Diploma Thesis

March 23, 2006

Change Prediction Cost Model

Developing a CPCM based on Version History data and Change Couplings

Béla Grossmann
of St.Gallen, SG, Switzerland (00-703-421)

supervised by
Prof. Dr. Harald Gall
Dr. Martin Pinzger

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Diploma Thesis

Change Prediction Cost Model

Developing a CPCM based on Version History data and Change Couplings

Béla Grossmann

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Diploma Thesis

Author: Béla Grossmann, bela.grossmann@gmail.com

Project period: 23rd September 2005 - 23rd March 2006

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank Professor Harald Gall for giving me the opportunity of writing this thesis and for his advice.

Special thanks goes to Martin Pinzger for his great support, guidance, advice, patience and encouragement during the last six months.

Thanks also goes to Beat Fluri for his help in LateX.

I would like to thank Guerino Mazzola and Christoph Luchsinger for their valuable inputs concerning the development of the change prediction cost model.

Many thanks to Adrian Röllin for his introduction to R.

Further thanks goes to all the students I met during my studies and while writing the diploma thesis for the good humour.

Abstract

Software maintenance and evolution is an expensive part in the life cycle of a software system. It is of great interest to apply cost estimation models to predict future maintenance costs, especially for the management.

In this thesis we examine if release history database metrics like change couplings and modification reports can support the prediction of software maintenance and evolution costs. In three approaches we show the development of a change prediction cost model. This thesis presents an insight into the data retrieval and provides a detailed description of the significance analysis of the input values.

Zusammenfassung

Softwarewartung und -evolution ist ein teurer Abschnitt in der Lebensdauer eines Softwaresystems. Es ist deshalb von grossem Interesse, Modelle, die die zukünftigen Wartungskosten schätzen, anzuwenden, speziell für das Management.

In dieser Arbeit wird untersucht, ob Informationen aus Release-History Datenbanken wie Change Couplings und Modification Reports die Vorhersage von Softwarewartungs- und Softwareevolutionskosten unterstützen. Die Entwicklung des Change Prediction Cost Models ist in drei Ansätze aufgeteilt. In dieser Arbeit werden die Datenbereitstellung und das Vorgehen der Signifikanzanalyse detailliert beschrieben.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an introduction to the subject of this diploma thesis. At first, we specify the problem, followed by an introduction of software maintenance. At the end of this chapter we introduce the structure of the thesis.

1.1 Problem Statement

Software cost estimation is difficult in every stage of a software life cycle. We concentrate on the cost estimation for software evolution. A software evolves over time due to maintenance activities. Each cost estimation model is based on input values. They can be divided in source code metrics data, release history data and expert knowledge. In this thesis we develop a change prediction cost model based on version history data and change couplings. We do not develop a cost model from scratch, but use an existing cost model as basis and extend it to our purpose.

1.2 Software Maintenance

In successful software projects, the maintenance costs mount up to 70 % of the total project costs spent over a software life cycle. Despite of this, only the development costs are brought to the foreground. The estimation of upcoming maintenance costs is not trivial and depends on a large number of influencing variables. To be aware of the total costs of a software system, the whole life cycle has to be considered.

1.2.1 The need of Software Maintenance

Normally, the owners of new software products think, that their product does not need any care for ages. But as mentioned before, more than 70 % of the costs of a software are caused after its release by the need of maintenance. But why? - Software ages!

David L. Parnas mentions in his paper [Par94] two main causes for software aging. The first one is the result of the lack of movement. This means, that every software has to be modified to meet changing needs. The second is the result of ignorant surgery, the bad consequence of the changes that are made.

Lack of Movement

The expectations about software products changed during the last years. In the end of the 1970s users have been happy accessing the computer and using its programs over a command prompt. Storage was very expensive and the capacity of the machines was limited. Today, mass storage and high processor speed is affordable even for home users. Due to the fact, that users normally prefer to work with an userfriendly interface, software without a graphical user interface gets replaced even if the product still works properly. As explained above, users expect more about software than they did 30 years ago. Unless software is not updated frequently, its users become dissatisfied and the software will be replaced by a new state-of-the art software product.

Ignorant Surgery

It is essential to update software to prevent aging. But these updates may cause aging as well. Software is usually designed with a certain concept in mind. If updates are made, this concept must be understood, otherwise the changes will be inconsistent with the original concept. But normally the updates do not refer to the original concept. After the first changes, the software system can only been understood, when both of the concepts are known. After some further changes like this, the original concept dilutes more and more and neither, the original designer nor the ones that made the updates will understand the system. Software that is maintained like this becomes very expensive to update. The longer software is updated in this way the longer it takes to find and to solve the errors. The possibility to introduce new bugs during writing updates increases. Documentations are neglected and not kept up to date. This makes it even worse and more complicate to understand the program code.

1.2.2 Weight gaining Software

Software maintenance does not have top priority among programmers. Very often those duties have to be done by trainees or junior programmers. Of course they are not that experienced like a senior programmer. The consequences of which are, that problems are "quick-fixed", the code that was changed during the maintenance task causes even more bugs and in the end the software is not maintainable anymore.

The easiest way to add a new feature is to add new code. The more such changes are made, the more the original design is diluted and changes are getting more and more difficult to implement. First, there is more code to change. Second, a change that in the original source code affected only one part of the code, concerns now many sections of the code. This interconnectednesses are called *change couplings*. The time it takes to maintain the software system increases and causes delays for the customer. It ends in an *inability to keep the software up*.

As the software grows, it becomes difficult to maintain and uses more computer memory and storage space. The program becomes slowly and customers must upgrade their computer infrastructure to keep the acceptable response time up. Maintenance causes a *reduction of the software performance*.

Even when software gets maintained, bugs are introduced. More bugs denotes *decreasing reliability* for the whole software system. There exist documented studies, that every error correction introduced more than one new error. In other words, every attempt to make software more reliable made it even worse [Par94].

1.2.3 Preemtive Medicine

Software aging is a serious problem. There are some methods to prevent software systems from aging. One of the first slogans mentioned about this topic was *design for change* [Par94]. A lot of methods might be applied in connection with the design for change, e.g. "information hiding", "abstraction", "separation of concerns", "data hiding" or "object orientation". To apply this principle we have to characterise the changes that might upcome over the life cycle of a software system. Since we are not able to predict exactly which changes will have to be done, it is very important to think about the probability of each type of change. Then, the software has to be arranged so that the items that have to be changed with the utmost probability concern only a small part of the code. Despite of the existence of methods to minimise such couplings, they are rarely applied during the design of a software system.
There are two main reasons for it:

1. It requires a lot of knowledge about the application and its environment and about further needs of change.

2. Programmers are not eager in thinking about principles like design for change. Software projects normally underly time pressure. Deadlines have to be reached and future maintenance does not have top priority.

A software project that was built after the principle of information-hiding is unusual. Normally the code of a software has to be efficient and correct. It is rarely designed to be easily changed. Most of the programmers do not use this simple principle because they think that their product will be so good, that it never has to be changed. Only software that can be understood, due to the inclusion and the observance of some guidelines, can be changed and kept alive. Designing for change needs some additional thoughts, but it is designing for success.

Not only the design is important to keep the overview over a software product. The documentation is as important as the design but suffers from neglect as well. It is common that programmers argue that the code is its own documentation. Even if documentation is written it is poor, incomplete and not up to date. Maintainers do not revere it, because they can not rely on it. Documentation is neglected because it decelerates the progress of software development. This is the way one might think in short-term. Long-term costs are going to explode if documentation is neglected.

Even if all the possible preemtive measures are taken, aging is inevitable. The success of the design for change depends on the ability to predict upcoming changes that will have to be made. Over several years, changes will have to be made that will not fit the original prospects. The documentation will never be up to date, even if held formal and precise. Preventive measures are worth it, but they can not stop software aging [Par94].

1.3 Structure of the Thesis

In Chapter 2 we present related work done in the area of cost models. Chapter 3 introduces the function of concurrent versions systems and bug tracking systems and their function as feeder for the release history database. The development of the change prediction cost model is presented in Chapter 4 and its implementation and validation in Chapter 5 and Chapter 6. We terminate the thesis with Chapter 7 where we present its contribution, the lessons learned, and provide inputs for future work.

Chapter 2

State of the Cost Models

There is not that much literature about cost models in the software engineering and maintenance area that can be used to create a CPCM. At first we based our "search for the needle in the haystack" on the paper titled "Software Maintenance Cost Estimation And Modernization Support" [KLT03]. General concepts of software cost estimation models by Lehman, Boehm and Kemerer are introduced first. Then studies about software maintenance and cost estimation are presented. Although the paper gives a wide and deep overview over the state of the research and provides points to remember, we did not find a suitable model for our purpose and had to expand our investigation. In this chapter we describe four cost models in details.

2.1 Barry Boehm

2.1.1 COCOMO

The COCOMO[1] is one of the most popular cost estimation model. It was originally published in the book titled *Software Engineering Economics* by Barry Boehm in 1981 [Boe81]. He developed an easy-to-understand model that predicts the effort and duration of a software project. He based his model on an analysis of 63 software-development projects. As inputs he used values relating to the size of the resulting systems and a number of "cost drivers" that he believed affect productivity.
The effort equation for the Basic Model of the COCOMO is of the form:

$$MM = C * (KDSI)^k,$$

where:
- MM := number of man-months (where 1 man-month equals 152 working hours),
- C := a constant,
- KDSI := thousands of "delivered source instructions" (DSI), and
- k := a constant.

DSI is defined as program instructions including job control language, format statements, and data declarations and excluding comments and unmodified utility software. Those instructions are created by project members and are delivered as part of the final project.

[1] COnstructive COst MOdel

After validating the Basic Model with the 63 software projects, Boehm realised that this Model was not precise enough. To improve the models accuracy he refined the equation and included 15 cost drivers. These cost drivers include the project environment, the personnel staffing, the computer used and provide some additional inputs over the end product. Boehm calls this version of the model the Intermediate Model.

The Detailed Model is very similar to the Intermediate Model except that it divides the project into four phases: Product Design, Detailed Design, Coding/Unit Test and Integration Test. The 15 cost drivers are estimated and applied to each of the phases separately, which provides a more accurate cost estimation [Kem87].

2.1.2 COCOMO II

As a matter of fact, the way how software is developed has changed a lot since the late 1970s. This made it important for the COCOMO to be updated and reinvented for the 1990s and the beginning of the new millennium. After several years of research the COCOMO II [Bai98] was published in 1995. It represents a reinvented cost estimation model reflecting the changes in professional software development that have come along since the first publication of COCOMO. The old model gets called COCOMO 81 from now on.

> "COCOMO II focuses on issues such as non-sequential and rapid-development process models; reuse-driven approaches involving commercial-off-the-shelf (COTS) packages, reengineering, application composition and application generation capabilities; object-oriented approaches supported by distributed middleware; software process maturity effects and process-driven quality estimation." [Bai98]

In the late 1970s there was a single software life cycle model. Today and in the future, software projects will adjust their development process to its cost drivers. In the beginning of a software development process there is only very little information available about its size, personnel needed or the detailed technology to be used. As the process continues further up the life cycle a lot of information about the projects design parameters and its cost drivers[2] are better known. According to this increasing level of information-granularity the COCOMO II provides three different models correspondent to its input values:

1. The first cost estimations have to be proceeded with the *COCOMO II Application Composition* model. It is used in the first phases of the project or spiral cycles where prototyping is used. If there is any other prototyping activity later on in the life cycle, the application composition model will be used for the estimation again.

2. In the next phases or spiral cycles there is generally not yet enough information for a fine grained cost estimation. The software is developed incremental and alternative ways of the architecture have to be examined for usability. At this point of the project the input values available out of the process and the estimation provided by the *Early Design* model are on a general level.

3. The *Post Architecture* model is used when the project is ready to be developed and the life cycle architecture is defined. This state of the project provides a lot of accurate information on cost driver inputs and enables a fine grained cost estimation.

[2] "A cost driver refers to a particular characteristic of the software development that has the effect of increasing or decreasing the amount of development effort, e.g. required product reliability, execution time constraints, project team application experience." [Bai98]

2.2 Magne Jørgensen

Magne Jørgensen wrote in 1995 a paper about the experience and use of eleven different software maintenance effort prediction models [rge95]. They where based on the theoretical background of regression analysis, neural network pattern recognition, and on a collection of randomly selected maintenance tasks out of more than 70 applications with a size of up to 500,000 lines of code.
The design of the effort models was split up into four important decisions:

1. *Definition of the size of a maintenance task*
   The estimation of the size of a maintenance task is the most important input to an estimation model. Normally it is solved either with lines of code (LOC) or function points. LOC are easy to measure and handle but it is important to use always the same counting standards and definitions. It has to be defined whether comments and blank lines have to be included or not. But LOC provide the user with no information about the characteristics and complexity of the code.

2. *Variables to include in the prediction models*
   They decided to include only variables that significantly $(p < 0.5)$ correlate to maintenance productivity. Maintenance productivity is defined as following:

   $$\text{Maintenance Productivity} := \frac{(LOC_{inserted} + LOC_{updated} + LOC_{deleted})}{\text{effort used on the task}}$$

   It is also very important to select those variables where it is possible to estimate accurate values.

3. *The prediction approach to use*
   When analysing the functional relations between variables the regression is the approach most often chosen. For a nonlinear regression approach feed forward neural networks with back propagation are used. In opposition to typical regression models, neural networks use a large number of input values to allow complex functional relations between the dependent and the independent variables. The last approach relies on historical project data and applying pattern recognition.

4. *The prediction model to implement and compare*
   The last step was to develop the prediction models. Eleven models where built and compared.

Prediction models are not built to replace experts opinions and estimations. They should support their knowledge about the skills of the maintainers, the environment and methods used which can not be represented in variables to include into a formal model. The best results were achieved by models applying logarithmic linear multiple regression and a hybrid model type based on pattern recognition and simple regression.

2.3 Chris F. Kemerer and Sandra Slaughter

Chris F. Kemerer and Sandra Slaughter wrote a paper in 1999 about "An Empirical Approach to Studying Software Evolution" [KS99]. As it is written already in the title, the paper shows a number of attempts to measure the dynamic behaviour of software systems that are maintained over its lifetimes.

2.3.1 Software Evolution

After the rollout of a software system, the software engineering process is not yet finished. The tasks that are focussed on are upgrading, migrating, and evolving. The year 2000 problem actually disclosed the fact that there are still a lot of old systems up and running. By this, we are not talking about small applications running somewhere in the background but rather about huge, important enterprise systems developed years ago. Unfortunately, while it is known, that maintenance is an important and expensive part in the software life cycle, still little systematic and scientific researches have been published about it.

> "...only about 2 percent of empirical studies in software engineering focused on maintenance, despite reports that at least 50 percent oft software effort is devoted to this activity." [KS99] p.493

In the beginning of research projects like this, a lot of problems have to be solved to get valuable and accurate results in the end. It is difficult to collect useful and significant data about the software development process. In addition there do not exist that much theories or models that could be used. But those are normal researcher problems. Extra difficulties underlie the distinction between the parts of the system that were changed during maintenance or were just part of planned increments.While talking about software maintenance you will get in touch with software evolution. Normally those two concepts are not fully understood and get mixed up.
Software Maintenance is defined in the IEEE Standard 1219-1998 as:

> "Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment." [Soc98]

A similar definition is given by the ISO/IEC 1995:

> "The software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity." [Org95]

The term Software Evolution, lacks a standard definition. Corresponding to Kemerer and Slaughter [KS99] and the paper "Software Maintenance and Evolution: a Roadmap" of Bennett and Rajlich [BR00], evolution can be described as:

> Software Evolution points out the dynamic behaviour of programming systems over their lifetime. It covers development, maintenance, migration, phase-out and close-down.

Every activity that takes place after the rollout is called maintenance. By comparison, software evolution highlights the dynamic behaviour of a software system over time after the initial implementation. To gain insight in a software systems evolution, data of at least two different point in time must be considered.

2.3.2 Data Analysis

23 different COBOL business systems where examined, providing more than 25,000 change logs. Whenever a software update was made, a log file was written by the maintainer. Out of this logs, the change history could be extracted. It provides information about the creation data of the original software module and its author, the function of the module, the programmer making the change, the date of the change and the description of the change. The logs were used to classify the change events into three basic classes: corrections, adaption, and enhancement according

to Swanson [Swa76]. These basic types were refined into 30 subcategories. Each log file was analysed and compared to a decision workflow to allocate it to its corresponding subclass for better understanding and describing the dynamics of software evolution.

The proceeding described above was applied to a Financial Sales System and a Manifest Shipping System. Over 81 % of the changes to the Financial Sales Reporting System and over 83 % of the changes to the Manifest Shipping System were either enhancement or new programs. This is consistent to prior research, and supports the breakdown into 30 subcategories. In addition to the change event data, detailed attributes about the two systems are presented in Table 2.1.

	Financial System	**Manifest System**
Age (years)	20	10
Size in Total LOC	181,652	189,999
Number of Modules	109	45
Average Module Size (LOC per Module)	1,666	3,878
Cyclomatic Complexity per LOC	0.04624	0.04296

Table 2.1: Profile of two Systems

The Financial System is twice as old and has more than twice as many modules as the Manifest System. Even though the Manifest System has half as many modules it is larger than the Financial System. As a result, the average module size for the Financial System is less than half that for the Manifest System.

To learn how systems evolve over time based upon a large set of detailed change events, time series and sequence analysis like phase mapping, gamma analysis, and gamma mapping were used to analyse the data. Time series models were not able to provide an insight into the software evolution process. The sequence analysis turned out to be more appropriate:

1. *Phase Mapping.* The phase mapping analysis produces a detailed phase map that describes phases that can be of different lengths, identifies phase repetition, and identifies disorganized periods. Phases are defined as four consecutive occurrences of an event.
2. *Gamma Analysis.* The phase mapping provides a starting point for understanding and comparing the evolution of the systems. The gamma analysis provides a precise, statistical analysis of the phases. It calculates a measure of the general order of the phases in a sequence.
3. *Gamma Mapping.* As the last step in gamma analysis we introduce the the gamma mapping. In this map, boxes of different size are drawn to indicate the separation of the phases. This maps permit an overview about a systems phases like introduction of new modules, changes of logic and user interfaces and provides a basis for systems comparison.

The paper points out accumulated needs in the area of software maintenance research. Although maintenance is a very important part in the life cycle of a software system, the accurate knowledge about its correlation of cause and impact is missing. While analysing the maintenance and its effects and the evolution of the whole system, one gets a deep insight in the structure and the dependence that lies between several parts of the system. An understanding of how the system evolved in the past over time can provide information about its behaviour for the future when further maintenance and enhancement has to be done.

2.4 Lucia et al.

Andrea de Lucia et al. introduced in the paper "Early Effort Estimation of Massive Maintenance Processes" [LPSV02] a cost model based on an empirical study experienced of a major interna-

tional software enterprise, namely EDS Italia Software (EDS SC). The goal was to analyze the adaptive maintenance processes to formulate rules for future projects.

2.4.1 Introduction

The deployment of the model and the needed analysis of the software based on a subset of the software system. Normally the systems are installed on customers mainframes, not accessible from outside. Furthermore, due to copyright and security reasons, the maintainers are not allowed to take a copy of the whole system but only of parts of it. As a consequence, every maintenance process has to be done on a subset of the system. After finishing the task, the updated code gets merged with the rest of the operating system. Therefore, it would be useful to have an effort estimation model that could predict the effort for the whole systems maintenance based on parameters gained out of the subset. Additionally the whole estimation should be based on product metrics that can be obtained out of the subset.

2.4.2 The Background of the Model

EDS SC was involved to the maintenance of software systems threatened by the Y2K problem or systems that had to be switched to the new currency used in the European Union, the Euro. Every project started with a segmentation of the source code into loosely coupled but cohesive work-packets (WPs).
This segmentation has four advantages:

1. Large projects are affected to high risks. The split-up of large projects allocates the risks among the pieces and restricts failures to single WPs. Furthermore, the management of a small project is less time-consuming.

2. Different teams can work parallel on their WPs, which reduces the time to delivery.

3. Relying on the effort it took to maintain a WP and the size of the team that was necessary, the total effort and the size of the staff can be estimated for the whole system.

4. The time it takes to set up the management for a WP is shorter compared to the whole project. While the split-up of the source code into WPs is continuing, the first team is already able to start its work on the assigned WP.

The project used and analysed for the development of the model was a project consisting of about 40,000 software components. About 15,000 components had to be modified, including 7,082 programs and 6,850 JCL [3] procedures.

The analysis of the project proceeded in three steps: First, information about the applications, software platforms and programs and their locations where collected. The second step consisted of a static analysis of the objects of each application which were executed trough automatic tools. In the last step, the acquired information were analysed. The analysis mainly focussed on the identification and interpretation of values containing data about the maintainability as shown in Table 2.2.

[3] JCL statements provide information that the operating system needs to execute a job.
http://www.okstate.edu/cis_info/cis_manual/jcl_over.html (last visited 10-02-2006)

Metric	Description
SC	Number of software code components of the WP
LOC	Lines Of Code, including comments, blanks, and declarative lines
CYC	McCabe cyclomatic complexity
CVAR	Number of control variables
VOL	Halstead Software Science Volume
UPL	Number of logical branch not used
Effort	actual effort of the WP measured as man-days

Table 2.2: Maintainability Metrics

2.4.3 The Model at a glance

The basic idea behind the model was to predict the effort of the whole system on the basis of the characteristics of a WP which represents a certain percentage of the system. Based on a correlation analysis over the maintainability metrics shown in Table 2.2 and the constraint that the model should consider at least one one-dimensional metric like LOC, a structural metric like CYC and the number of programs each WP consists of, the metrics were determined.
The following multivariate model was created:

$$Eff = (\alpha_1 * SC + \alpha_2) * log(\overline{LOC}) + (\alpha_3 * SC + \alpha_4) * \overline{CYC} + \alpha_5 * SC \tag{2.1}$$

where:
- Eff := the effort estimated,
- SC := number of software code components of the WP,
- $\overline{LOC}$:= the mean number of LOC of the components of the WP,
- $\overline{CYC}$:= the average McCabe cyclomatic complexity.

To build the model, the mean values of metrics composing each WP were computed. Unfortunately there is no exact description in the paper about the mode of calculation of the variables $\alpha_1 \ldots \alpha_5$ shown in Table 2.3.

α_1	α_2	α_3	α_4	α_5
4.9271	49.8522	0.0741	4.1061	39.9794

Table 2.3: Model Variables

The main advantage of this model is its simplicity and understandability. It allows a cost estimation in an early stage of the software maintenance process due to the available metrics through the aids of code analysis tools and data repositories.

2.5 Summary

While analysing the cost models, we paid attention on the input values. We just did not want to establish another model, relying on expert knowledge. Every single input variable should be gained out of the release history database. Our idea was to use its data to learn from the past and estimate future development costs.

The COCOMO is a powerful tool to estimate costs in different stages of a software life circle. The input values have to be provided in different granularities according to the software life circle.

A weakness of the model constitutes the comprehension and dependency on expert knowledge in the preparation of input values.

Magne Jørgensen presented in his paper a number of interesting effort prediction models. Due to their complexity we did not find an appropriate model for our purpose. Nevertheless, the paper provides a sophisticated insight in the development of effort estimation models.

An interesting method to analyse and visualise the evolution of a software system is introduced by Kemerer and Slaughter. As an interesting extension of this approach we suggest the connection to a cost estimation model relying on the extracted knowledge about the system.

Finally, we decided to base the development of our change prediction cost model (CPCM) on the model introduced by Lucia et al. The idea to provide a simple to use cost model based on easily available metrics is similar to our intent. The plain structure of the model allows an extension without touching its initial design. In Chapter 4 we present the extension of this model to a change prediction cost model.

Chapter 3

Tools of the Trade

While developing a software system, a lot of documents and data are stored in several databases. After the release of the software, this storage of data continues. Version control and bug tracking systems contain data about a softwares historical evolution and provide an insight into its structure and growth. The combination of both of these two databases provides the possibility of deeper analysis of software evolution aspects. The paper "Populating a Release History Database from Version Control and Bug Tracking Systems" [FPG03] introduces this approach on the open source project Mozilla.

3.1 Version Control System

Release history data is obtained from repositories of versioning systems such as the Concurrent Version Systems [ea05] or Subversion [CSFP02].

Concurrent Versions System

The Concurrent Versions System (CVS) is a version control system, recording the history of source files. When developing software in a team, you have to be careful not to overwrite each others' changes. CVS helps to keep the different changes. Every developer works in his own workspace and CVS merges the source code to one piece after the updates of the developers are made and checked in. CVS stores only the differences between two subsequent file versions to save storage space. Another advantage provided by CVS is the easy backtracking of versions when a bug has been found. The old versions can be retrieved and the changes that caused the bug can be found.

But CVS is not the medicine for every problem. The version system is not a substitute for management or communication among developers and all other members of the team. The management is still in charge of leading the project. They have to set milestones and control the progress of the design and development of the project. CVS will help to manage the versions of the source files, but it will not do it on its own. Furthermore, CVS does not have change control. That means, that there is no bugtracking. A database that keeps the reported bugs and their corresponding status, must be set up externally.

Subversion

CVS has been the tool used for many years among open source communities. The usage of the CVS made it possible that a lot of members worked simultaneous on the same project distributed over several countries. But the CVS is no longer state-of-the art. Subversion belongs to the new

generation of version control systems succeeding CVS. The designers of Subversion tried to gain the further CVS users on two ways: First, they developed an open source system with a design similar to CVS. Second, they tried to remove the bothering bugs known from CVS. The main target was not to reinvent the concepts of tracking the versions of a system, but to provide a powerful, easy-to-use, and flexible tool.

With Subversion it is possible to track files and directories instead of tracking files only. Operations such as copying or renaming files which are actually workings concerning the content of the directory are supported in Subversion. The tracking of the hierarchy in which files are organized makes it possible to add, delete, copy or rename both, files and directory. In addition, the metadata of files and directories are stored as well.

3.2 Bugzilla Bug Tracking System

Bugzilla is a free to use bug or defect tracking system. It allows to keep track of outstanding bugs found in software projects. It supports the user with information about the status, resolution, severity, priority of the defect, to mention only the key-attributes [Org98].

3.3 Release History Database

A Release History Database (RHDB) introduced in the paper of Fischer et all. [FPG03] combines version and bug tracking data to provide a deep insight into the evolution of a software project. It is build out of the versioning data from CVS and bug report data from Bugzilla. CVS stores modification reports but does not provide enough tools to get an insight into the evolution of a software system itself. Unfortunately, the modification reports are stored in informal text files. To connection the corresponding bug reports, the information in the reports have to be parsed and linked with data from the bug tracking system. The combination of data concerning information about the versioning and data concerning the bug reports comprehends useful advantages. First, logically coupled files can be detected. These are files coupled to the appearance of the same bug report ID. Second, error prone classes and their affected components can be detected.

The data is stored in a structured way and enables developers to browse and analyse the historical data. In normal case, those data are not considered due to its large amount and difficult to access and understand. Stored in a structured way, these information provides engineers with a detailed overview over the history of the software system and builds the basis to learn from the past.

3.4 Source Code Model Databases

Each of the seven Mozilla releases that we used in this thesis is stored in a database of its own. When we retrieve data concerning the Mozilla source code we query these seven databases sequential. The database schema is identical for all of the releases. Furthermore, it provides a number of precalculated values like the LOC of a module or the cyclomatic complexity of a module. The table `mapping_rhdb` maps the file IDs that were retrieved from the source code model database to its corresponding file ID used in the RHDB.

3.5 Summary

We validate the development of the CPCM on data of the open source project Mozilla. For this purpose we need the databases described above. In the RHDB there is data stored about Firefox, Thunderbird, Mozilla, etc. which are all products of the Mozilla Foundation. To retrieve only data from Mozilla, we need the information stored in the source code model databases. We retrieve the IDs of the files belonging to the Mozilla releases, map them in the `mapping_rhdb` table and get the corresponding RHDB file IDs.

Our model consists of three different groups of values: First, we need the LOC added and deleted for each file in a certain time slot, to calculate the effort. This data we retrieve out of the RHDB. Second, the number of files a module consists of, the LOC of a module, and the McCabe cyclomatic complexity of a module has to be retrieved. These are the values used in the initial model and are provided by the source code model databases. Third, we have to retrieve the change couplings and modification reports which build our extension. This data is queried in the RHDB again. In Chapter 4 we develop the CPCM, Chapter 5 provides further details about the data retrieval, and in Chapter 6 we present the validation.

Chapter 4

Change Prediction Cost Model

In this section we develop the Change Prediction Cost Model (CPCM) out of the cost model presented by Lucia et al. [LPSV02]. First, we highlight the previous knowledge needed for the understanding of the development of the CPCM, followed by the conceptual description of the different approaches of model extensions we investigated. The implementation and validation of the model will be presented in Chapter 5 and 6.

4.1 Prepare the Scene

While analysing the release history database, we were keeping an eye on applicable attributes to extend the original model. Our goal was to introduce a model that includes the dependencies of change couplings and modification reports.

4.1.1 Problem Reports

Whenever a bug was found in the Mozilla application, everybody is allowed to report and track it in the bug tracking system Bugzilla [1] developed by the Mozilla Foundation. There are guidelines to be followed, to help that the bugs will be fixed by the engineers.
A good bug report has two basic properties:

1. *Reproducible.* If an engineer can not see or better not reproduce the bug, the engineer will move on to the next bug report. Every information provided to a bug can help to locate and fix it.

2. *Specific.* The quicker the issue can be isolated by the engineer, the better. If one reports a crash on a site without indication about its accurate circumstances, the issue will not be fixed in an appropriate way.

An example of a bug report can be accessed on the bugzilla homepage [2]. It contains information like steps to reproduce the bug, results, regressions, configurations tested and a HTML snippet. Such information are helpful to the engineers to find and fix the bug fast.

The problem reports will be looked over by engineers. They try to reproduce the bug and locate its reason. When they managed to fix it they write a modification report in which they sum-up the work they have done to fix the bug.

[1] https://bugzilla.mozilla.org/ (last visited 12-02-2006)
[2] https://bugzilla.mozilla.org/show_bug.cgi?id=2683 (last visited 12-02-2006)

4.1.2 Modification Reports

Whenever a file was changed due to a problem report or any other maintenance reasons, the changed file is checked-in and committed to the CVS. For each commit a modification report (log entry) is created that contains the time of the commit, the name of the author, and a modification description. Figure 4.1 shows the developing of the number of modification reports over the time slots for the Mozilla module *NewLayoutEngine*.

4.1.3 Change Couplings

As already mentioned in Chapter 1.2.2, *change couplings* correspond to the interconnectedness of modules, sub-modules or files. In this thesis we involve the change couplings among files. They are calculated out of the modification reports stored in a CVS repository. In Figure 4.1 the progression of the change couplings over the time slots for the Mozilla module *NewLayoutEngine* is plotted.

> "Two files exhibit a change coupling if they are commonly committed, i.e., at the same time [3], by the same author, and with the same modification description." [FGP05] p.66

Additional information about the changes are needed to detect couplings that are caused by non-structural changes like renaming of variables in the whole system accomplished in a simple find-and-replace task or changes to the interface, concerning the whole system. Such couplings have to be filtered because they indicate false couplings. Only file-changes caused by structured changes like source code changes, concerning for example the functionalities of the system, have to be included to the investigation of change couplings. A strong coupling between files is indicated by files that are often changed together, detected by their common commitment. [FGP05]

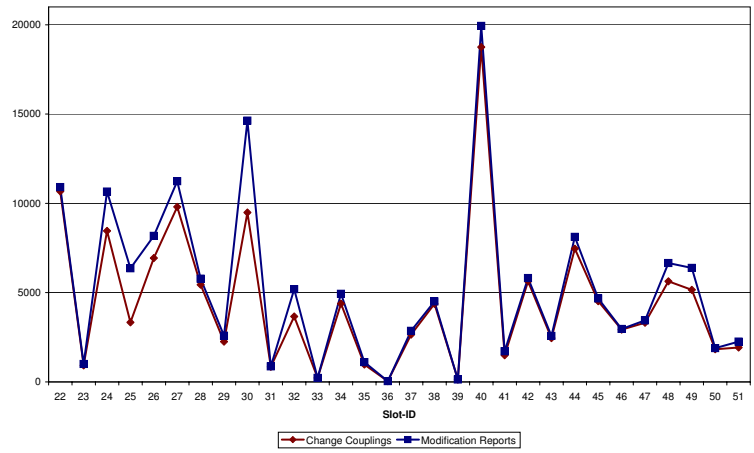

Figure 4.1: Mozilla Module NewLayoutEngine

The difference of the graphs is small. In each time slot the number of modification reports is on a higher level than the number of change couplings. This implies that not every modification report has its corresponding change coupling.

[3] In a time slot of about fifteen minutes.

4.1.4 Effort Definition

As effort we defined a value that reflects the changes, counted in LOC, that have been done on the files of a module, compared to the LOC of the whole module. The LOC of a module can change in two ways: First, files can be added, deleted or moved to another module. Second, code can be changed in the files, which can be determined in an increasing or decreasing number of LOC for each file. For the definition of the effort we are interested in the summed-up number of lines added or deleted in each file.

We defined the effort as:

$$Effort = \frac{LOC_{added} + LOC_{deleted}}{LOC} \tag{4.1}$$

With this formula we are able to determine the effort in a given time period. It provides us directly with the percentage rate of the changes done to the whole module. The LOC of the module multiplied by the percentage rate yields the absolute rate of changes.

4.2 Model Extension

In this chapter we present the conceptual layer of our CPCM. The data retrieval can be seen in Chapter 5. The validation of the extensions is presented in Chapter 6.

4.2.1 First Approach

First we extended the initial model from Lucia et al. with the two input variables *change couplings* and *modification reports*, each weighted with a model variable:

$$Eff = (\alpha_1 * SC + \alpha_2) * log(\overline{LOC}) + (\alpha_3 * SC + \alpha_4) * \overline{CYC} + \alpha_5 * SC +$$

$$\alpha_6 * ChangeCouplings + \alpha_7 * ModificationReports$$

The advantage of this approach is based upon the availability of the data. Each input value can be queried from the release history database. The effort on the left hand side of the equal sign, was calculated with the effort formula (4.1). The model variables $\alpha_1 \dots \alpha_5$ were provided by the initial model as listed in Table 2.3. We assumed that it will be possible to adapt the initial model to our demands with the extension of two additional input values and the introduction of their weighting variables α_6 and α_7. In this approach we imply the change couplings and modification reports overlapping the module boarders as shown in Figure 4.2. The rectangles represent modules containing files pictured by circles.

The grey module in the middle of the picture signifies the starting module. Every connection to a file belonging to a module other than the starting module and every connection from a file outside the starting module to a file inside the starting module is counted.

4.2.2 Second Approach

Relying on the results gained out of the first approach, we established the second approach. We found out that we had to readjust the variables $\alpha_1 \dots \alpha_5$ as well. A simple extension was not able to adapt the initial model to an acceptable degree of effort estimation. We removed the extension and concentrated on the initial model and the estimation of the five variables. For this calculation,

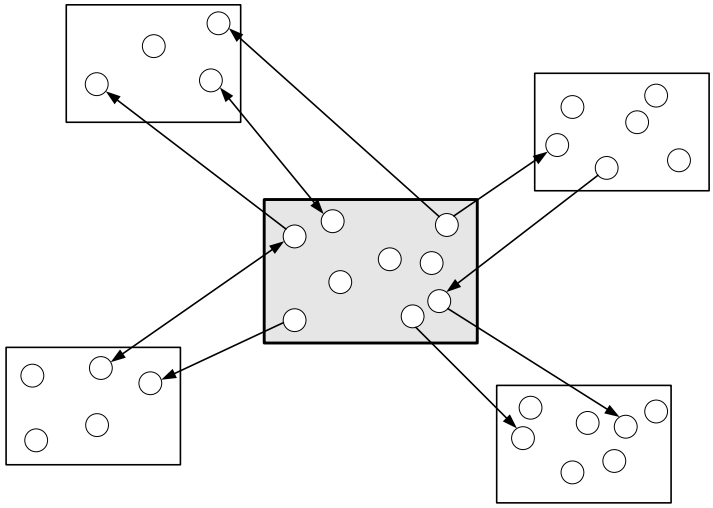

Figure 4.2: Change Couplings and Modification Reports

we wanted to use R, "a free software environment for statistical computing and graphics" [4]. To use this environment, we had to represent the system of equations by a matrix equation of the form $m\,x = b$. The number of columns in m represents the number of variables, and the number of rows give the number of equations [Wol88]. To provide the input in this format, we redefined the initial model:

$$Eff \quad = \quad \alpha_1 * SC * log(\overline{LOC}) + \alpha_2 * log(\overline{LOC}) + \alpha_3 * SC * \overline{CYC} + \alpha_4 * \overline{CYC} + \alpha_5 * SC,$$

In a next step we substituted the expressions:

$$\begin{aligned} Z &= \alpha_1 * A + \alpha_2 * B + \alpha_3 * C + \alpha_4 * D + \alpha_5 * E \\ \text{where:} & \\ Z &= Effort, \\ A &= SC * log(\overline{LOC}), \\ B &= log(\overline{LOC}), \\ C &= SC * \overline{CYC}, \\ D &= \overline{CYC}, \\ E &= SC. \end{aligned}$$

The effort on the left hand side of the equation was calculated with the formula (4.1). All the other input values, A . . . , E were obtained from the source code model database. The variables α_1 . . . α_5 then were calculated with the R-environment.

4.2.3 Third Approach

To diversify our ideas and to expand our perception, we met Guerino Mazzola before we went further on. He received his Ph.D. in mathematics at the University of Zurich and qualified as a

[4] http://www.r-project.org/ (last visited 05-03-2006)

professor in mathematics and also in computational science. To introduce statistical aspects we invited Christoph Luchsinger. He received his Ph.D. in mathematics at the University of Zurich. He first qualified as lecturer in mathematical finance and holds his lectures in probability and statistics in Basel and Zurich. With his help, we enhanced the ideas accumulated out of the meeting with G. Mazzola, towards our final CPCM. First we will have a glance at the approach we discussed with G. Mazzola, before we present the adding of the statistic point of view, introduced by Ch. Luchsinger.

Dimensions of the Change Couplings

G. Mazzola mentioned the idea of an analysis of different dimensions of coherences among the files of the models. We split up the model into different layers and analysed the specific performance of the measured aspects:

1. *Pre-Stage.* In an anticipated zero-dimension we examined the intrinsic value of the files first. That means, that we analyse the files unattached to any dependencies in a static way. The measured values are number of files per module, lines of code, and the McCabe cyclomatic complexity as they are represented in the initial model by Lucia et al. This point of view depicts actually the initial model, we use as basis for the development of the CPCM. Figure 4.3 clarifies the suggested zero-dimension. The surrounding rectangle represents the boarder of a module in a specific slot. Inside the module, the files belonging to this module are represented by circles of different diameters. The difference in size represents the different number of LOC a file contains. Bigger circles indicate files with a higher number of LOC, smaller circles indicate fewer LOC. All the values are computed for each of the thirty slots.

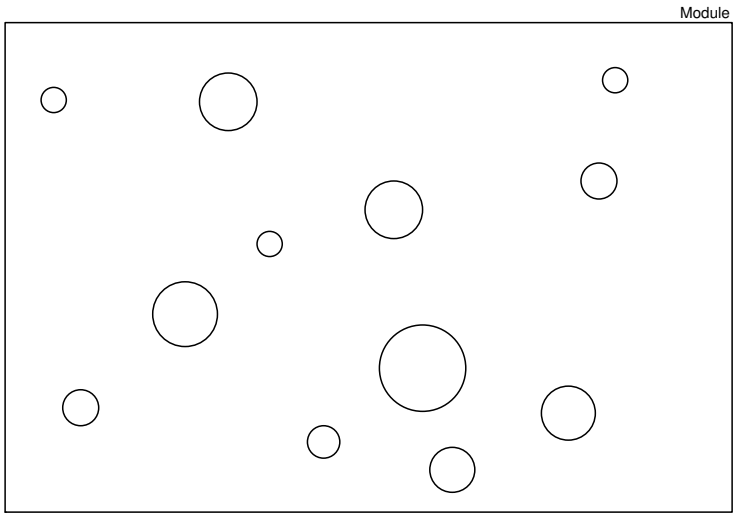

Figure 4.3: Module and Files - Pre-Stage

2. *The First-Dimension.* After the pre-stage, where we examined the files independently, we move on to the first layer and analyse the dependencies among the files. We examine the one-dimensional change couplings. In other words, the files changed in pairs. These couplings can be illustrated by connecting lines between two files. A line has only one volume

extension. That is the reason why we talk about one-dimensional couplings. Figure 4.4 plots the scene.

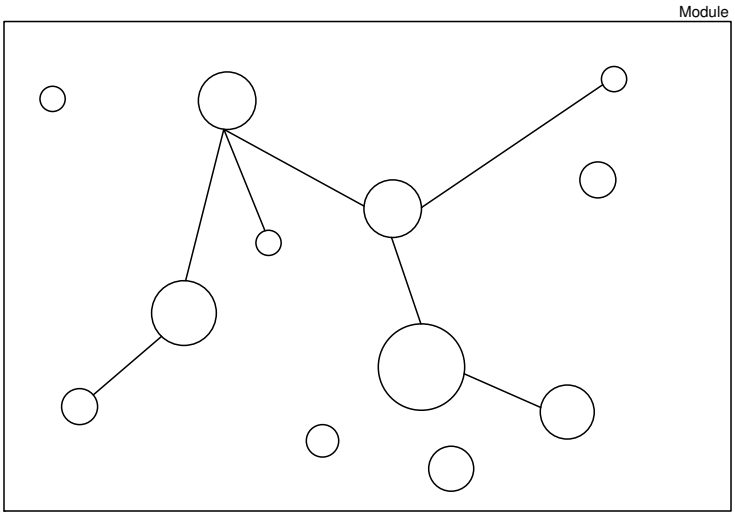

Figure 4.4: Module and Files - One Dimensional Connections

The files that were changed together during a specific time slot are linked with lines. The resulting graph represents all the files that were changed in pairs. When a file *i* is linked to three other files, it does not mean that those four files where changed together but that the file *i* was changed at different points in time during the observed slot together with one of the three linked files. The files not connected in the graph have no change couplings with other files. This is in contrast to our first approach in which we counted every occurring change coupling. In fact we count the number of lines.

3. *The Second-Dimension.* The second dimension is developed in the same way as the first one. We examine the files that are changed in triples. The connections among these three files exhibit a triangle. We count the number of groups of files changed in triples (triangles). Figure 4.5 presents this further extension.

4. *The X-Dimension.* The change couplings can be divided in as many dimensions as required or possible. It depends on the project that is analysed and the depth of the breakdown needed. In this thesis we counted the number of items belonging to a group up to the 15th dimension separately and summarised every higher dimension to one further group.

If required, the dimensions can be clustered and weighted. A possibility is to compute the first 15 dimensions separately and to summarise the dimensions higher than 15. Building three clusters merging the first and the second dimension, then the fourth up to the tenth and then summing up every dimension above ten is another grouping approach.
As it concerns the model, this approach provides us with a model that is built from the bottom-up and can be extended up to the dimension desired. We start at the pre-stage with the initial model and add the upcoming dimensions:

As it concerns the first-dimension we have to sum up all the change couplings between two files of the dimension one, starting with file *i* and ending with file *j*, where *i* is the first file and *j* is the last file belonging to a module. These are all the couplings belonging to the group of files changed in pairs.

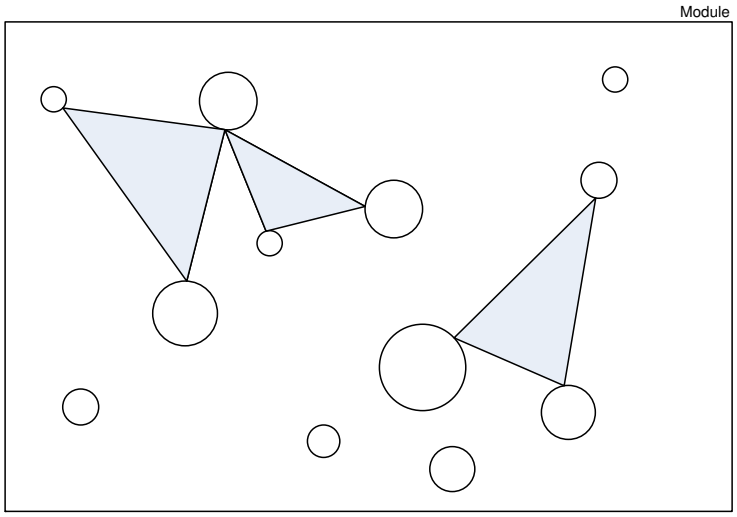

Figure 4.5: Module and Files - Two Dimensional Connections

Pre-Stage:	$Effort = M_0$ $M_0 = (\alpha_1 * SC + \alpha_2) * log(\overline{LOC}) + (\alpha_3 * SC + \alpha_4) * \overline{CYC} + \alpha_5 * SC$
First-Dimension:	$Effort = M_0 + \alpha_6 * (M_1)$ $M_1 = ChangeCouplings\_1dim$
Second-Dimension:	$Effort = M_0 + \alpha_6 * (M_1 + M_2)$ $M_2 = ChangeCouplings\_2dim$
X-Dimension:	$Effort = M_0 + \alpha_6 * (M_1 + M_2 + M_3 \dots)$

Weighting of the Groups

The connections can be weighted as described in the following example: Supposing that a file *i* was changed 20 times and a file *j* 30 times. 10 times the files were changed together, detected by their modification reports. The weighting α can be figured out the following way:

$$\alpha = (\frac{10}{20} + \frac{10}{30}) * \frac{1}{2} \tag{4.2}$$

The weighting describes the bidirectional probability that we have to adjust a file *j* when we have to modify a file *i* due to a maintenance task. The weighting of the two-dimensional connections can be built up the same way as described in formula (4.2). Equation 4.3 shows the calculation of the two-dimensional weighting in an universal example.

$$\alpha = (\frac{r}{a} + \frac{r}{c} + \frac{t}{a} + \frac{t}{b} + \frac{s}{b} + \frac{s}{c}) * \frac{1}{6} \tag{4.3}$$

As we can see, the complexity to calculate the weighting is increasing disproportionately. The parameters a, b and c denote the number of changes done to a file in the edges of the triangle. r, s and t denote the number of change couplings. r refers to the coupling between file a and c, s carefree to b and c, t refers to a and b. Figure (4.6) shows the coherences.

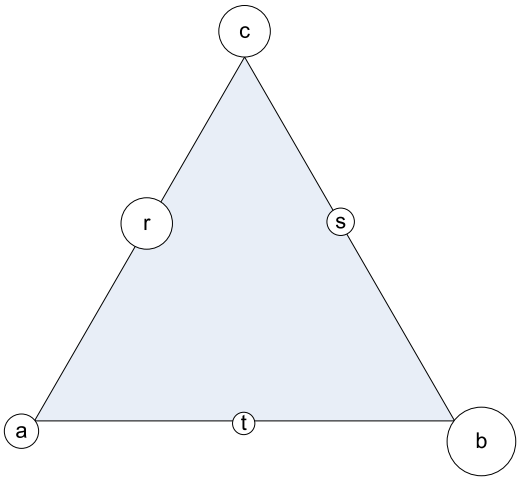

Figure 4.6: Weighting of Couplings

Statistical Aspect

Keeping the above approach in mind, we invited Ch. Luchsinger to provide us with inputs from a statistical point of view. The fact, that we do not know, how the model of de Lucia et al. was developed and what kind of input values were used, turned out to be a problem. The analysis of the model itself is easier and in other words even possible, if we knew where the causes were lying for such a development of the model. Ch. Luchsinger was of the opinion that the approach, we developed with G. Mazzola is interesting to apply statistics on it. The approach to cluster the couplings turned out to be inapplicable in this thesis due to an extraordinary work. It is an interesting input for future work.

In the scope of this thesis we will constrict our approach to the estimation of the couplings from the first dimension up to the 15th and to cluster every higher dimension. As weighting value we choose the output of the formula to calculate the number of edges needed to fully connect a given number of nodes:

$$Weighting = \frac{n * (n-1)}{2} = \begin{pmatrix} n \\ 2 \end{pmatrix} \tag{4.4}$$

The formula (4.4) calculates a value that rises disproportionately fast the more nodes are involved. This reflects the increasing complexity of maintaining multiple coupled files. Table 4.1 shows the relation between the number of files belonging to a group and their value of complexity.

If more than one file has to be changed, there can arise at least one dependency to another file. When more than three files are involved, the complexity rises faster than the number of files increase. Each change coupling group is multiplied by its value of complexity. We weight every change coupling group from dimension one to 15, one by one and cluster the dimensions above

Number of Files (n)	1	2	3	4	5	6	7
Value of Complexity	0	1	3	6	10	15	21

Table 4.1: Value of Complexity

15. In order of not to lose the number of files that were changed alone (n=1) due to a multiplication by zero, the value of complexity is raised to one.

$$Eff = (\alpha_1 * SC + \alpha_2) * log(\overline{LOC}) + (\alpha_3 * SC + \alpha_4) * \overline{CYC} + \alpha_5 * SC + \alpha_5 * ChangeCouplings_{weighted} \quad (4.5)$$

To verify the model we contacted the Statistical Consulting of the University of Zurich [5] to get some help in the use of the R-environment. The first requirement is the analysis of the extended initial model. This advancement should testify whether our extension yields an added value or not. In a second step we will check whether the structure of the extended model makes any sense when applied to our software project. We will show this with the functions in the R-environment.

4.3 Summary

In the first part of the chapter, we explained the meaning of problem reports, modification reports, and change couplings and their interrelationship. Furthermore the calculation of the effort was defined.

In the second part of the chapter, we presented the tree extensions of the model of Lucia et al. In a first approach, we extended the initial model with the two input variables *change couplings* and *modification reports*, each weighted with a model variable. In the second approach, we concentrated on the estimation of the model variables. In the third extension we developed in collaboration with G. Mazzola and Ch. Luchsinger a further input value – grouped change couplings – and moved towars statistical analysis.

The next chapter presents the implementation and data retrieval. The evaluation of the extensions is presented in Chapter 6.

[5] http://www.math.unizh.ch/baps/services/index-e.php (last visited 11-03-2006)

Chapter 5

Implementation

This chapter introduces Mozilla as the data source for the validation of the model and points out the database design and the linkage of the tables used. Furthermore we will give an insight in the implementation of the data retrieval coded in Java.

5.1 Mozilla

> "Mozilla was the original code name for the product that came to be known as Netscape Navigator, and later, Netscape Communicator. Later, it came to be the name of Netscape Communications Corporation's dinosaur-like mascot. Netscape Communications Corporation holds trademarks on the names Netscape, Navigator, and Communicator; it has not yet been decided what, if any, restrictions Netscape will place on the use of those names. Now, we use the name "Mozilla" as the principal trademark representing the Foundation and the official releases of internet client software developed through our open source project." [1]

There is a little confusion in the use of the name "Mozilla". "Mozilla" is a web-browser, e-mail and newsgroup client, IRC chat client, and HTML editor developed by the Mozilla Foundation. The foundation's mission is to coordinate and integrate the work of the developers. They provide a CVS and Bugzilla to synchronize and merge the work. In this data repositories, we find data from Firefox, Thunderbird, Camino, SeaMonkey etc. as they are all developed by the Mozilla Foundation. While querying the RHDB we have to pay attention on selecting only data belonging to the Mozilla web-browser.

We validated the development of the model on data out of seven Mozilla releases running from the 7th of June, 2001 to the 18th of March, 2004. These span of time was split-up into 30 time slots which build subclasses of the releases. Each slot takes about one month approximately. The breakdown into slots and the assignment to a release are listed in Table A.1 in the appendix. Furthermore, Mozilla is split-up in modules each containing several source files. Sometimes they are moved from one module to another, deleted or added, due to maintenance activities. The number of files a module contains varies over time. But in general, the modules can be sorted by the ascending amount of files containing as shown in Table 5.1.

The DOM-module is the largest one and consists of eight times more files and contains ten times more LOC than MathML which is the smallest module. In the middle there is the XSLT-module. It consists of three times more files than MathML and 2.5 times less files than DOM. As it

[1] http://www.mozilla.org/mission.html (last visited 12-03-2006)

	Av. Number of Files	Av. LOC per Module
MathML	56	17,412
XML	75	27,315
NewHTMLStyleSystem	95	46,877
XSLT	182	42,660
XPToolkit	217	84,959
NewLayoutEngine	230	158,146
DOM	450	182,539

Table 5.1: Mozilla Modules

concerns the LOC, XSLT contains 2.5 times more LOC than MathML and 4.3 times less LOC than DOM.

5.2 Database Design

As basis we used a linkage of databases described in the paper "Towards an Integrated View on Architecture and its Evolution" [PFG05]. The databases store the source code model and metrics data extracted from seven Mozilla release databases and the RHDB database called rcafe3.

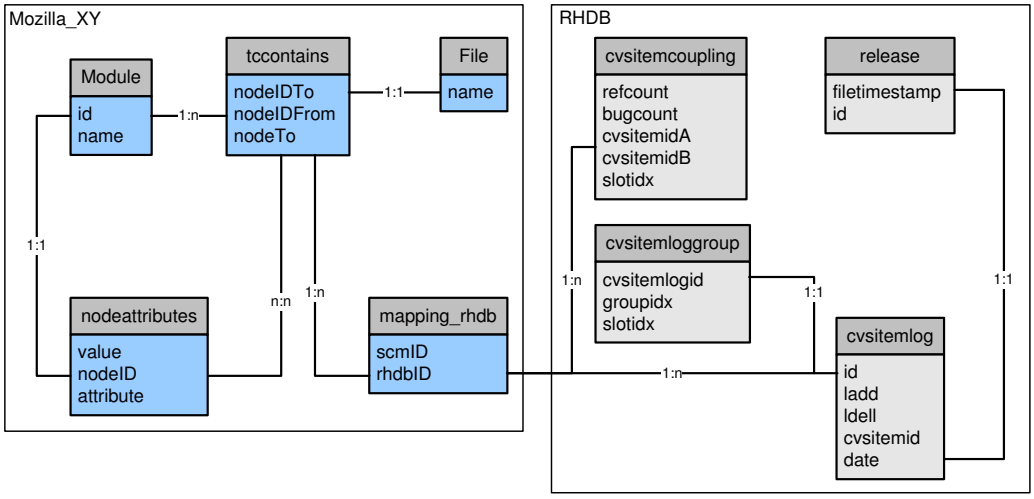

Figure 5.1: Database-Schema used for Data Retrieval

On the left hand side, the Mozilla release databases are shown. While the data we need is retrieved, each of the seven releases are queried. On the right hand side, the RHDB database is shown. All the data of the seven releases are stored in this tables. To retrieve data in the RHDB, the Mozilla-file-IDs have to be mapped in the table mapping_rhdb.

5.3 Queries

To get all the required data out of the databases, we developed a Java application. It is able to query all the seven Mozilla release databases and the RHDB which we query interrelated and

iterative. This requirement was solvable with the automation through a application.

5.3.1 General Part

We built Java classes that are able to query all the data required over all Mozilla releases, slots and modules. For this reason two static strings were built, containing the identifiers of all the releases or modules. The string containing the releases, has a length of thirty entries. Each entry maps to a slot running from 22 to 51. The release-slot-mapping can be seen in the Table A.1 in the Appendix.

To run the query over all modules and releases split-up into slots, three iterations where set up. The outmost handled the modules, followed by the slots and releases. This architecture enabled a data output sorted by modules, releases, and slots. It built the basis for every further query. The output provided a very clear and detailed overview over the queried data. In addition, the output is ordered by one slot per row and delimited by commas. The output can be saved in a simple text file, containing the data delimited by commas and simplifies the import to every further tool like Microsoft-Excel or can be directly imported to the R-environment.

Because the Mozilla releases were parsed separately each module has an ID that differs from release to release. This `ModuleID` is queried dynamically corresponding to the actual release and module. To retrieve all files a module consists of a further query is necessary.

```
SELECT nodeIDTo
FROM tccontains t, File f
WHERE t.nodeIDFrom = "ModuleID" and t.nodeTo = f.name
```

The table `tccontains` covers data about the files belonging to a module. To get these IDs of this files, the `nodeIDFrom` is compared with the current `ModuleID`. To be sure to get only files belonging to Mozilla, the `nodeTo` has to be compared to the name of the files in the table `File`, where all the files of the queried release are saved. The retrieved file IDs are stored in the string named strSCMIDs.

The data of the CVS and Bugzilla are stored in one database for all releases called `rcafe3` (RHDB). To query files in the `rcafe3` database the file-IDs out of the Mozilla releases have to be mapped on the table `mapping_rhdb` stored in each Mozilla release database . For further queries, we have to pay attention on the file-IDs we use. For queries in any Mozilla release database we have to use the IDs stored in the string `strSCMIDs`, on the RHDB we have to use the IDs stored in the string called `strRHDBIDs`. It is possible, that one `scmID` is mapping on two `rhdbIDs`. This is caused by the possibility, that files can be moved from one directory to another, which is represented by a multiple matching in the mapping table.

5.3.2 LOC added or deleted

To get the LOC added or deleted in a module of a specific time slot, the slot's starting and end time has to be queried out of the table `rcafe3.release`. As end time, we use the time stamp stored to a slot and use the time stamp of the previous slot as its starting time. The number of LOC added or deleted and the number of files concerned can be queried from the modification reports stored in the table `rcafe3.cvsitemlog`.

```
SELECT COUNT(cvs.id) itemNumber,
SUM(cvs.ladd) linesAdd,
SUM(cvs.ldel) linesDel
FROM rcafe3.cvsitemlog cvs
```

```
WHERE cvs.cvsitemid in("RHDBIDs")
AND cvs.date > "timestampFrom"
AND cvs.date <= "timestampTo"
```

The table `rcafe3.cvsitemlog` provides the LOC added or deleted in the attributes `ladd` and `ldel` respectively. All the LOC added or deleted have to be summed up over all files provided in the string `RHDBIDs`.

5.3.3 Lines of Code and McCabe cyclomatic complexity

Source code metrics such as the lines of code (LOC) or the McCabe cyclomatic complexity are computed by the parser and stored in the table `nodeattributes` for each Mozilla release and for each file.

```
SELECT sum(value) as LOC
FROM nodeattributes
WHERE nodeID in ("strSCMIDs") and attribute = 'imagix_FileLines'
```

The string `strSCMIDs` contains the IDs of all the files that belong to the queried module in a specific slot. To get the total number of LOC for the module, the LOC of each single file is summed up.

The complexity value can be retrieved from the same table. It is calculated for the whole module. The SQL statement above has to be adjusted to `attribute = 'imagix_FncCyclCmplx'`.

5.3.4 Change Couplings and Modification Reports

To quantify the change couplings and modification reports of a module we had to check all the one-dimensional relations of the files belonging to a module, from inside the module to another file outside the starting module and vice versa. We counted all the change coupling relationships of a module's files that cross the module boundary. For instance, we want to count the change couplings of the module "XML". Then the files of this module are investigated on common commits with files in modules other than "XML" and vice versa. The result are couplings pointing to files outside the original module and couplings pointing from files outside the module to the original module. Generalized we talk about couplings of files in direction from module A to B, whereas B can be every module apart from A, and B to A.

```
SELECT count(*) sumCCs, sum(refcount) + sum(bugcount) sumMRs
FROM rcafe3.cvsitemcoupling cvs
WHERE cvsitemidA in("strRHDBIDs")
AND cvsitemidB not in("strRHDBIDs")
AND slotidx in("slotID")

SELECT count(*) sumCCs, sum(refcount) + sum(bugcount) sumMRs
FROM rcafe3.cvsitemcoupling cvs
WHERE cvsitemidB in("strRHDBIDs")
AND cvsitemidA not in("strRHDBIDs")
AND slotidx in("slotID")
```

5.3.5 Change Coupling Groups

The change couplings retrieved in the section before were used in the first approach. In the third approach we decided to group the change couplings. To retrieve this data, two more queries had to be set up.

In the table `cvsitemloggroup` the files, identified over the attribute `cvsitemlogid` are attached to the same `groupidx` when they were committed together. In a first step, we select all the `groupidx` that have as many members as provided in the variable `GroupDimension`.

```
SELECT groupidx
FROM rcafe3.cvsitemloggroup
WHERE slotidx = "slotID"
GROUP BY groupidx HAVING COUNT(*) = "GroupDimension"
```

In a second step we count the `groupidx` in the table `cvsitemloggroup` that observe certain constraints. The `cvsitemid` has to be in the string `strRHDBIDs`. The `slotidx` is compared to the `slotID` we are retrieving. The `groupidx` has to lie in the range of possible groupidx, selected in the previous query (`strGroupFileIDs`). The IDs in the `cvsitemloggroup` and `cvsitemlog` are compared and have to match.

```
SELECT count(distinct cl.groupidx) number
FROM rcafe3.cvsitemloggroup cl, rcafe3.cvsitemlog log
WHERE cl.cvsitemlogid = log.id AND cl.groupidx in ("strGroupFileIDs")
AND cl.slotidx = "slotID" AND log.cvsitemid in ("strRHDBIDs");
```

5.4 Summary

In this chapter, we introduced the Mozilla Foundation and its products particularly the Mozilla web-broser. Then the data retrieval from the seven source code model databases and the RHDB is described. In details we present the queries for the LOC added and LOC deleted in a module, the LOC and the McCabe cyclomatic complexity of a module, the number of change couplings and modification reports, and the change couplings grouped by number of changes. This data is needed for the evaluation of the CPCM, presented in the next chapter.

Chapter 6

Validation

In this chapter we validate and discuss the three approaches introduced before. A discussion of the results at the end of the chapter will sum up the outcome. The calculation of the effort with the Formula (4.1) is the target. Out of the statistical point of view, the effort, on the left hand side of the cost model is the dependent value, which we try to declare as good as possible with the formula and the input values on the right hand side of the equal sign. In this validation we proof the approaches against the dependent value.

6.1 Prearrangement

Before we started to modify the initial model, we populated the database and retrieved the described metrics and estimated the effort with the initial model (2.1) and compared it to the effort calculated with the formula (4.1) over the changes of LOC.

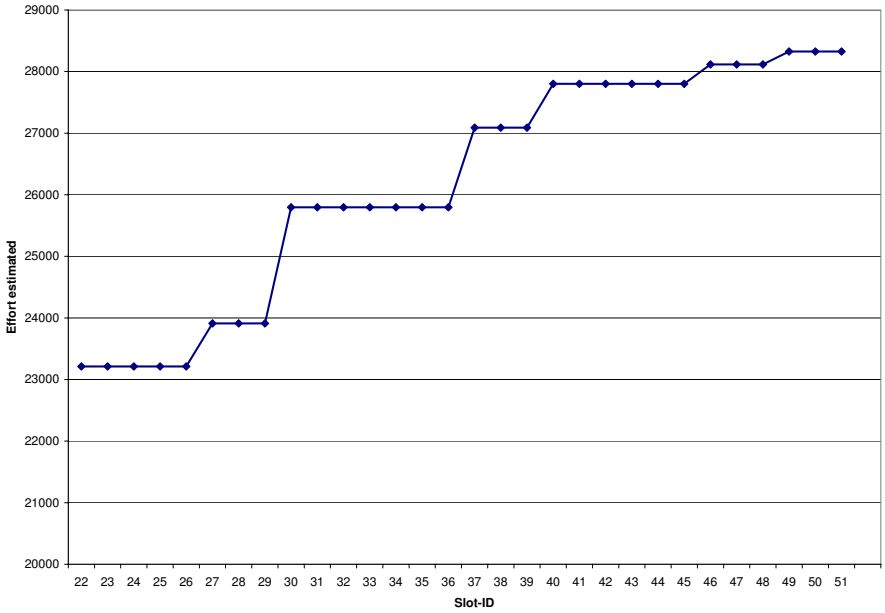

Figure 6.1: Mozilla Module DOM - Effort Estimation

Figure 6.1 shows the effort estimation of the initial model (2.1) for the DOM-Module. The dots of the same height are the slots belonging to the same release of the model. Unfortunately the database does not provide the needed data for every slot but only for a release as a whole. Slot 22 on the left side of the chart starts with an estimated effort of 23,212 and increases all along up to 28,327 in slot 51.

Figure 6.2 shows the calculated effort - the target values - out of formula (4.1) of the DOM-Module. It starts on the left side with the highest value of 0.43 and has another peak in slot 31 with 0.25. The values vary from 0.43 to 0.008.

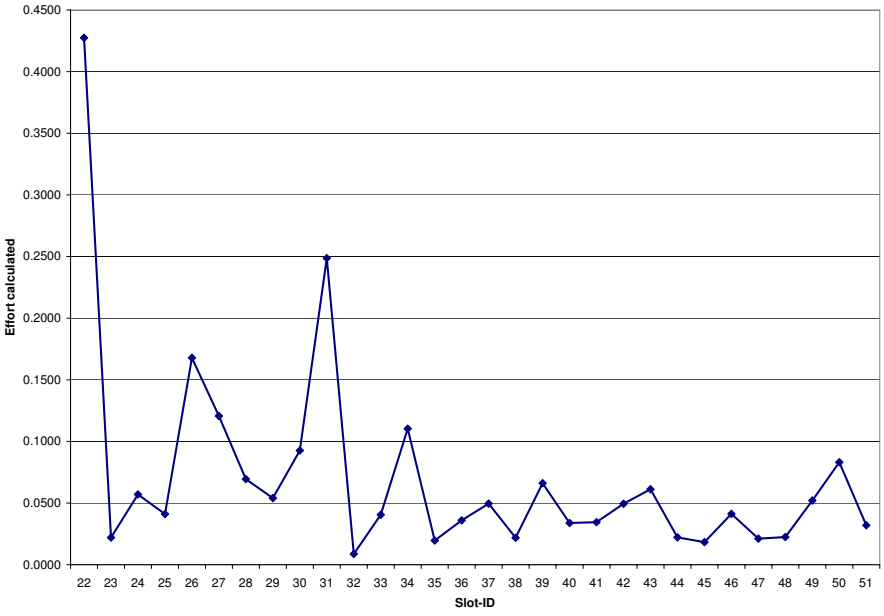

Figure 6.2: Mozilla Module DOM - Effort Calculation

6.1.1 Results

While comparing the two charts, the different development of the effort over the time slots is of interest. The target values shown in Figure 6.2 oscillate, while the estimation shown in Figure 6.1 increases steadily. The oscillation of the values denotes a changing effort over time, while the steady increase denotes an increasing effort over time.

6.2 First Approach

In the first approach we took the initial model and extended it with the two input values *change couplings* and *modification reports*. The target was to estimate the variables α_6 and α_7 and to calculate the effort.

After the data retrieval we were able to set up a system of equations containing 210 (30 time slots multiplied by 7 Mozilla modules) equations and the two undetermined module variables α_6 and α_7. We decided to validate the estimated variables on the module XSLT due to the fact that regarding the module size in number of files it lies in the middle of the modules as listed in Table

5.1. Due to some zero values in the change couplings or modification reports and the exclusion of the module XSLT, the system was reduced to 155 equations. We solved the system in a trivial way: We compared the first equation with the second one, and determined the two module variables α_6 and α_7. Then we went to the second equation and compared it with its following one and determined the two equations. Like this we went through the whole system and got in the end 155 different solutions for the two variables. We built the median and the mean value of the variables and pasted them into the extended formula and estimated the effort.

	α_6	α_7
mean value	-1800.90	1650.07
median	-13.03	9.20

Table 6.1: Extended Model Variables

The spread of the single results of the 155 calculations of the extended model variables was wide as it can be seen in the Table 6.1. The mean values are at least 100 times higher than their median. The median displays where the majority of the values are located. When we estimated the effort, the results we got using the mean values of the extended model variables were too far away from the target. The effort, estimated using the median values is plotted in Figure 6.3.

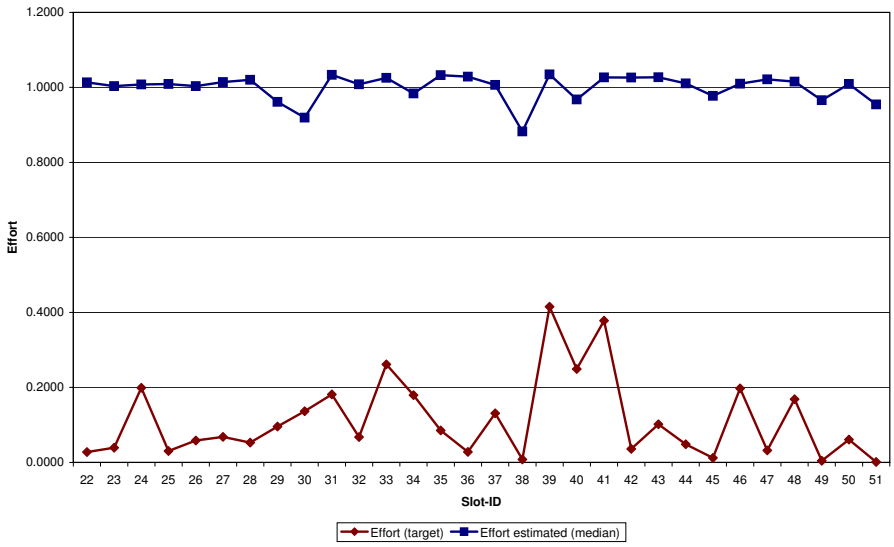

Figure 6.3: Effort Comparison of Module XSLT

The target effort is plotted on its real level. To get the estimated effort into the same chart we transformed its values by keeping up its relations. In details, the values were all divided by the mean value (343,651) of the sum of the estimated efforts. The values of the estimated effort plotted in the chart have to be multiplied by 343,651.

6.2.1 Results

In this first approach, we thought, that we will be able to match the target values, by simply adding an extension to the initial model, without touching the variables $\alpha_1, \ldots, \alpha_5$. By the analysis of the results, we realized, that we will not be able to suit our approach to the target value only by

adding two more input values. To adapt the model to our purpose a redetermination of the five module variables $\alpha_1, \ldots, \alpha_5$ is inevitable. The trend of the target effort could not be represented by the effort estimation.

6.3 Second Approach

As we found out in our first approach, we removed the extension and concentrated on the original model and the estimation of the five variables. This time we used R, "a free software environment for statistical computing and graphics" [1]. To use this environment, we have to represent the system of equations by a matrix equation of the form $m\,x = b$. The number of columns in m represent the number of variables, and the number of rows give the number of equations [Wol88]. To provide the input in this format, we have to redefine the original model.

$$
\begin{aligned}
Z &= \alpha_1 * A + \alpha_2 * B + \alpha_3 * C + \alpha_4 * D + \alpha_5 * E \\
\text{where:} & \\
Z &= Effort, \\
A &= SC * log(\overline{LOC}), \\
B &= log(\overline{LOC}), \\
C &= SC * \overline{CYC}, \\
D &= \overline{CYC}, \\
E &= SC.
\end{aligned}
$$

The values for A, B, C, D, and E can be queried out of the RHDB, Z is calculated by our effort model. We get an overdetermined, rectangular matrix, where the number of equations is more than the number of variables [Wol88]. The *qr.solve*-function [Tea03], provided by the R-environment can handle non-square matrix systems. As we get more than one solution, the algorithm provides the best least-squares solution.

First we set up the input tables. We populate the database and put the results in excel-sheets. We built up two scenarios: In the first one, we calculated the matrices differentiated by the modules. In the second one, we calculated the unknowns for the whole Mozilla project. The excel-sheets were saved with the extension .cvs to import them into the R-environment and assign them to a matrix. We get a system of seven times 30 x 5 matrices in the first scenario and one matrix of 210 x 5 in the second scenario.

We used two approaches for the calculation of the module variables. In the first approach we calculated the variables for the given six (without XSLT) Mozilla modules and determined the mean and median values. In the second approach we computed the module variables over all modules. Table 6.2 lists the values obtained through the first approach and Table 6.3 the values for the second approach. With the two different approach we obtained three different models that we evaluated with the XSLT module. Figure 6.4 shows the results.

We skip the graph showing the effort calculation with the mean value because all values are negative therefore useless.

The red graph (identified by the diamond sign) represents the target. It is the calculated effort of the "XSLT" module and should be reached as near as possible. The blue graph (identified by the square) represents the graph estimating the effort with the calculated unknowns split up by modules. For the first 17 slots its value lies above the target and below the target for the last 13 slots. Actually it lies below the x-axis from slot 46 on upwards.

[1] http://www.r-project.org/ (last visited 05-03-2006)

	α_1	α_2	α_3	α_4	α_5
MathML	0.10486608	-8.84359712	-0.01066350	0.62635221	0.11452564
XML	-0.01106659	-0.30690100	-0.00009498	0.01186688	0.03504087
NewHTMLStyleSystem	-0.03688799	-0.96255915	-0.00015719	0.02795536	0.11444189
XPToolkit	-0.07287829	13.92826393	0.00366768	-0.77221923	0.01758780
NewLayoutEngine	0.27334600	-55.60539798	-0.00872103	2.01849582	-0.09392555
DOM	-0.00305802	0.50125960	0.00002170	-0.01267048	0.00563350
mean value	0.042386866	-8.548155287	-0.002657885	0.316630093	0.032217358
median	-0.00706230	-0.63473008	-0.00012608	0.01991112	0.02631434

Table 6.2: Calculation of the Unknowns split up by Modules

	α_1	α_2	α_3	α_4	α_5
All Modules	-0.00123348	-0.01060003	0.00000521	0.00132887	0.00293442

Table 6.3: Calculation of the Unknowns over all Modules in one Piece

The green graph (identified by the triangle), which represents the effort calculated with the unknowns calculated unseparated, equals almost to the grid line with the value 0.0500.

6.3.1 Results

This chart points out, that it is a must to redetermine the unknowns alpha 1 to 5 for every new sofware project where we want to estimate the effort. One might think that the blue graph seems to be a better estimation due to its volatility. The green graph seems to be too smooth. The assimilation to the oscillation of the target effort has to be adjusted by the extension of the model, which we present in the next section.

6.4 Third Approach

In this approach we analyse the model with its extension of the grouped change couplings as introduced in the Formula (4.5) and estimate the parameters with the *lm-function* (linear model) of the R-environment. This function is used for linear regression analysis.

We start the analysis of the model with an import of two datasets. Each dataset consists of 210 rows – a row for each of the 30 time slots for each of the seven Mozilla modules. One dataset contains the effort calculated with the effort formula. The second dataset consists of six columns, containing the values A to F, where

$$\begin{aligned} A &= SC * log(\overline{LOC}), \\ B &= log(\overline{LOC}), \\ C &= SC * \overline{CYC}, \\ D &= \overline{CYC}, \\ E &= SC, \\ F &= ChangeCouplings_{weighted}. \end{aligned}$$

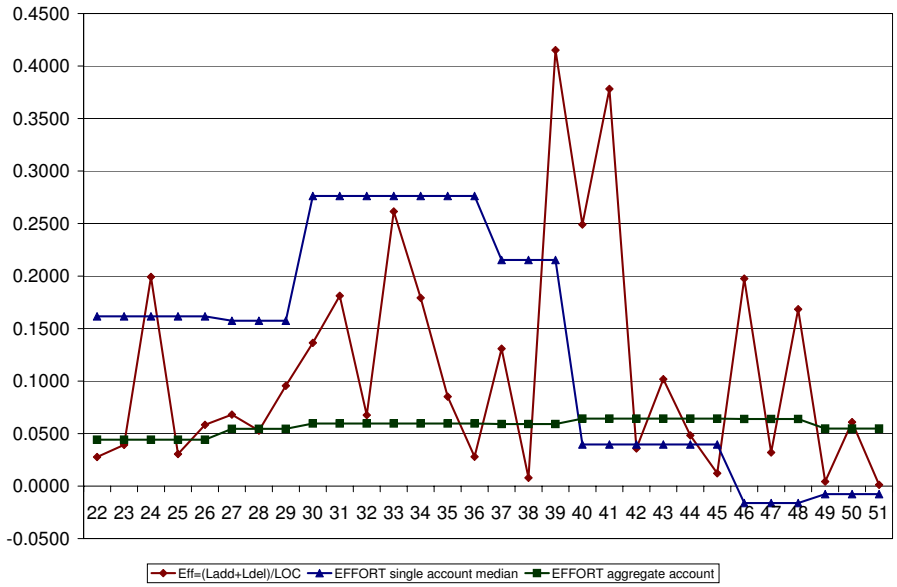

Figure 6.4: R Results - Effort Comparison

To import the datasets into the R-environment, they have to be saved as a .cvs document and stored in the R-environment directory. The following R-environment statements assign the dataset, containing the values A ... F to the variable *AtoF* and the dataset, containing calculated effort to the variable *Eff*. The imported dataset do not contain a header and are separated by semicolons.

```
> AtoF <- read.csv("AtoF.csv", header=FALSE, sep=";");
> Eff <- read.csv("Effort.csv", header=FALSE, sep=";");
```

Before we can apply the linear model, the presentation of the effort has to be changed.

```
> Eff=Eff$V1
```

The syntax of the linear model (*lm*)is similar to an equation. First, we indicate (*Eff*) as the value we want to describe. Then a ˜ follows, which means "is described by". Then the describing variables follow. As we did not provide any headers in the datasets we imported before, the R-environment automatically assigns the parameters V1 ... V6 to the imported columns. V1 maps to A, V2 maps to B, and so on. At the end we define the data source.

```
> lm01 = lm(Eff ~ V1 + V2 + V3 + V4 + V5 + V6, data=AtoF);
```

The output assigned to the variable *lm01* contains an intercept and the values of α_1 to α_6 listed in Table 6.4.

The result of *lm* is a *model object*, encapsulating the result of the model fit. The desired information can be obtained using *extractor functions*. We use the extractor function *summary* [Dal02].

Coefficients:	Intercept	V1	V2	V3	V4	V5	V6
Values:	-3.839e-01	-2.617e-03	1.525e-01	8.578e-06	8.201e-04	6.346e-03	-9.074e-07

Table 6.4: Coefficients of the Extended Model

```
> summary(lm01)

Call:
lm(formula = Eff ~ V1 + V2 + V3 + V4 + V5 + V6, data = AtoF)


 Residuals:
 Min            1Q            Median         3Q         Max
 -0.12188        -0.04336      -0.02095       0.01283    0.66617

 Coefficients:
                Estimate       Std.  Error    t value    Pr(>|t|)
 (Intercept)     -3.839e-01    9.181e-01      -0.418     0.676
 V1              -2.617e-03    1.875e-03      -1.396     0.164
 V2              1.525e-01     3.847e-01      0.396      0.692
 V3              8.578e-06     8.400e-06      1.021      0.308
 V4              8.201e-04     1.357e-03      0.605      0.546
 V5              6.346e-03     4.466e-03      1.421      0.157
 V6              -9.074e-07    4.656e-06      -0.195     0.846

Residual standard error:  0.09046 on 203 degrees of freedom
Multiple R-Squared:  0.1256, Adjusted R-squared:  0.09971
F-statistic:  4.858 on 6 and 203 DF, p-value:  0.0001167
```

The summary function provides us with a number of statistical information. We are interested in the regression coefficients and their *p*-values that can be seen on the outer right column. The value indicates the significance of the coefficients in the analysed model. If $(p < 0.5)$ the coefficient is significant, if $(p > 0.5)$ the coefficient is not significant. The coefficient that is the least significant is V6 which are the weighted change couplings we added to the model. This signifies that the extension of the model does not provide any added value to the cost estimation. We remove the coefficient V6 and assign the resulting linear model, which is the initial model, to the variable *lm02*.

```
> lm02 = update(lm01, ~ .  -V6)

> lm02 = lm(formula = Eff ~ V1 + V2 + V3 + V4 + V5, data = AtoF)
```

Table 6.5 lists the coefficients and their values for the initial model.

Coefficients:	Intercept	V1	V2	V3	V4	V5
Values:	-3.586e-01	-2.573e-03	1.419e-01	8.416e-06	8.442e-04	6.238e-03

Table 6.5: Coefficients of the Initial Model

The summary of *lm02* lists a *p*-value of 0.709 for the coefficient V2 which we eliminate. In the next step V3 is eliminated too. After this step, the remaining coefficients are all significant. If

we apply the model of de Lucia et al. on the data of the Mozilla releases, we get the following adjusted model:

$$Eff = -0.047973 + (-0.001502) * SC * log(\overline{LOC}) + 0.001773 * \overline{CYC} + 0.003957 * SC$$

Figure 6.5 shows the efforts for the module XSLT. The estimation of the effort module with the extension, the initial model, and the optimised model are close to each other. We can not see an added value when we extend the model with the grouped change couplings, as already indicated by the p-value. Furthermore, we can not estimate the outliers, we are just able to estimate an average value.

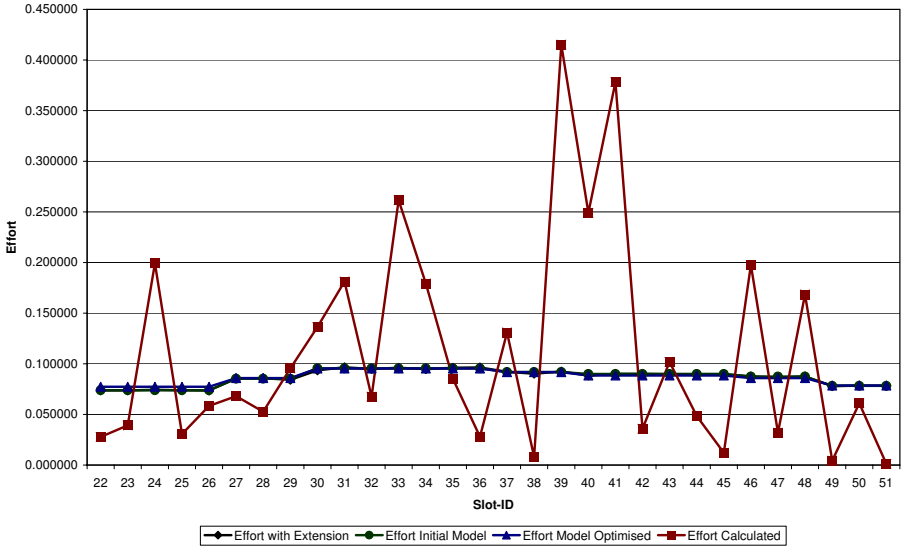

Figure 6.5: Effort Comparison

6.4.1 Results

The regression analysis generates two results: First, the simple extension of the initial model with the weighted change couplings does not provide an added value. Second, the initial model applied on the Mozilla model allows an adjustment of the initial model due to insignificance of input values.

6.5 Discussion of Results

In the first approach we noticed that we can not simply add an additional input value to adapt the model to our purpose. However, we got an insight into the output of the cost estimation model and compared the plots of our cost model to the initial model. The developing of the values of our cost model over the time slots is plotted in a to and fro, while the plot of the extended model is almost a flat line.

The second approach was a step into the right direction. The calculation of the model variables $\alpha_1 \ldots \alpha_5$ through the R-environment delivered valuable outputs. We could not determine,

whether the results are more accurate when we calculate the model variables for each module and build the mean value and the median afterwards out of these single results or when we calculate the model variables over all modules in one step. The effort calculation with the mean value model variables was negative over all the time slots. The remaining model variable calculations yielded valuable results and emphasised the approach to use the R-environment for our further analyses.

In the third approach we provide a statistical evaluation with the result that the extension of the initial model did not provide an added value. Which does not mean that the idea of an extension with data out of the RHDB does not work. We only examined an addition of the grouped change couplings to the initial model. It has to be examined in which combination with other input values we can establish an added value to the cost estimation model. Furthermore, we optimised the initial model of Lucia et al. through the elimination of insignificant input values. It signifies that the initial model is not universally valid to every software project. In addition to the second approach, with which we found out, that the model variables have to be redetermined according to the software project, the third approach introduces the assumption that the concatenation of the input variables can be optimised or that input variables can be neglected. This assumption is verified for the Mozilla project and has to be certified for other projects.

Approach one and two turned out to be too trivial to analyse the complex structure and dependencies of a cost model. Nevertheless, they provided a first insight to the performance of the output of the initial model and our cost model. The third approach yields valuable results and represents a basis for future work. It has to be mentioned that the calculation of the effort we introduced has to be checked on adequateness. The use of ”real” effort information would be desirable to improve our analysis results and cost models.

Chapter 7

Conclusion

7.1 Contribution

This thesis provides an insight in the area of cost estimation based on change couplings. We elaborated three approaches to extend the initial model and presented methods to proof them. As there exist no tools that could be used to solve the problems, we had to find a way to solve them. The following depicts the main contributions:

- A database schema of the databases used and the description of their feeders,
- A detailed description of the implementation of the data retrieval,
- The introduction of the R-environment and its use, and
- Traces for further developments.

The description of the databases used and the database schema build the basis for the understanding of the data retrieval. Due to a lack of documentation, the understanding of the connections of the different tables is impossible without expert knowledge. The understanding of the database builds the basis for any data retrieval.

With the R-environment, we present a powerful statistic tool. In this thesis we only provide an introduction to its use. With the usage of R, it will be possible to carry out further research.

7.2 Lessons Learned

In this thesis we found out how to develop a CPCM, through trial and error. The area of cost models is complex and a number of values carry out hidden influence on the development of the cost estimation's accuracy. These hidden influences can be detected with the help of statistical analysis. We tried different approaches and found out, which way leads to the target. The third approach turned out to be the most promising one. The development of a cost model is not just extending an existing model. We found out, that the extension of the initial model with the weighted change couplings did not provide an added value to the accuracy of the cost model. This result depicts only, that the approach did not deliver the designated result.

7.3 Future Work

This thesis is a preliminary work and uncovers a number of future work:

- Further options of grouping and weighting the change couplings introduced in the third approach might be of interest. The groups can be built according to programmer's experiences while maintaining software systems. The question is, where to set the boundaries of the groups.
- The different dimensions of the change couplings groups could be visualized to detect strong change couplings among files. Interesting would be the visualisation in the second dimension, where we examine the triangles.
- With the use of the R-environment, further developments of models can be evaluated.
- Other metrics than change couplings could be queried from the RHDB and introduced in development of cost models.

Appendix A

Tables

Release	Slot-ID	Name	Filetimestamp
Mozilla_092	22	Mozilla 0.9.1	2001-06-18 22:23:43
Mozilla_092	23	Mozilla 0.9.2	2001-06-30 03:02:08
Mozilla_092	24	Mozilla 0.9.3	2001-08-03 23:42:05
Mozilla_092	25	Mozilla 0.9.4	2001-09-15 00:17:44
Mozilla_092	26	Mozilla 0.9.5	2001-10-15 23:33:21
Mozilla_097	27	Mozilla 0.9.6	2001-11-21 08:33:57
Mozilla_097	28	Mozilla 0.9.7	2001-12-22 03:26:04
Mozilla_097	29	Mozilla 0.9.8	2002-02-05 00:59:14
Mozilla_10	30	Mozilla 0.9.9	2002-03-11 23:22:33
Mozilla_10	31	Mozilla 1.0	2002-05-31 21:17:45
Mozilla_10	32	Mozilla 1.1 Alpha	2002-06-12 20:39:18
Mozilla_10	33	Mozilla 1.1 Beta	2002-07-23 20:59:45
Mozilla_10	34	Mozilla 1.1	2002-08-27 23:59:20
Mozilla_10	35	Mozilla 1.2 Alpha	2002-09-12 20:56:19
Mozilla_10	36	Mozilla 1.2 Beta	2002-10-18 18:28:47
Mozilla_13a	37	Mozilla 1.2	2002-12-01 00:25:55
Mozilla_13a	38	Mozilla 1.3 Alpha	2002-12-13 20:16:50
Mozilla_13a	39	Mozilla 1.3 Beta	2003-02-11 00:27:52
Mozilla_14	40	Mozilla 1.3	2003-03-13 16:56:07
Mozilla_14	41	Mozilla 1.4 Alpha	2003-04-02 04:54:13
Mozilla_14	42	Mozilla 1.4 Beta	2003-05-08 21:22:04
Mozilla_14	43	Mozilla 1.4	2003-07-01 00:42:28
Mozilla_14	44	Mozilla 1.5 Alpha	2003-07-22 21:20:48
Mozilla_14	45	Mozilla 1.5 Beta	2003-08-28 00:11:16
Mozilla_16	46	Mozilla 1.5	2003-10-15 22:39:08
Mozilla_16	47	Mozilla 1.6 Alpha	2003-10-31 19:07:59
Mozilla_16	48	Mozilla 1.6 Beta	2003-12-09 11:08:46
Mozilla_17	49	Mozilla 1.6	2004-01-16 11:46:08
Mozilla_17	50	Mozilla 1.7 Alpha	2004-02-20 03:34:35
Mozilla_17	51	Mozilla 1.7 Beta	2004-03-18 17:37:54

Table A.1: Mozilla releases and related Slot-IDs

References

[Bai98] Jongmoon Baik. COCOMO II Model Definition Manual. Center for Software Engineering, University of Southern California, 1998.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, October 1981.

[BR00] K. H. Bennett and V.T Rajlich. Software Maintenance and Evolution: a Roadmap. In *Future of Sofware Engineering*, pages 75–87, Limerick, Ireland, 2000. ACM, Communications of the ACM.

[CSFP02] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. Version Control with Subversion. http://svnbook.red-bean.com/, 2002.

[Dal02] Peter Dalgaard. *Statistics and Computing – Introductory to Statistics with R*. Springer Science+Business Media, Inc., 2002.

[ea05] Per Cederqvist et al. Version Management with CVS. http://www.cvshome.org/docs/manual/, 2005.

[FGP05] Beat Fluri, Harald C. Gall, and Martin Pinzger. Fine-grained Analysis of Change Couplings. In *Proceedings of the 2005 Fifth IEEE International Workshop on Source Code Analysis and Manipulation*, pages 66–74, Department of Informatics, University of Zurich, September 2005. IEEE, IEEE Computer Society.

[FPG03] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM)*, pages 23–32, Amsterdam, The Netherlands, September 2003. IEEE, IEEE Computer Society.

[Kem87] Chris F. Kemerer. An Empirical Validation of Software Cost Estimation Models. pages 416–429. ACM, Communications of the ACM, 1987.

[KLT03] Jussi Koskinen, Henna Lahtonen, and Tero Tilus. Software Maintenance Cost Estimation and Modernization Support. In *Eltis Technical Project Report*, pages 1–60, Juväskylä, August 2003. Information Technology Research Institute, Univ of Juväskylä.

[KS99] Chris F. Kemerer and Sandra Slaughter. An Empirical Approach to Studying Software Evolution. In *IEEE Transactions on Software Engineering, Vol. 25, No.4*, pages 493–509, University of Pittsburgh, Pittsburg, PA 15260, July/August 1999. IEEE, IEEE Computer Society.

[LPSV02] Andrea De Lucia, Massimiliano Di Penta, Silvio Stefanucci, and Gabriele Venturi. Early Effort Estimation of Massive Maintenance Processes. In *Proceedings of the International Conference on Software Maintenance*, pages 234–237, Department of Engineering, University of Sannio, 2002. IEEE, IEEE Computer Society.

[Org95] ISO International Standards Organisation. Information technology - Software life cycle processes. *ISO/IEC NP 12207–1995*, 1995.

[Org98] The Mozilla Organization. Bug Tracking System – Bugzilla. http://www.bugzilla.org/, 1998.

[Par94] David Lorge Parnas. Software Aging. pages 279–287, McMaster University, Hamilton, Ontario, Canada, 1994. Communications Research Laboratory, Dept. of Electrical and Computer Engineering, IEEE Computer Society.

[PFG05] Martin Pinzger, Michael Fischer, and Harald Gall. Towards an integrated view on architecture and its evolution. *Electronic Notes in Theoretical Computer Science*, 127(3):183–196, April 2005.

[rge95] Magne Jørgensen. Experience With the Accuracy of Software Maintenance Task Effort Prediction Models. In *IEEE Transactions on Software Engineering, Vol. 21, No. 8*, pages 674–681. IEEE, IEEE Computer Society, August 1995.

[Soc98] IEEE Computer Society. IEEE Standard for Software Maintenance. *IEEE Std 1219–1998*, 1998.

[Swa76] E. Burton Swanson. The Dimensons of Maintenance. pages 492–497, University of Los Angeles, California, CA 90024, 1976. Graduate School of Management.

[Tea03] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Development Core Team, 2003.

[Wol88] Stephen Wolfram. *Mathematica*. Addison-Wesley Publishing Company, second edition edition, August 1988.