does it take to correct a fault?

When the number of predicted faults in the product is known, it is time to estimate how long time it will take to correct a fault. This depends on which defect category the fault belongs to, see Figure 5.1. The time to correct each fault should be a mean value for each defect category. A prerequisite for this is that there exists a defect categorisation within the company.
Symbian divides the defects into four categories, critical, high, medium and low, as described in chapter 4.

The number of faults in each defect category has to be predicted before the time can be calculated. This is easiest done by direct estimation, but can also be based on statistics from earlier releases.

5.1.2.1 Statistics to use
If the company has statistical data from earlier releases, they can see how many of the faults that belong to each defect category and which test case it originate from. The time to correct each fault within a specific category has to be measured if this is not in the defect database. For each defect category a mean value has to be calculated.

5.1.2.2 Direct estimation
If direct estimation is used, which is the most used way, but also the most unreliable way, the estimate of the number of faults in each category is based on human intuition and experience. The mean time for correction can also be estimated in a direct way.

In addition to the time it takes to correct the faults, the time that is required to test through all test cases has to be added.

5.1.3 Reused code VS. new or changed code
We have included reused and new or changed code since it can be used for supporting the model. If you know how the fault categories and the type of code relate to each other, you can estimate in which category most of the faults will belong to. This relation can improve the prediction of which category the fault belongs to.

In addition, if you know which type of code the module belongs to it will provide a way to plan the test phase better, since new code often has more faults than reused code. The test of new code should start earlier so that these modules are heavily tested before release.

In addition to our model, we give a more verified model, see section 5.2. We propose the use of Capture – Re-capture, but first we give some examples how to use the proposed model.

5.1.4 Example of the proposed model
Here follow some examples of how the model can be used. We start by giving an example of how the model can be used in a planning phase. The second example shows how the model behaves with randomly drawn numbers in a small interval. The third example shows the model in the case of a larger interval. Example 2 and 3 can be considered as ‘real’ projects. The last example shows a worst case. This example can be a project, which is dissimilar to the other projects. It can for example include new technologies like a new programming language or some new tools.
Proposal for estimation improvement

All data in example 2 and 3 is randomly created for this example and is uniformly distributed in the interval. We will use 20 use cases as base in all examples and the times needed to correct the faults are constant. We assume that a critical fault takes 16 hours to correct, a high fault takes 10 hours, a medium takes 6 hours and a low fault takes 3 hours.

5.1.4.1 Example 1
In this example we describe the model with use of mean values. This is an example of how the model can be used in the planning phase to predict the time it will take to perform the system test phase.

Correlation between use and test cases
From the number of use cases we predict how many test cases will be created. We assume that all the use cases are used for the creation of test cases and that no additional test cases are created, see Figure 5.1.

\[
\text{# test cases} = \text{# use cases} \times Y \quad (4)
\]

where Y is a coefficient which describes the relationship between use and test cases. For the example we have chosen to give Y the interval 3 - 4 and a mean value of 3.5. The number of use cases in the example is 20. This gives us 70 test cases (4).

Correlation between test cases and number of faults
From the number of test cases we want to predict the number of faults that will occur in the system.

\[
\text{# of faults} = \text{# test cases} \times Z \quad (5)
\]

where Z is a coefficient which describes the relationship between test cases and the number of faults. Z is in the interval 1 – 2 with a mean value of 1,5. This gives us 105 faults (5).

Defect categories
We have randomly selected the percentage of appearance of faults in four defect categories, based on Symbian’s categorisation, see section 4.2.1. The result is:

Critical: 16%
High: 17%
Medium: 36%
Low: 31%

This gives us 17 critical faults, 18 high faults, 38 medium faults and 32 low faults. These values are used for the time estimation, the time needed for correcting the faults.
**Time estimation**

The total time for testing and correcting the faults can be estimated based on defect categories and statistic from earlier releases. The time it takes to correct each fault is given in section 5.1.4 above. The total time to correct the faults is 776 hours (6).

\[
\text{# critical faults} \times \text{time to correct} + \text{# high faults} \times \text{time to correct} + \text{# medium faults} \times \text{time to correct} + \text{# low faults} \times \text{time to correct} \\
= 17 \times 16 + 18 \times 10 + 38 \times 6 + 32 \times 3 = 776
\]  

(6)

In addition to this we have to add the time it takes to run through the test cases. We have assumed that this takes 1.5 hour per test case, in an interval of 1 - 2. This gives us 105 hours (7).

\[
\text{# test cases} \times \text{time to test} = 70 \times 1.5 = 105
\]  

(7)

The total time for testing and correcting the faults will be 881 hours or about 22 weeks for one person.

5.1.4.2 Example 2

In this example we describe the model with values in a small interval. This could be compared with a project that resembles the mean project, example 1 above, but has some small deviations.

**Correlation between use and test cases**

The coefficient Y is within the interval 3 – 4 and randomly selected to 3.2. This gives us 64 test cases (4).

**Correlation between test cases and number of faults**

The coefficient Z is within the interval 1 – 2 and randomly selected to 1.6. This gives us 102 faults (5).

**Defect categories**

We have randomly selected the percentage of appearance of faults in the four defect categories. The result is:

- Critical: 20%
- High: 7%
- Medium: 32%
- Low: 41%

This gives us 20 critical faults, 7 high faults, 33 medium faults and 42 low faults. These values are used for the time estimation, the time needed for correcting the faults.

**Time estimation**

The time it takes to correct each fault is given in section 5.1.4 above. This would give us 714 hours to fix the faults (6).
In addition to this we have to add the time it takes to run through the test cases. We have drawn that this takes 1.6 hour per test case, in an interval of 1 - 2. This gives us 102 hours (7).

The total time for testing and correcting the faults will be 816 hours or about 21 weeks for one person. If this is compared with the planned project in example 1, we can see that the estimation only differs in 1 week. This depends on the many small estimation steps within the proposed model. In the next example we will see how the model tolerates a larger interval.

5.1.4.3  Example 3
In this example we describe the model in the case of a larger interval. This example can also be compared with a project that resembles the mean project, but it has larger deviations.

**Correlation between use and test cases**
The coefficient Y is within the interval 1 – 6 and randomly selected to 4.2. This gives us 84 test cases (4).

**Correlation between test cases and number of faults**
The coefficient Z is within the interval 0.5 – 2.5 and randomly selected to 1.9. This gives us 160 faults (5).

**Defect categories**
We have randomly selected the percentage of appearance of faults in the four defect categories. The result is:

- Critical: 28%
- High: 22%
- Medium: 18%
- Low: 32%

This gives us 45 critical faults, 35 high faults, 29 medium faults and 51 low faults. These values are used for the time estimation, the time needed for correcting the faults.

**Time estimation**
The time it takes to correct each fault is given in section 5.1.4 above. This would give us 1397 hours to fix the faults (6).

In addition to this we have to add the time it takes to run through the test cases. We have drawn that this takes 2.1 hour per test case, in an interval of 0.5 – 2.5. This gives us 176 hours (7).

The total time for testing and correcting the faults will be 1573 hours or about 40 weeks for one person.
If this is compared with the planned project in example 1, we can see that the estimation differs in 18 weeks. This depends on that all randomly chosen values are on the higher side of the interval, above mean value. If any of the random values were below mean value, the final result would have been closer to the estimation.

**5.1.4.4 Example 4**

In this example we describe the model with values in the high end of the large interval. This could be compared with a project that largely differs from the mean project, example 1 above. This is because we want to show how the proposed model handles a worst case.

*Correlation between use and test cases*

The coefficient Y is within the interval 1 – 6 and is selected to 6. This gives us 120 test cases (4).

*Correlation between test cases and number of faults*

The coefficient Z is within the interval 0.5 – 2.5 and randomly selected to 2.5. This gives us 300 faults (5).

*Defect categories*

We have randomly selected the percentage of appearance of faults in the four defect categories. The result is:

- Critical: 4%
- High: 20%
- Medium: 51%
- Low: 25%

This gives us 12 critical faults, 60 high faults, 153 medium faults and 75 low faults. These values are used for the time estimation, the time needed for correcting the faults.

*Time estimation*

The time it takes to correct each fault is given in section 5.1.4 above. This would give us 1935 hours to fix the faults (6).

In addition to this we have to add the time it takes to run through the test cases. We have drawn that this takes 2.5 hour per test case, in an interval of 0.5 – 2.5. This gives us 300 hours (7).

The total time for testing and correcting the faults will be 2235 hours or about 56 weeks for one person.

If this is compared with the planned project in example 1, we can see that the estimation differs in 35 weeks. As can be seen in this last example the model does not cope with
extreme cases. This can mean that projects that differ from each other, for example integrates new technology, should be carefully compared with the mean project.

The many small steps in the model produces a uniform distribution between high and low values. This makes the model more tolerant to fluctuations. But, as we can see in example 3 and 4 above, if all values in the model tend to be higher or lower than the mean value, the model will produce under- or over-estimated predictions.

### 5.2 Capture – Re-capture

Capture – Re-capture is a model for prediction of faults. It is based on Inspections, as mentioned in section 3.2.1.1. We have included it in this section since we think that it is an easy way to start controlling the development process. It will give the company statistical data to base future estimations on and also rise the quality of the developed product. Inspections are also a rather easy process to learn.

Capture – Re-capture should be applied through the whole development process, it is applicable to requirement specification, design and code. The prerequisite for using Capture – Re-capture are few, but some kind of Inspections has to be done, and there has to be something to inspect. Each inspector must log the faults found in these Inspections. It is recommended that at least 4 inspectors are present at each meeting [Briand et al. 00].

The basic foundations of the model are described in section 3.2.1.1.

The model predicts the faults that remain within a document or code after the Inspection is done (the right pointing arrows in Figure 5.3). These faults will escape the phase if nothing is done. The last right pointing arrow describes how many faults that will remain in the product when it is released.

Vertical arrows going into the box represent faults/size measure introduced at this step. Vertical arrows out represent fault/size removed at this step. Arrows pointing to the right represent fault/size being transmitted from one step to the next. For example the DESIGN, where we have 21 faults/KLOC generated, 4 faults/KLOC coming from REQS. 12 of these 25 faults/KLOC are detected and removed, while 13 faults/KLOC are passed on to CODE.
To be able to use this model as a test effort prediction model, the time needed for correcting each fault has to be estimated by statistic or direct estimation.

Capture – Re-capture can also be complemented by curve fitting models as mentioned in section 3.2.1.1.

In chapter 6 follows the conclusion of this report.
6 Conclusion

The conclusion from this master thesis, which has been performed at Symbian, is that it is difficult to find, or create, an estimation model that will cover all problems encountered in an organisation.

We have in this thesis looked at the State of the Art within cost/effort estimation and fault prediction. This has given us a broad view of how different models try to solve estimation and prediction problems.

The models have been developed during the last 40 years and as mentioned in this thesis there still do not exist standards of how to measure cost and effort, and predict faults. This must mean that it is very hard to make a general proposal of how to apply such techniques. Because of this, every organisation needs to adapt or create a model for their own needs, as we have tried to do in this thesis for Symbian.

Since many of the models require a specific way of working they have been hard to adapt to Symbian, because they use their own procedures. We have therefore proposed a model, which is mainly based on procedures used at Symbian. In this case we have chosen to create a test effort estimation model that has its origin in use and test cases. We have not been able to verify this model, since there has been a lack of data. This can be a foundation for future studies.

Our second proposed model is Capture – Re-capture since it is an easy way to introduce more control of the developed product. Capture – Re-capture requires more documentation from the organisation and is therefore not our primary solution for Symbian.

We have tried to make the proposed test effort estimation model as general as possible and we think it is applicable to projects that create test cases from use cases.

In chapter 7 we give some specific recommendations to Symbian of how to apply this model.
As we have seen in this report there exist many different models for cost/effort and fault estimations. Many of them require a lot of measurement data and new procedures. We here give our recommendations for Symbian of how to best apply such a model.

We have found that there exist some things to be improved in the development process while we been at Symbian. The way to handle requirements, design and metrics does not follow the State of the Art within software engineering. This does not necessary mean that it is the wrong way, but it makes it difficult to find verified solutions to apply.

We therefor recommend a solution, the use of use cases to predict test effort, which does not have too strict requirements on documentation and the development process. Our proposal is to use what already exist in the company as far as possible. In Symbian’s case this means to use and further develop their use cases and the use of them to create test cases. It also includes a few improvements in the metrics area, to be able to control the test time.

We also want Symbian to see this master thesis as a starting point to think about what is needed to introduce a higher control of the development process. There might be bits and pieces in other models, which you find useful. Just remember to collect metrics, which you find useful and needed and do not start to collect too many before you know what to do with them.
8 References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Year</th>
<th>Title and Details</th>
</tr>
</thead>
</table>