University of Oslo
Department of Informatics

Estimating Object-Oriented Software Projects with Use Cases

Kirsten Ribu

Master of Science Thesis

7th November 2001

Abstract

In object-oriented analysis, use case models describe the functional requirements of a future software system. Sizing the system can be done by measuring the size or complexity of the use cases in the use case model. The size can then serve as input to a cost estimation method or model, in order to compute an early estimate of cost and effort.

Estimating software with use cases is still in the early stages. This thesis describes a software sizing and cost estimation method based on use cases, called the 'Use Case Points Method'. The method was created several years ago, but is not well known. One of the reasons may be that the method is best used with well-written use cases at a suitable level of functional detail. Unfortunately, use case writing is not standardized, so there are many different writing styles. This thesis describes how it is possible to apply the use case points method for estimating object-oriented software, even if the use cases are not written out in full. The work also shows how use cases can be sized in alternative ways, and how to best write use cases for estimation purposes. An extension of the method providing simpler counting rules is proposed.

Two case studies have been conducted in a major software company, and several student's projects have been studied in order to investigate the general usefulness of the method and its extension. The results have been compared to results obtained earlier using the method in a different company. The investigations show that the use case points method works well for different types of software.

Data from the various projects have also been used as input to two commercial cost estimation tools that attempt to estimate object-oriented projects with use cases. The goal was to select a cost estimation method or tool for a specific software company. The findings indicate that there is no obvious gain in investing in expensive commercial tools for estimating object-oriented software.

Acknowledgements

This thesis was written for my Master of Science Degree at the Department of Informatics, the University of Oslo.

I would like to thank my advisors Bente Anda and Dag Sjøberg for their support and co-operation, with special thanks to Bente who let me use her research work for some of the investigations. I would also like to thank everybody who contributed to this work by sharing their experience and ideas with me: the participants on the various projects, fellow student Kristin Skoglund for practical help, and Johan Skutle for his interest and co-operation.

I want to thank my children Teresa, Erika and Patrick for their patience and good-will during the last hectic weeks, and my special friend Yngve Lindsjørn for his support all the way.

Oslo November 2001

Kirsten Ribu

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Estimates of cost and schedule in software projects are based on a prediction of the size of the future system. Unfortunately, the software profession is notoriously inaccurate when estimating cost and schedule.

Preliminary estimates of effort always include many elements of insecurity. Reliable early estimates are difficult to obtain because of the lack of detailed information about the future system at an early stage. However, early estimates are required when bidding for a contract or determining whether a project is feasible in the terms of a cost-benefit analysis. Since process prediction guides decision-making, a prediction is useful only if it is reasonably accurate [FP97].

Measurements are necessary to assess the status of the project, the product, the process and resources. By using measurement, the project can be controlled. By determining appropriate productivity values for the local measurement environment, known as *calibration*, it is possible to make early effort predictions using methods or tools.

But many cost estimation methods and tools are too difficult to use and interpret to be of much help in the estimation process. Numerous studies have attempted to evaluate cost models. Research has shown that estimation accuracy is improved if models are calibrated to a specific organisation [Kit95]. Estimators often rely on their past experience when predicting effort for software projects. Cost estimation models can support expert estimation. It is therefore of crucial interest to the software industry to develop estimation methods that are easy to understand, calibrate, and use.

Traditional cost models take software size as an input parameter, and then apply a set of adjustment factors or 'cost drivers' to compute an estimate of total effort. In object-oriented software production, use cases describe functional requirements. The use case model may therefore be used to predict the size of the future software system at an early development stage. This thesis describes a simple approach to software cost estimation based on use case models: the 'Use Case Points Method'. The method is not new, but has not become popular although it is easy to learn. Reliable estimates can be calculated in a short time with the aid of a spreadsheet. One of the reasons that the method has not caught on may be that there are no standards for use case writing. In order for the method to be used effectively, use cases must be written out in full. Many developers find it

difficult to write use case descriptions at an appropriate level of detail.

This thesis describes how a system may be sized in alternative ways if the use case descriptions are lacking in detail, and presents guidelines for writing use cases for estimation purposes. An extension of the use case points method with simplified counting rules is also proposed.

A variety of commercial cost estimation tools are available on the market. A few of these tools take use cases as input. The use case points method and two tools have been subjected to a feature analysis, in order to select the method or tool which best fits the needs of the software company where the case studies described in this work were conducted.

1.1 The Problem of Object-Oriented Software Estimation

Cost models like COCOMO and sizing methods like Function Point Analysis (FPA) are well known and in widespread use in software engineering. But these approaches have some serious limitations. Counting function points requires experts. The COCOMO model uses lines of code as input, which is an ambiguous measure. None of these approaches are suited for sizing object-oriented or real-time software.

Object-oriented analysis and design (OOAD) applies the Unified Modeling Language (UML) to model the future system, and use cases to describe the functional requirements. The use case model serves as the early requirements specification, defining the size of the future product. The size may be translated into a number, which is used to compute the amount of effort needed to build the software.

In 1993 the 'Use Case Points' method for sizing and estimating projects developed with the object-oriented method was developed by Gustav Karner of Objectory (now Rational Software). The method is an extension of Function Point Analysis and Mk II Function Point Analysis (an adaption of FPA mainly used in the UK), and is based on the same philosophy as these methods. The philosophy is that the functionality seen by the user is the basis for estimating the size of the software.

There has to date been little research done on the use case points method. Applying use cases or use case points as a software sizing metric is still in the early stages. A few cost estimation tools apply use case point count as an estimation of size, adapting Karner's method.

Karner's work on Use Case Point metrics was written as a diploma thesis at the University of Linköping. It was based on just a few small projects, so more research is needed to establish the general usefulness of the method. The work is now copyright of Rational Software, and is hard to obtain. The method is described by Schneider and Winters [SW98], but the authors leave many questions unanswered, for instance why Karner proposed the metrics he did. Users of the method have had to guess what is meant by some of the input factors [ADJS01].

Some work has been published on use case points in conjunction with function points. Certain attempts have been made to combine function points and use case points [Lon01]. A modification of Karner's method to fit

the needs of a specific company is discussed by Arnold and Pedross [AP98], where use case points are converted to function points. An attempt to map the the object oriented approach into function points has been described by Thomas Fetke *et al.* [FAN97], and converting use case point counts to lines of code by John Smith [Smi99]. But there does not seem to be much research done on these ideas.

1.2 Problem Specification and Delimitation

The purpose of this thesis is to present the results of my work on the following problems:

Although the use case points method has shown promising results [ADJS01], there are several unsolved problems. The projects described in a research study by Bente Anda *et al.* were relatively small, and the use cases were were well structured and written at a suitable level of functional detail [ADJS01]. The overhead for all development projects increases with increasing size, and it is not certain that the method would perform as accurately in larger projects. The use case points method was created several years ago, has been little used and has not been adjusted to meet the demands of today's software production. Although the method seems to work well for smaller business applications, it has not been shown that it can be used to size all kinds of software systems. More experience with applying the method to estimating different projects in different companies is needed to be able to draw any final conclusions about the general usefulness of the method.

The method is also dependent on well written, well structured use cases with a suitable level of textual detail if it is to be used effectively, and this is often not the case in the software industry. It is therefore necessary to verify that use cases are written at an appropriate level of detail. if this is not the case, alternative, reliable methods for defining use case complexity must be applied. Use case descriptions vary in detail and style. There are no formal standards for use case writing. The use case descriptions must not be too detailed, but they must include enough detail to make sure that all the system functionality is captured. Sometimes use case descriptions are not detailed enough, or they are completely lacking. It may still be possible to size the system with the use case points method, but other approaches to defining use case size and complexity must be found.

Practitioners of function point methods have discarded the cost drivers that measure technical and quality factors, because they have found that unadjusted counts measure functional size as accurately as adjusted counts. It is possible that the technical adjustment factors can be omitted in the use case points method.

There are a few tools on the market that claim to support estimation methods based on use cases. Some are expensive, and difficult to learn and use. It has not been proved that there are any advantages to such tools. Barbara Kitchenham states that little effort has been directed towards evaluating tools [Kit98]. I have therefore chosen to evaluate two tools, and compare them with the use case points method.

In order to investigate these issues in depth, I have conducted case studies in a major software company. The projects in these case studies were typical in that many of the use cases lacked detailed textual descriptions. I have investigated several alternative ways of defining use case complexity. Applying the use case points method, estimates have been made with and without the technical adjustment factors. I have also studied students' projects in order to define the appropriate level of textual detail necessary for estimation purposes.

Estimates of effort have been computed for all the projects described in the thesis with the use case points method and two commercial cost estimation tools. An estimate is a value that is equally likely to be above or below the actual result. Estimates are therefore often presented as a triple: the most likely value, plus upper and lower bounds on that value [FP97]. The use case points method and the tools calculate estimates of effort as numerical values, and this thesis therefore presents estimates as a single number, not as a triple.

1.3 Contribution

My contribution to estimating effort with use cases is:

- **To investigate the general usefulness of the use case points method by applying the method to two industrial projects, as well as to ten students' projects.**

  I have compared the results to findings that have been made earlier in a different company [ADJS01]. The results indicate that the method can be used to size different kinds of object-oriented software applications.

- **To establish that the Technical Value Adjustment Factor may be discarded as a contributor to software size.**

  By applying the use case points method to several projects, I have uncovered that the technical value adjustment factor may be dropped in the use case points method. I have made estimates with and without the technical adjustment factor, and observed that the estimates do not differ much. Dropping the technical factors means simpler counting rules and more concise measures.

- **To define the appropriate level of detail in use case descriptions, and provide guidelines for use case writing for estimation purposes. I also describe alternative ways of sizing the software with use cases, even when the use cases are not written in full detail.**

  In order to effectively produce estimates with use cases, these must be written out in detail. I have described the appropriate level of detail in textual use case descriptions necessary for estimation purposes. These description may serve as a basis for company guidelines for use case writing. If the use cases are lacking in textual descriptions, other approaches to use case sizing must be applied, and I have described several alternative approaches.

- **To specify the environmental adjustment factors, and provide guidelines for setting values for the factors.**

  In the use case points method, a set of environmental factors are measured and added to functional size in order to predict an estimate of total effort. There are many uncertainties connected with assigning values to these factors. I have defined guidelines for determining the values for each factor in order to obtain more consistent counting rules. This again means more accurate counts and estimates.

- **To select an appropriate cost estimation method or tool for the software company where the case studies were conducted.**

  The software company in question was interested in selecting a cost estimation method or tool suited to their specific needs. To decide which method or tool was the most appropriate, I conducted a feature analysis. I compared accuracy of estimates produced with the use case points method with accuracy of estimates produced by two commercial tools that take use cases as input. I also compared features like usability and learnability. The goal was also to investigate if there are any advantages to using commercial cost estimation tools.

1.4 Thesis Structure

- Chapter 2 presents software cost estimation techniques, traditional cost estimation models, and the function point methods Function Point Analysis (FPA) and MkII FPA. Two commercial cost estimation tools are also described.

- Chapter 3 gives an overview of use cases and use case modeling, and describes the use case points method in detail. Related work that has been done on sizing object-oriented software is described, and the Unified Modeling Language (UML) is presented.

- Chapter 4 describes the research methods that have been used in the case studies and the studies of the students' projects.

- Chapter 5 presents two case studies that were conducted in a major software company. The two projects differ in several ways, although they are similar in size. The first project is a Web application, while the second project is a real-time system. Estimates were produced with the use case points method, an extension of the method dropping the technical adjustment factor, and the tools described in Chapter 2. Alternative approaches to sizing are studied in detail.

- Chapter 6 describes students' projects that were studied in order to try out the method and the tools on several more projects. Although some of the data may be uncertain, the results indicate that the use case points method produces fairly accurate estimates. A use case is analysed to show how to write use cases at a suitable level of detail.

- Chapter 7 analyses the results from the investigations and case studies described in Chapters 5 and 6. The role of the technical factors is discussed, and whether they may be omitted. I also describe how to assign values to the environmental adjustment factors and define guidelines for setting scores. The chapter describes how to define complexity when the use cases are lacking in detail, and how writing and structuring the use cases influence estimates.
- Chapter 8 presents an evaluation of method an tools based on a feature analysis. The goal was to choose the most appropriate method or tool for the software company.
- Chapter 9 presents an extension of the use case points method, where the technical adjustment factor is dropped, and describes how use case complexity may be defined using alternative approaches.
- Chapter 10 presents conclusions and ideas for future work.
- Appendix A gives examples of textual use case descriptions.
- Appendix B describes regression-based cost models, the forerunner of function points methods and the use case points method.
- Appendix C gives an overview of measurement and measurement theory.

Chapter 2

Cost Estimation of Software Projects

Sizing and estimating are two aspects or stages of the estimating procedure [KLD97]. This chapter presents various approaches to sizing and cost estimation. In Section 2.1, system sizing and cost estimation models and methods are described. The Use Case Points method is inspired by traditional Function Point Analysis (FPA), and Mk II Function Point Analysis. These two function point methods are described in Section 2.2. Two commercial cost estimation tools are presented in Section 2.3.

2.1 Software Size and Cost Estimation

Software measurement is the process whereby numbers or symbols are assigned to entities in order to describe the entities in a meaningful way. For software estimation purposes, software size must be measured and translated into a number that represents effort and duration of the project. See Appendix C for more information on software measurement and measurement theory.

Software size can be defined as a set of internal attributes: length, functionality and complexity, and can be measured statically without executing the system. Reuse measures how much of a product was copied or modified, and can also be identified as an aspect of size. Length is the physical size of the product and can be measured for the specification, the design, and the code. Functionality measures the functions seen by the user. Complexity refers to both efficiency and problem complexity [FP97]. Approaches to estimation are expert opinion, analogy and cost models. Each of these techniques can be applied using bottom-up or top-down estimation [FP97].

2.1.1 Bottom-up and Top-down Estimation

Bottom-up estimation begins with the lowest level components, and provides an estimate for each. The bottom-up approach combines low-level estim-

ates into higher-level estimates. Top-down estimation begins with the overall product. Estimates for the component parts are calculated as relative portions of the full estimate [FP97].

2.1.2 Expert Opinion

Expert opinion refers to predictions made by experts based on past experience. In general, the expert opinion approach can result in accurate estimates, however it is entirely dependent on the experience of the expert. Expertise-based techniques are useful in the absence of quantified, empirical data and are based on prior knowledge of experts in the field [Boe81]. The drawbacks to this method are that estimates are only as good as the expert's opinion; they can be biased and may not be analyzable. The advantages are that the method incorporates knowledge of differences between past project experiences [FP97].

2.1.3 Analogy

Analogy is a more formal approach to expert opinion. Estimators compare the proposed project with one or more past projects. Differences and similarities are identified and used to adjust the estimate. The estimator will typically identify the type of application, establish an initial prediction, and then refine the prediction within the original range. The accuracy of the analogy approach is dependent on the availability of historical project information [FP97].

2.1.4 Cost Models

Cost models are algorithms that relate some input measure, usually a measure of product size, to some output measure such as project effort or duration. Cost models provide direct estimates of effort, and come in two main forms; mathematical equations and look-up tables [KLD97].

Mathematical equations use size as the main input variable and effort as the output variable. They often include a number of adjustment factors called cost drivers. Cost drivers influence productivity, and are usually represented as ordinal scale measures that are assigned subjectively, for instance when measuring programmer experience: very good, good, average, poor, very poor.

The advantages of cost models are that the they can be used by non-experts. The disadvantages are that the basic formula must be updated to allow for changes in development methods. Models assume that the future is the same as the past, and give results that apply to 'average' projects [KLD97].

The origin of the different cost models and methods described in this work are the earlier developed regression-based models. See Appendix B.

2.2 Function Point Methods

The use case points method is based on the well-known Function Point Analysis (FPA) developed by Allan Albrecht, and on the less well-known Mk II Function Point Analysis, which is an adaption and improvement of Albrecht's method. In order to understand the use case points method, it is necessary to have some knowledge of the two function points methods on which it is based, FPA and Mk II FPA. The methods are therefore described in some detail, and it will be made clear which features have been adopted by the use case points method. The use case points method is more similar to the Mk II method than to the traditional function points method FPA.

2.2.1 Traditional Function Point Analysis

Function Point Analysis was developed in the late seventies by Allan Albrecht, an employee of IBM, as a method for sizing, estimating and measuring software projects [Sym91].

The Function Point metric measures the size of the problem seen from a user point of view. The basic principle is to focus on the requirements specification, thus making it possible to obtain an early estimate of development cost and effort. It was the first method for sizing software which was independent of the technology used for its development. The method could therefore be used for comparing performance across projects using different technologies, and to estimate effort early in a project's life cycle.

The function point metric is based on five external attributes of software applications:

- Inputs to the application
- Outputs from the application
- Inquiries by users
- Logical files or data files to be updated by the application
- Interfaces to other applications

The five components are weighted for complexity and added to achieve the 'unadjusted function points', UFPs. Albrecht gave no justification for the values used in this weighting system, except that they gave 'good results', and that they were determined by 'debate and trial.' The total of UFPs is then multiplied by a Technical Complexity Factor (TCF) consisting of fourteen technical and quality requirements factors. The TCF is a cost driver contributing to software size [FP97].

The original Function Point method has been modified and refined a number of times. But the method has serious limitations. One of these is that function points were developed for data-processing applications. Their use in real-time and scientific applications is controversial [FP97]. Counts appear to be misleading for software that is high in algorithmic complexity, but sparse in inputs and outputs [Jon]. Also, object oriented software development and function points have not come to terms [SK]. Concepts

such as objects, use cases and Graphical User Interfaces (GUIs) can not be translated into the twenty year old concepts of 'elementary inputs' and 'logical files' [Rul01]. The research of Kitchenham and Känsälä has also shown that the value adjustment factor does not improve estimates, and that an effort prediction based on simple counts of the number of files is only slightly worse than an effort prediction based on total function points. Using simple counts may improve counting consistency as a result of simpler counting rules [KK97].

2.2.2 MKII Function Point Analysis

MKII Function Point Analysis is a variation of Function Point analysis used primarily in the United Kingdom. The method was proposed by Charles Symons to take take better account of internal processing complexity [Sym91]. Symons had observed a number of problems with the original function point counts, amongst them that the choice of the values complex, average and simple was oversimplified. This meant that very complex items were not properly weighted.

MK II function points, like the original function points, are not typically useful for estimating software projects that include many embedded or real-time calculations, projects that have a substantial amount of underlying algorithms, or Internet projects [Rul01].

Instead of the five component types defined in the function point method, MkII sees the system as a collection of **logical transactions**. Each transaction consists of an input, process and output component [Sym91]. A logical transaction type is defined as a unique input/process/output combination triggered by a unique event of interest to the user, for instance

- Create a customer
- Update an account
- Enquire on an order status
- Produce a report

I made the observation that the definition of a logical transactions is very similar to the concept of a use case. See Chapter 3 on use cases. Because of these similarities between the concepts, the two approaches can be combined to produce better estimates under certain circumstances. The Mk II function points method is therefore described in some detail.

The Mk II function point method modified the Function Point Analysis approach by extending the list of Technical Adjustment Factors. These factors were discarded a few years ago because it was decided that they were no longer meaningful in modern software development. For both the function point methods FPA and MkII FPA, the adjustment factors have been discredited as being unrealistic. Therefore, many practitioners have ignored the adjustment and work using the unadjusted function points instead [Rul01].

The list of general application characteristics is presented here in order to show that Karner adapted the factors T15, T16, T17 and T18 for the

use case points method, in addition to the original factors proposed by Albrecht, T1 to T14.

T1 Data Communications
T2 Distributed Functions
T3 Performance
T4 Heavily used configuration
T5 Transaction Rate
T6 On-line Data Entry
T7 Design for End User Efficiency
T8 On-line Update
T9 Complexity Processing
T10 Usable in Other Applications
T11 Installation Ease
T12 Operations Ease
T13 Multiple Sites
T14 Facilitate Change
T15 Requirements of Other Applications
T16 Security, Privacy, Auditability
T17 User Training Needs
T18 Direct Use by Third Parties
T19 Documentation
T20 Client Defined Characteristics

This list may be compared to the list of technical factors in the use case points method, see Table 3.1 on page 22, to verify that many of these factors are indeed the same.

2.3 The Cost Estimation Tools

There are a number of commercial cost estimation tools on the market. A few of them take use cases as input. For the feature analysis described in Chapter 8, I selected the cost estimation tool 'Optimize' and the UML modeling tool 'Enterprise Architect'. 'Enterprise Architect' also has an estimation function, which was the feature that was evaluated.

I first read the documentation that came with the tools. The approach is called *qualitative screening*, where in order to get an impression of a number of tools, evaluations are based on literature describing the tools, rather than actual use of the tools [KLD97]. The documentation for 'Optimize' is found on the web-site of the tool vendor. The documentation for 'Enterprise Architect' is found as help files in the tool.

The tools were used for computing estimates for the projects in Case Studies A and B, and the students' projects. The evaluation of the use case points method and the tools is presented in Chapter 8.

2.3.1 Optimize

The cost and effort estimating tool 'Optimize' applies use cases and classes in the computation of estimates of cost and effort early in the project life cycle, as well as during the whole project process.

According to the documentation, [Fac], the default metrics used by the tool have been extrapolated from real project experience on hundreds of medium to large client-server projects. The tool uses a technique that is object-oriented, based on an incremental development life-cycle. Estimation results can be obtained early in the project and may be continually refined.

An **Object-oriented development project** needs information about the number of subsystems, use cases and classes. A **Component-based project** needs information about the number of components, interfaces and classes, whereas a **web-based project** uses the number of web pages, use cases and scripts to compute an estimate. These elements are called **scope elements.**

The size of the problem is measured by counting and classifying scope elements in a project. At an early stage, a top-down approach is used, as the amount of information is limited. Bottom-up estimating is used later in the project.

Optimize can import from design models in CASE tools. Importing a use case model from for instance Rational Rose will create a list of use cases. The tool will then create an early estimate based solely on the number of use cases. However, this estimate is not very accurate. To produce more reliable estimates, additional information must be used as input.

A productivity metric of person-days effort is assigned for each type of scope element. Time allocated for each scope element is worked out for different development activities such as planning, analysis, design, programming, testing, integration and review. Qualifiers are applied to each scope element. The complexity qualifier defines each task as simple, medium or complex. The tool provides a set of default metric values based on data collected from real projects, but the user can customize her own metric data to produce more accurate estimates.

The use cases alone do not yield enough information to compute a reliable estimate. One must therefore find analysis classes that implement the functionality expressed in the use cases, and use this number and their complexity as input. It is not quite clear what is meant by 'analysis classes'. An analysis model can be a domain model or a Business Concept Model, which is a a high level analysis model showing the business classes for the system. Figure 2.1 shows a Business Concept Model for the Hour Registration System described in Section 3.4.1.

There are five levels of size and five levels of complexity for each element. A use case or class can range from tiny to huge in size, and trivial to complex in complexity.

Assessing the levels for setting qualifiers is subjective. The following guidelines provide rules of thumb for choosing between the levels during the elaboration phase of the project.

- Setting size for use cases is done by applying the textual descriptions, or if they have yet to be written, by considering the amount which would have to be written to comprehensively document the specific business activity. A couple of lines sets size to tiny, a short paragraph sets size to small, a couple of paragraphs sets size to medium, a page sets size to large, and several pages set size to huge.

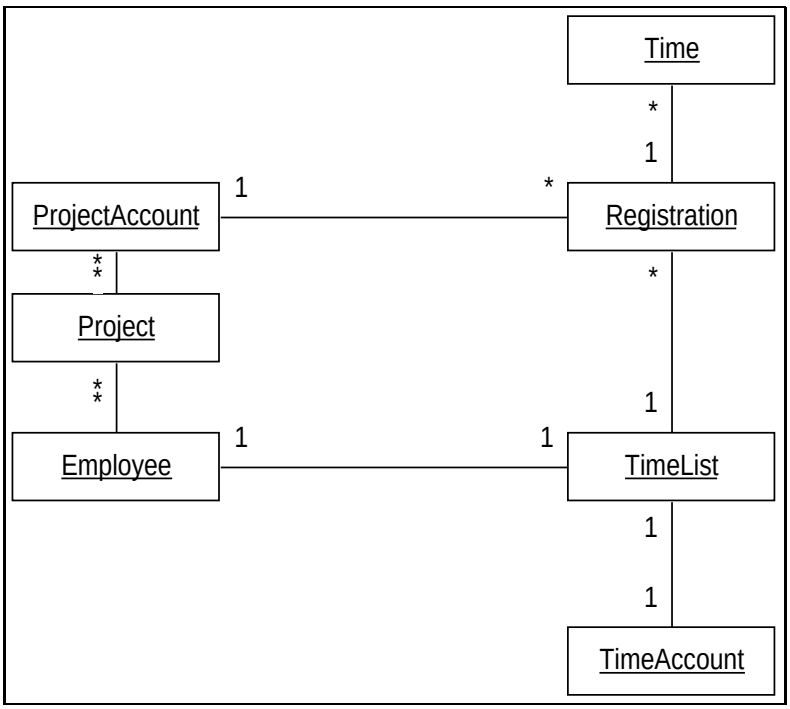

Figure 2.1: A business concept model

- When setting complexity for use cases, the number of decision points or steps in the use case description and the number of exceptions to be handled are used. Basic sequential steps sets complexity to trivial, a single decision point or exception sets complexity to simple, a couple of decision points or exceptions sets complexity to medium, several decision points or exceptions sets complexity to difficult, many decision points and exceptions set complexity to complex.

- Sizing classes is done by considering the amount of business data that is needed to adequately model the business concept: 1 to 3 attributes sets size to tiny, 4 to 6 attributes set size to small, 7 to 9 attributes set size to medium, 10 to 12 attributes set size to large, 13 or more attributes set size to huge.

- When setting the complexity for business classes, algorithms required to process business data are considered.

The scope elements and metric data are organized to compute an estimate of effort and costs. The estimate is given as a total number of staff hours. Duration is also expressed in months. The skills and number of team members are then taken into account and a final estimation for the duration of the project is made.

2.3.2 Enterprise Architect

Enterprise Architect is a CASE tool for creating UML model elements, documenting the elements and generating code. The use case model is imported into an estimating tool. A total estimate of effort is calculated from the complexity level of the use cases, project environment factors and build parameters. The method corresponds exactly to Karner's method. Inputs to the application are the number of use cases and their complexity, the number of actors and their complexity, technical complexity factors (TCF), and environmental complexity factors (ECF). The tool computes unadjusted use case points (UUCP), adjusted use case points (UPC), and the total effort in staff hours.

Use case complexity has to be manually defined for each use case. The following ratings are assigned by defining complexity as described:

- If the use case is considered a simple piece of work, uses a simple user interface and touches only a single database entity, the use case is marked as 'Easy'. Rating: 5.
- If the use case is more difficult, involves more interface design and touches 2 or more database entities, the use case is defined as 'Medium'. Rating 10.
- If the use case is very difficult, involves a complex user interface or processing and touches 3 or more database entities, the use case is 'Complex'. Rating: 15.

The user may assign other complexity ratings, for instance by counting use case steps or implementing classes. The use cases can be assigned to phases, and later estimates will be based on the defined phases.

The technical and environmental complexity factors are calculated from the information entered. The unadjusted use case points (UUCP) is the sum of use case complexity ratings. The UUCP are multiplied together with the TCF and ECF factors to produce a weighted Use Case Points number (UCP). This number is multiplied with the assigned hours per UCP (10 is the application default value), to produce a total estimate of effort. For a given project, effort per use case is shown for each category of use case complexity. For instance, staff hours for a simple use case may be 40 hours, for an average use case 80 hours, and for a complex use case 120 hours. These figures are project-specific, depending on the number of use cases and default staff hours per use case point.

Chapter 3

Use Cases and Use Case Estimation

This chapter describes use cases and how to write them, and presents the Use Case Points method. Section 3.1 gives an overview of the history of use cases, and explains the use case model. Section 3.2 describes the use case points method in detail, Section 3.3 explains how to convert use case point to effort, and Section 3.4 describes how to structure and write use cases. Section 3.5 describes related work on estimating with use cases. The Unified Modling language (UML) is presented in Section 3.6.

3.1 Use Cases

3.1.1 The History of the Use Case

While working at Ericsson in the late 1960s, Ivar Jacobson devised what later became known as use cases. Ericsson at the time modeled the whole system as a set of interconnected blocks, which later became 'subsystems' in UML. The blocks were found by working through previously specified 'traffic cases', later known as use cases [Coc00].

Jacobsen left Ericsson in 1987 and established Objectory AB in Stockholm, where he and his associates developed a process product called 'Objectory', an abbreviation of 'Object Factory'. A diagramming technique was developed for the concept of the use case.

In 1992, Jacobson devised the software methodology OOSE (Object Oriented Software Engineering), a use case driven methodology, one in which use cases are involved at all stages of development. These include analysis, design, validation and testing [JCO92].

In 1993, Gustav Karner developed the Use case Points method for estimating object-oriented software.

In 1994, Alistair Cockburn constructed the 'Actors and Goals conceptual model' while writing use case guides for the IBM Consulting Group. It provided guidance as how to structure and write use cases.

3.1.2 Actors and Goals

The term 'use case' implies 'the ways in which a user uses a system'. It is a collection of possible sequences of interactions between the system under construction and its external actors, related to a particular goal. Actors are people or computer systems, and the system is a single entity, which interacts with the actors [Coc00] .

The purpose of a use case is to meet the immediate goal of an actor, such as placing an order. To reach a goal, some action must be performed [Ric01]. All actors have a set of responsibilities. An action connects one actor's goal with another's responsibility [Coc97].

A *primary actor* is an actor that needs the assistance of the system to achieve a goal. A *secondary actor* supplies the system with assistance to achieve that goal. When the primary actor triggers an action, calling up the responsibilities of the other actor, the goal is reached if the secondary actor delivers [Coc97].

3.1.3 The Graphical Use Case Model

The use case model is a set of use cases representing the total functionality of the system. A complete model also specifies the external entities such as human users and other systems that use those functions. UML provides two graphical notations for defining a system functional model:

- The **use case diagram** depicts a static view of the system functions and their static relationships with external entities and with each other. Stick figures represent the actors, and ellipses represent the use cases. See figure3.1.
- The **activity diagram** imparts a dynamic view of those functions.

The use case model depicted in Figure 3.1 is the model of an hour registration system. The user enters user name and password, is presented with a calendar and selects time periods, and then selects the projects on which to register hours worked. The example is taken from a students' project on a course in object modeling at the University of Oslo, Department of Informatics. The textual use case description of this use case is shown in Section 3.4.1 on page 23, 'The Textual Use Case Description'.

3.1.4 Scenarios and Relationships

A *scenario* is a use case instance, a specific sequence of actions that illustrates behaviours. A main success scenario describes what happens in the most common case when nothing goes wrong. It is broken into *use case steps*, and these are written in natural language or depicted in a state or an activity diagram [CD00].

Different scenarios may occur, and the use case collects together those different scenarios [Coc00].

Use cases can include relationships between themselves. Since use cases represent system functions, these relationships indicate corresponding relationships between those system functions. A use case may either *always*

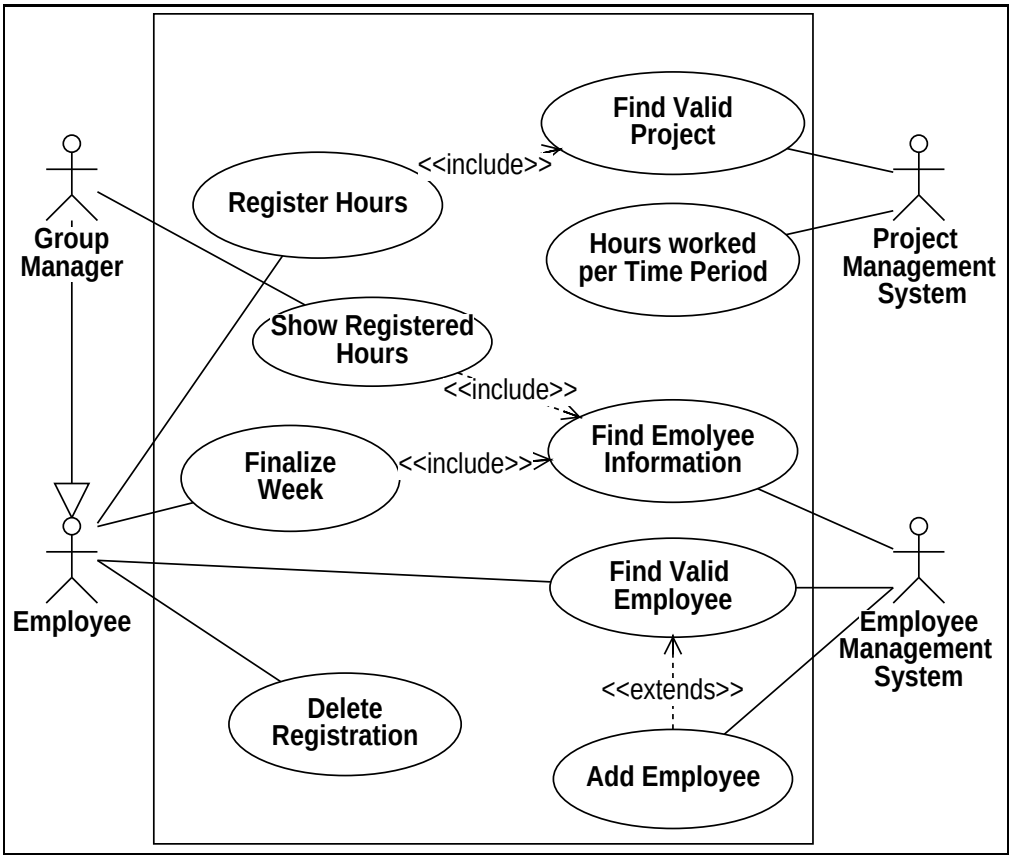

Figure 3.1: A graphical use case model

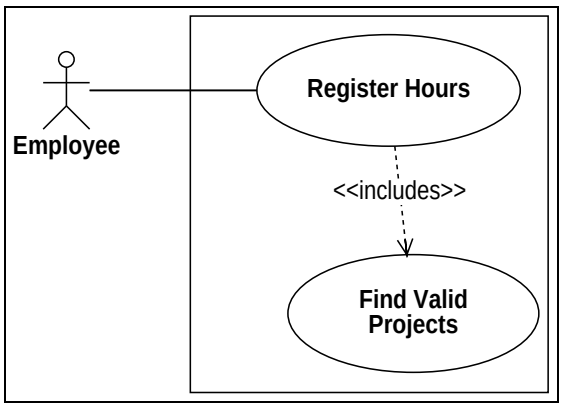

Figure 3.2: The include relationship

or *sometimes* include the behaviour of another use case; it may use either an 'include' or an 'extend' relationship. Common behaviour is factored out in included use cases. Optional sequences of events are separated out in extending use cases.

An *include* relation from use case A to use case B indicates that an instance of the use case A will also include the behaviour as specified by use case B [Ric01]. The include relationship is used instead of copying the same text in alternative flows for several use cases. It is another way of capturing alternative scenarios [Fow97]. When an instance of the including use case reaches a step where the included use case is named, it invokes that use case before it proceeds to the next step. See figure 3.2. The use case 'Find Valid Project' is included in the use case 'Register Hours'.

An *extend* relation from use case A to use case B indicates that an instance of the use case A may or may not include the behaviour as specified by use case B.

Extensions describe alternatives or additions to the main success scenario, and are written separately. They contain the step number in the main success scenario at which the extension applies, a condition that must be tested before that step, and a numbered sequence of steps that constitute the extension [CD00]. See the use case template in Section 3.4.1. Extension use cases can contain much of the most interesting functionality in the software [Coc00]. The use case 'Add Employee' is an extension of the use case 'Find Valid Employee', since this is a 'side-effect' of not finding a valid employee. See Figure 3.3.

3.1.5 Generalisation between Actors

A clerk may be a specialisation of an employee, and an employee may be a generalisation of a clerk and a group manager, see Figure 3.4 on the next page. Generalisations are used to collect together common behaviour of actors.

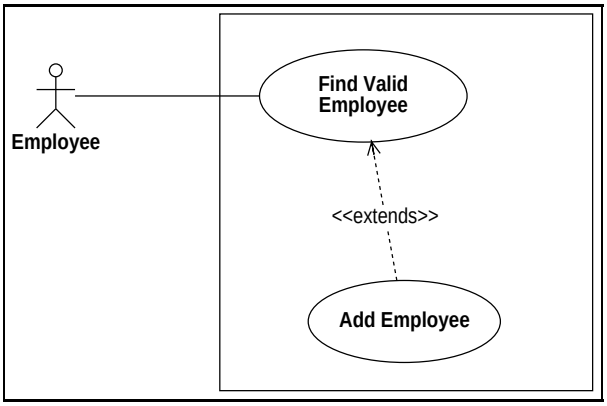

Figure 3.3: The extend relationship

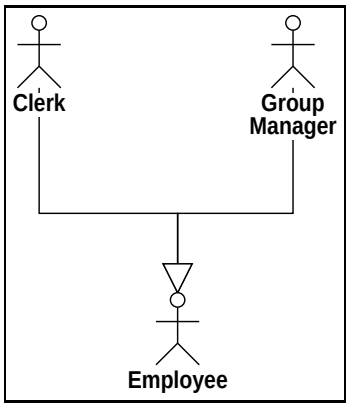

Figure 3.4: Generalisation between actors

3.2 The Use Case Points Method

An early estimate of effort based on use cases can be made when there is some understanding of the problem domain, system size and architecture at the stage at which the estimate is made [SK]. The use case points method is a software sizing and estimation method based on use case counts called use case points.

3.2.1 Classifying Actors and Use Cases

Use case points can be counted from the use case analysis of the system. The first step is to classify the actors as simple, average or complex. A simple actor represents another system with a defined Application Programming Interface, API, an average actor is another system interacting through a protocol such as TCP/IP, and a complex actor may be a person interacting through a GUI or a Web page. A weighting factor is assigned to each actor type.

- Actor type: Simple, weighting factor 1
- Actor type: Average, weighting factor 2
- Actor type: Complex, weighting factor 3

The total **unadjusted actor weights (UAW)** is calculated by counting how many actors there are of each kind (by degree of complexity), multiplying each total by its weighting factor, and adding up the products.

Each use case is then defined as simple, average or complex, depending on number of transactions in the use case description, including secondary scenarios. A transaction is a set of activities, which is either performed entirely, or not at all. Counting number of transactions can be done by counting the use case steps. The use case example in 3.4.1 on page 23 contains 6 steps in the main success scenario, and one extension step. Karner proposed not counting included and extending use cases, but why he did is not clear. Use case complexity is then defined and weighted in the following manner:

- Simple: 3 or fewer transactions, weighting factor 5
- Average: 4 to 7 transactions, weighting factor 10
- Complex: More than 7 transactions, weighting factor 15

Another mechanism for measuring use case complexity is counting analysis classes, which can be used in place of transactions once it has been determined which classes implement a specific use case [SW98]. A simple use case is implemented by 5 or fewer classes, an average use case by 5 to 10 classes, and a complex use case by more than ten classes. The weights are as before.

Each type of use case is then multiplied by the weighting factor, and the products are added up to get the **unadjusted use case weights (UUCW).**

The UAW is added to the UUCW to get the **unadjusted use case points (UUPC):**

UAW+UUCW=UUCP

3.2.2 Technical and Environmental Factors

The method also employs a technical factors multiplier corresponding to the Technical Complexity Adjustment factor of the FPA method, and an environmental factors multiplier in order to quantify non-functional requirements such as ease of use and programmer motivation.

Various factors influencing productivity are associated with weights, and values are assigned to each factor, depending on the degree of influence. 0 means no influence, 3 is average, and 5 means strong influence throughout. See Table 3.1 and Table 3.2.

The adjustment factors are multiplied by the unadjusted use case points to produce the **adjusted use case points**, yielding an estimate of the size of the software.

The *Technical Complexity Factor (TCF)* is calculated by multiplying the value of each factor (T1- T13) by its weight and then adding all these numbers to get the sum called the *TFactor.* The following formula is applied:

TCF=0.6+(0.01*TFactor)

The *Environmental Factor (EF)* is calculated by multiplying the value of each factor (F1-F8) by its weight and adding the products to get the sum called the *EFactor.* The following formula is applied:

EF= 1.4+(-0.03*EFactor)

The *adjusted use case points (UPC)* are calculated as follows:

UPC= UUCP*TCF*EF

3.2.3 Problems With Use Case Counts

There is no published theory for how to write or structure use cases. Many variations of use case style can make it difficult to measure the complexity of a use case [Smi99]. Free textual descriptions may lead to ambiguous specifications [AP98]. Since there is a large number of interpretations of the use case concept, Symons concluded that one way to solve this problem was to view the MkII logical transaction as a specific case of a use case, and that using this approach leads to requirements which are measurable and have a higher chance of unique interpretation [Sym01]. This approach will be described in Chapter 7.

Factor	Description	Weight
T1	Distributed System	2
T2	Response adjectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Reusable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Security features	1
T12	Access for third parties	1
T13	Special training required	1

Table 3.1: Technical Complexity Factors

Factor	Description	Weight
F1	Familiar with RUP	1.5
F2	Application experience	0.5
F3	Object-oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	2

Table 3.2: Environmental Factors

3.3 Producing Estimates Based on Use Case Points

Karner proposed a factor of 20 staff hours per use case point for a project estimate. Field experience has shown that effort can range from 15 to 30 hours per use case point, therefore converting use case points points directly to hours may be an uncertain measure. Steve Sparks therefore suggests it should be avoided [SK].

Schneider and Winters suggest a refinement of Karner's proposition based on experience level of staff and stability of the project [SW98]. The number of environmental factors in F1 through F6 that are above 3 are counted and added to the number of factors in F7 through F8 that are below 3. If the total is 2 or less, they propose 20 staff hours per UCP; if the total is 3 or 4, the value is 28 staff hours per UCP. When the total exceeds 4, it is recommended that changes should be made to the project so that the value can be adjusted. Another possibility is to increase the number of staff hours to 36 per use case point. The reason for this approach is that the environmental factors measure the experience level of the staff and the stability of the project. Negative numbers mean extra effort spent on training team members or problems due to instability. However, using this method of calculation means that even small adjustments of an environmental factor, for instance by half a point, can make a great difference to the estimate. In an example from the project in Case Study A, adjusting the environmental factor for object-oriented experience from a rating of 3 to 2.5 increased the estimate by 4580 hours, from 10831 to 15411 hours, or 42.3 percent. This means that if the values for the environmental factors are not set correctly, there may be disastrous results. The COSMIC approach which is described in Section 3.5.4 does not take the technical and quality adjustment factors into the effect on size. Therefore, many practitioners have sought to convert use case measures to function points, because there is extended experience with the function point metrics and conversion to effort [Lon01].

3.4 Writing Use Cases

The use cases of the system under construction must be written at a suitable level of detail. It must be possible to count the transactions in the use case descriptions in order to define use case complexity. The level of detail in the use case descriptions and the structure of the use case have an impact on the precision of estimates based on use cases. The use case model may also contain a varying number of actors and use cases, and these numbers will again affect the estimates [ADJS01].

3.4.1 The Textual Use Case Description

The details of the use case must be captured in textual use case descriptions written in natural language, or in state or activity diagrams. A use case description should at at least contain an identifying name and/or number, the name of the initiating actor, a short description of the goal of the use

case, and a single numbered sequence of steps that describe the main success scenario [CD00].

The main success scenario describes what happens in the most common case when nothing goes wrong. The steps are performed strictly sequentially in the given order. Each step is an extension point from where alternative behaviour may start if it is described in an extension. The use case model in Figure 3.1 is written out as follows:

Use Case Descriptions for Hour Registration System

```
Use case No. 1
Name: Register Hours
Initiating Actor: Employee
Secondary Actors: Project Management System
                  Employee Management System
Goal: Register hours worked for each employee
      on all projects the employee participates on
Pre-condition: None

MAIN SUCCESS SCENARIO

1. The Systen displays calendar (Default: Current Week)
2. The Employee chooses time period
3. Include Use Case 'Find Valid Projects'
4. Employee selects project
5. Employee registers hours spent on project
   Repeat from 4 until done
6. The System updates time account

EXTENSIONS

2a. Invalid time period
       The System sends an error message and prompts
       user to try again
```

This use case consists of 6 use case steps, and one extension step, 2a. Step 2 acts as an extension point. If the selected time period is invalid, for instance if the beginning of the period is after the end of the period, the system sends an error message, and the user is prompted to enter a different period. If the correct time period is entered, the use case proceeds. The use case also includes another use case, 'Find Valid Projects'. This use case is invoked in step 3. When a valid project is found by the Project Management System, it is returned and the use case proceeds. The use case goes into a loop in step 5, and the employee may register hours worked for all projects he/she has worked on during the time period.

The use case 'Find Valid Employee' is extended by the use case 'Add Employee'.

```
------------------------------------------------------------
Use case No. 2
Name: Find Valid Employee
Initiating Actor: Employee
Secondary Actor:  Employee Management System

Goal: Check if Employee ID exists
Pre-condition: None

MAIN SUCCESS SCENARIO

1. Employee enters user name and password
2. Employee Mangement System verifies user name and
   password
3. Employee Mangement System returns Employee ID

EXTENSIONS
    2a. Error message is returned
    2b. Use Case 'Add Employee'

------------------------------------------------------------
```

Extensions handle exceptions and alternative behaviour and can be can be described either by extending use cases, as in this case, or as alternative flows, as in the use case 'Register Hours'. In this use case, the system verifies if the user name and password are correct in step 2. If there are no such entries, the user name and password may be incorrect, in which case an error message is returned. Another option is that no such employee exists, in which case the new employee may be registered in the Employee Management System, involving the use case 'Add Employee'. Invoking this use case is therefore a side effect of the use case 'Find Valid Employee.'

3.4.2 Structuring the Use Cases

The Unified Modeling Language, (UML), does not go into details about how the use case model should be structured nor how each use case should be documented. Still, a minimum level of detail must be agreed on, as it is necessary to establish that all the functionality of the system is captured in the given set of use cases. Ivar Jacobsen provided the following advice for use case creation:

'If we try to describe a use case that contains a great many courses of events, our text can easily become difficult to understand. Therefore, it is wise to use some form of structured writing approach' [JEJ95].

'Scenario explosion' must be avoided [Ber97]. Writing several pages descriptions makes it difficult to control what is actually happening. Scenario explosion is avoided using the three techniques: included use cases, extensions and variations.

The behaviour of **included use cases** is always invoked when the included use case is named in a use case step. See Section 3.2.4. Alternative behaviour and failure in a step are handled by an **extension scenario**. Many

structuring techniques may be used to combine main and extension scenarios. Some people use if statements and alternatives within a scenario, others write the entire alternate scenario from the beginning, so that each scenario can be read independently. Others again write scenario fragments as extensions to other scenarios, to save writing and reading [Coc97]. Functionality can also be separated out in alternative flows instead of in extending use cases, but this is a choice of design. Writing extending and included use cases is more effective than writing alternative flows [Coc00] [CD00]. It is also in accordance with the principles for good object-oriented analysis and design [Ric01].

Variations are a section of the use case text describing different alternatives of action. Often, a use case at a high level uses input or output of one of several types. An example is payment by cash, check, credit, credit card, or electronic funds transfer. The differences between all those possibilities will have to be described at some point in a lower-level, but it would be wasteful to spend the time doing so at the higher levels [Coc97].

The projects described in this thesis demonstrate various approaches to use case writing. I have studied approximately one hundred and fifty different use case descriptions, and observed that there was no 'scenario explosion' to be found anywhere. I therefore believe that people are naturally inclined to limit the size of the use cases, and prefer writing many smaller use cases instead of several large ones with many alternative scenarios. According to Alistair Cockburn, most well-written use cases have 3 to 8 steps, probably because people do not think in terms of processes that take more than 10 intermediate steps [Coc00]. This observation is supported by the research of George A. Miller, who states that there seems to be some limitations built into the human brain, either by learning or by the design of the nervous system. There is a span of absolute judgment that can distinguish about seven plus minus two different categories, and a finite span of immediate memory which is about seven plus minus two items in length. This imposes severe limitations on the amount of information that human beings are able to process and remember [Mil56]. These issues will be discussed in further detail in Section 7.5.

3.4.3 Counting Extending and Included Use Cases

Although Karner recommended that included and extending use cases should not be counted, the functionality described in these use cases must still be implemented [ADJS01]. If these use cases contain much of the essential functionality, it is necessary to include them in the counts. However, there is a danger that when writing included and extending use cases one may be more concerned with identifying opportunities for reuse, than with analyzing the problem and describing requirements [Rul01]. Bente Anda *et al.* describe how omitting extending and included use cases resulted in an estimate that was closer to the actual effort, after first having counted these use cases and obtained an estimate that was much higher than the actual effort. In a different project, the extending and included use cases were counted, because much of the essential functionality was described in these use cases [ADJS01]. According to Alistair Cockburn, this is often the

case [Coc00].

It seems difficult to form a precise rule for when to count extending and included use cases. But the findings of Bente Anda *et al.* make it clear that one should not always follow Karner's recommendations. In some projects, extending and included use cases contain a lot of the important functionality. If in doubt, I believe that it is better to count extending and included use cases to be sure that all the functionality is sized, in order to avoid underestimation.

3.5 Related Work

Different methods for sizing object-oriented software projects and computing estimates of effort have been proposed over the last years. Some of these methods are presented in the following.

3.5.1 Mapping Use Cases into Function Point Analysis

A method for mapping the object-oriented approach into Function point analysis is described by Thomas Fetke *et al.*, [FAN97]. The authors propose mapping the use cases directly into the Function point model using a set of concise rules that support the measurement process. These mapping rules are based on the standard FPA defined in the IFPUG Counting Practices manual.

Since the concept of actors in the use case model is broader than the concept of users and external applications in FPA, there cannot be a one-to-one mapping of actors and users to external applications. But each user of the system is defined as an actor. In the same manner, all applications which communicate with the system under consideration must also appear as actors. This corresponds to Karner's use case point method.

The level of detail in the use case model may vary, and the use case model does not provide enough information to how to count a specific use case according to function point rules. Therefore, as in Karner's method, the use cases must be described in further detail in order to be able to count transactions.

The authors conclude that Function Point Analysis can be used in the object-oriented paradigm, as the findings support the thesis that the function point method is technology independent. However, the method does not seem to have been used in software development projects apart from the projects described in the article. In referring to this work, John Smith states that the level of the use case must be described appropriately for the mapping to be valid, and asks if drawing parallels between function points and use case points may be misguided [Smi99].

3.5.2 Use Case Estimation and Lines of Code

John Smith of Rational Software describes a method presenting a framework for estimation based on use cases translated into lines of code [Smi99].

There does not seem to be any more research done on this method, although the tool 'Estimate Professional', which is supplied by the Software Productivity Center Inc, and the tool 'CostXpert' from Marotz Inc. produce estimates of effort per use case calculated from the number of lines of code.

3.5.3 Use Cases and Function Points

David Longstreet of Software Metrics observed that applying function points helps to determine if the use case is written at a suitable level of detail [Lon01]. If it is possible to describe how data passes from the actor to inside the boundary or how data flows from inside the application boundary to the actor, then that is the right level of detail, otherwise the use case needs more detail. By adopting both the use case method and the function points method, the quality of the requirement documents can be improved. Thus, sizing and estimating is improved.

3.5.4 The COSMIC-FFP Approach

Over the last 15 years or so, advances have been made towards a general Functional Size Measurement (FSM) method for measuring real-time software. Recently, the COSMIC FFP (Full Function Points) method has been developed as an improvement of the earlier function point methods. It is designed to work for both business applications and real-time software [Rul01].

When sizing software using the traditional function point methods, it is possible to measure only the functionality as seen by the human end-user. The large amounts of functionality that must be developed in today's advanced software systems are invisible to the users and cannot be measured by these methods. Using the traditional methods may correctly size the functionality seen by the user, but grossly undersize the total functionality that actually has to be developed.

The Full Function Points (FFP) methodology is a functional size measurement technique specifically designed to address the requirements of embedded and real-time software. The FFP methodology is based on a 'unit of software delivered' metric called the FFP point, which is a measure of the functional size of the software. The total FFP points of an application being measured is called an FFP count.

Functional user requirements are decomposed into 'functional processes' which in turn can be decomposed into 'functional sub-processes'. The functional processes are equivalent to the MKII logical functions and also to use cases. The method can therefore be used to size object-oriented software. The method does not take into account the effect on size of technical or quality requirements. These are measured separately [Ser01].

3.5.5 Experience from Industry 1

At the end of 1993, Martin Arnold and Peter Pedross developed a use case point method inspired by Karner for a major Swiss Banking Institute. The

purpose was to measure the size of large-scale software systems based on requirements specifications. Although they used the same name as Karner did, the concept was new. In this method, use case points are compared directly to function points, and the function points method is used to calibrate the use case points method.

The quality of requirements documents and the measured use case points were analyzed in order to test and calibrate the use case point method. The analysis showed that requirements specifications with use cases and scenarios can be used to measure the size of a software system, and that measurement can normally be done in a couple of hours [AP98].

3.5.6 Experience from Industry 2

The work of Bente Anda *et al.* [ADJS01] shows that the use case points method computes fairly accurate estimates for projects from a specific company where the method was tried on historical project data. The method was used on three projects similar in size and functionality. In order to establish the general usefulness of the method, the authors state that it must be tried out on different projects in different companies. I have used data from the three projects in question to investigate certain aspects of the method. The results are presented in Chapter 7.

3.6 The Unified Modeling Language

The Unified Modeling Language is used for analysis and design of object-oriented software. The language is based on earlier modeling languages like the *Object modeling Technique (OMT)*, and *Object-Oriented Software Engineering (OOSE)* [JCO92]. The UML has been used to model all the projects studied in this thesis, but with a varying degree of detail. I have used the various UML diagrams in the project documentation for data collection. I will therefore give a brief description of UML and the different diagrams.

In Object-oriented analysis and design (OOAD), use cases describe the functional requirements. The use case diagram has been incorporated into the UML standard and is therefore part of the UML notation. The UML standard does not discuss the content or writing of a use case. One may therefore be led to think that use cases are a graphical, not a textual, construction [Coc00].

Using UML, the developer can construct the following views of a system through the use of diagrams:

- The *functional view* describes the basic functional requirements of the system. **Use case diagrams** depict a static functional view and their static relationships. **Activity diagrams** depict a dynamic functional view, and are used to show activities, work flows, conditional processing and how work in general is done in the organisation.

- The *static structural view* defines the static structure of the system and is represented by **class diagrams** and **object diagrams**.

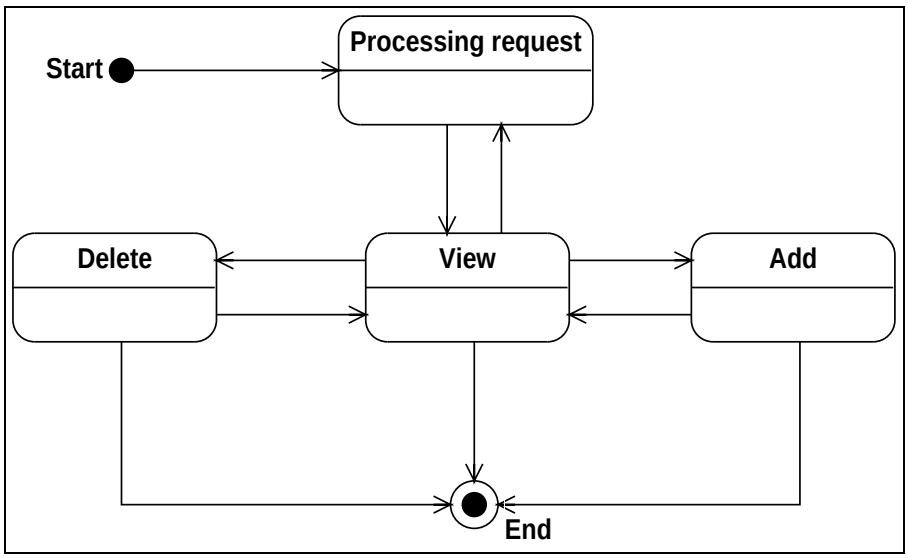

Figure 3.5: A state diagram

- The *behavioural view* describes the temporal behaviour of the system using **interaction diagrams** like **collaboration diagrams** and **sequence diagrams** to depict sequences of interactions between objects in different scenarios.

The UML specifications states:

A **use case** is a kind of classifier representing a coherent unit of functionality provided by a system, a subsystem, or a class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors) together with actions performed by the system. The behavior of a use case can be described in several different ways, depending on what is convenient: often plain text is used, but state machines, operations, and methods are examples of other ways of describing the behavior of the use case. A use case diagram is shown in Figure 3.1 on page 17.

A **state diagram** is used to show how an element, usually a class, changes state over time, and the allowable transitions and conditions for transition. **Statechart diagrams** represent the behavior of entities capable of dynamic behavior by specifying its response to the receipt of event instances. Typically, this diagram is used for describing the behavior of classes, but a statechart may also describe the behavior of other model entities such as use-cases, actors, subsystems, operations, or methods. A state diagram is shown in Figure 3.5.

An **activity diagram** is a special case of a state diagram in which all of the states are action or sub activity states and in which all of the transitions are triggered by completion of the actions or sub activities in the source states. An activity diagram is shown in Figure 3.6.

A **class diagram** is a graph of classifier elements connected by their vari-

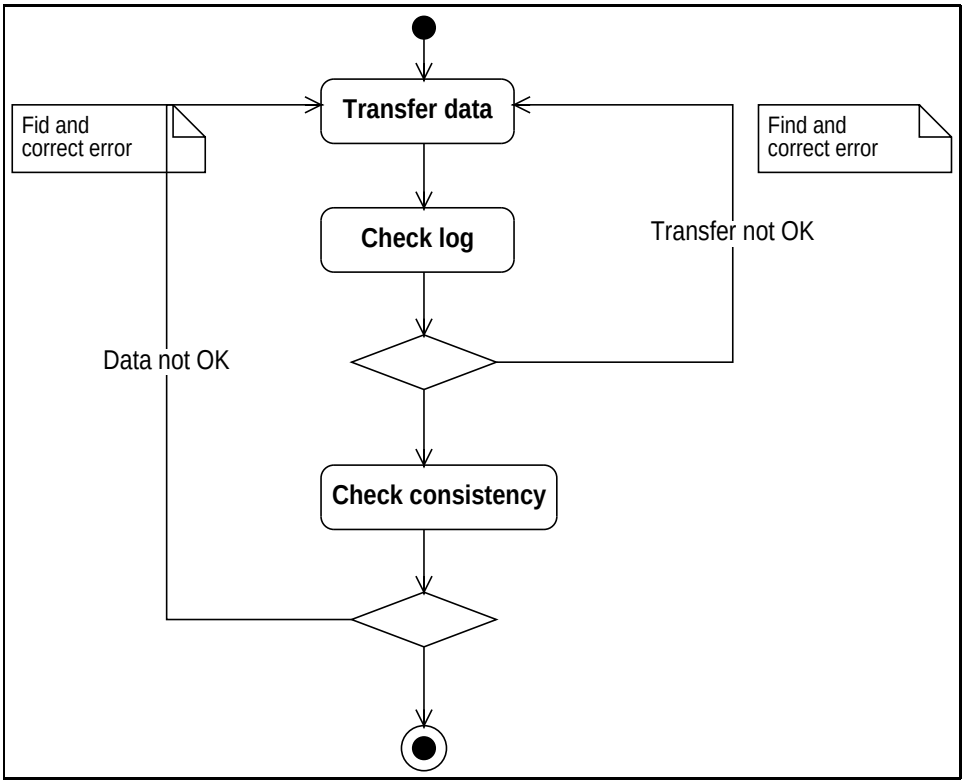

Figure 3.6: An activity diagram

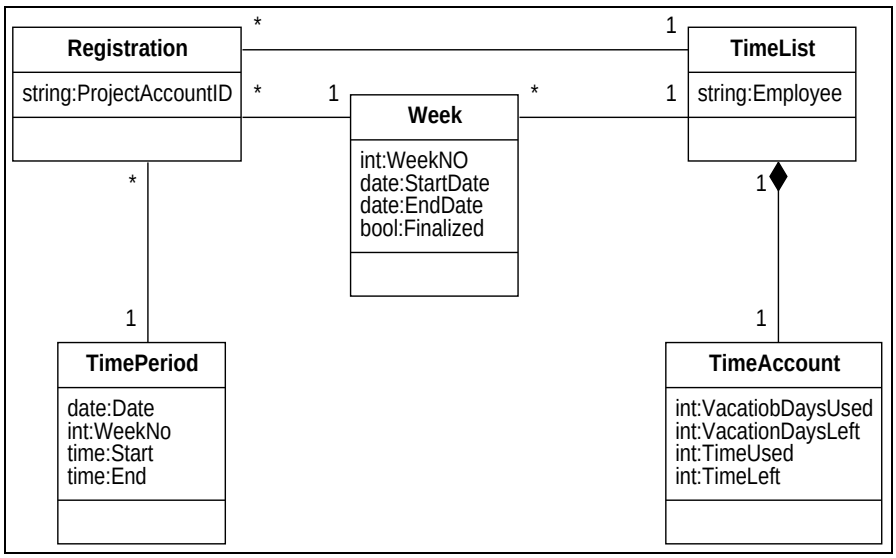

Figure 3.7: A class diagram

ous static relationships. Note that a diagram may also contain interfaces, packages, relationships, and even instances, such as objects and links. A simple class diagram for the hour registration system described by the use case model in Figure is 3.1 on page 17 is shown in Figure 3.7.

An **interaction** defined in the context of a collaboration specifies the details of the communications that should take place in accomplishing a particular task. A communication is specified with a message, which defines the roles of the sender and the receiver instances, as well as the action that will cause the communication. The order of the communications is also specified by the Interaction.

A **sequence diagram** presents an interaction, which is a set of Messages between classifier roles within a collaboration to effect a desired operation or result [Spe01]. A simple sequence diagram is shown in Figure 3.8.

For the work done on this thesis, use case models, their textual descriptions, activity diagrams, class diagrams and sequence diagrams have been used.

3.6.1 Using Sequence Diagrams to Assign Complexity

When there are no detailed use cases, use case complexity can be assigned by counting transactions from the sequence diagrams. Sequence diagrams belong to the *behavioural view* according to the UML standard. They are at a lower level of detail than the use cases, and one must therefore be careful not to count too many transactions. Only the actual behaviour: 'what' to do, not' how' to do it, must be counted.

In order to find out how to count transactions in the sequence diagrams without counting too many, subsystem number 3 in Case study B was studied in detail. See Section 5.2. This was the only system that had detailed

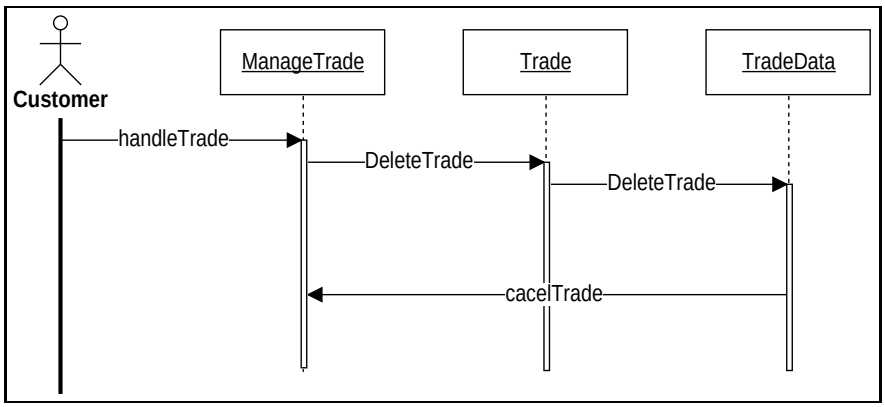

Figure 3.8: A sequence diagram

use case descriptions as well as sequence diagrams. In order to verify that use case transaction steps were written correctly, I compared the number of use case steps to the number of transactions found in the sequence diagrams. By omitting transactions in the sequence diagrams that were at too low a level of functional detail, describing 'how to do' things, not 'what to do' , as well as omitting the 'logging on' procedures that were the same in all the sequence diagrams for this subsystem, I arrived at a number that was comparable to the number of use case steps. In this manner, I also verified that the way I counted transactions from the sequence diagrams was reliable.

Chapter 4

Research Methods

In this chapter, the methods used in the work on the thesis are described. These methods include preliminary literature studies, case studies, interviews and feature analysis of methods and tools. Section 4.1 describes preliminary work. Section 4.2 describes the interviews I conducted with the team members of the projects of the case studies, and on the student's projects. Section 4.2 describes the case studies, and Section 4.3 explains the use of feature analysis.

4.1 Preliminary work

- The literature studies have covered measurement theory, process improvement, estimation techniques, function point analysis, use case point counts.
- Learning to use the methods and tools has been part of the preliminary studies. The use case points method is easy to learn, and the counts are entered into a spreadsheet which then computes the estimates. I selected two cost estimation tools to study. The tools have been used to compute estimates of the projects in the case studies, and the students' projects.

4.2 Case Studies

Case studies are a standard method of empirical studies in various 'soft' sciences such as sociology, medicine and psychology. Case studies usually look at a typical project, and are used to evaluate different methods and tools and select the one that best fits the needs of a specific company [KLD97].

I have conducted two case studies in a major software company in Norway. I started this work in April 2001, and finished in November 2001. The first decision when undertaking a case study is to determine what you want to know or investigate. [KP98]. The software company was interested in obtaining a cost estimation method or tool. In a case study, ideally only one

method or tool should be used per project, in order to study the effects of using the specific method or tool [KJ97]. But since I needed to investigate many aspects of the method and tools, I used the method and the tools on all the projects, including the students' projects.

I conducted a feature analysis with case studies to decide which method or tool would be the most appropriate for this specific company. I also compared the estimates obtained with the use case points method on the projects in the case studies with the results of the research described by Bente Anda *et al.* [ADJS01], in order to investigate the general usefulness of the method, and the effect of discarding the technical value adjustment factor as a contributor to software size. During this process, I uncovered certain weaknesses inherent in the method. I have therefore proposed an extension of the method which provides simpler counting rules. This extension is presented in Chapter 9.

4.2.1 Feature Analysis

In order to select an appropriate method or tool for the software company, *qualitative case studies* were conducted.

A qualitative case study is a feature-based evaluation of methods and tools performed by staff who have used the method or tool on a real project [KLD97]. Qualitative evaluation aims at establishing method/tool appropriateness: how well a method or tool fits the needs of an organization [KLD97]. This type of analysis is referred to as a *feature analysis*, and the evaluation as a *qualitative* or *subjective* evaluation.

In a feature analysis, there are four major roles. Some of them may be performed by the same person [KJ97]:

- The sponsor is the person who wants to know what purchasing decision to make, given a requirement for investment in some technology
- The evaluator(s) is the person or team responsible for running the evaluation exercise
- The technology user(s) are the people who will use the method/tool chosen
- The method/tool assessors are the people who will score each feature for the candidate methods and tools. The method/tool assessor can be the evaluator or current users of the method or tool

The responsibility of the evaluator is to select the method and tools to be investigated, decide which features to investigate, organise the assessments whereby each of the methods or tools is scored, analyze the scores and prepare the evaluation report. The responsibility of the assessor is to assign scores to the different features. The responsibility of the sponsor is to provide the funds, and receive and act on the evaluation report. Defining how evaluations are to be carried out include how the candidate method/tools will be identified, how the feature-sets and criteria of rejection or acceptability will be derived, how the results will be arrived at and presented, and how the evaluations are to be organised [KJ97].

After a a set of methods and tools have been assessed, the scores are analyzed and it is determined which method or tool is best suited to the needs of the organisation.

4.3 Interviews

Interviews can be formal, semi-formal or informal. Informal interviews are like ordinary conversations. I used interviews both in the case studies and when studying the students' projects.

4.3.1 The Case Studies

Semi-formal interviews were conducted with the project manager and the supervisor in Case Study A, and the mentor and the project manager in Case Study B. The reason for conducting the interviews was to gather information and data that could not be found in the project documents or the use case models. Some of the interviews were ordinary conversations. The interviews were conducted face to face and by telephone. The results of the interviews are presented in Sections 5.1.2 and 5.2.2, 'Data Collection'.

After a brief explanation of the use case points method, I asked general questions about the projects, the choice of technology and the teams, and specific questions about the actors and the technical and environmental factors.

The general questions were the following:

- What is your role on the team?
- What sort of system is this?
- How many members are there on the development team?
- What sort of programming experience do the team members have?
- Are the team members familiar with object-oriented analysis and design?
- What was the starting date of the project?
- When is the deadline for the project?
- How was the project estimated?

The specific questions were asked in order to achieve the values for the technical and environmental factors, using the Tables 3.1 and 3.2. The values were defined on a scale from 0 to 5. Some of the factors were unclear, and had to be specified through discussions about the meanings of each factor.

In Case Study A, the interview with the supervisor was conducted as an informal conversation. The interview lasted for half an hour. I asked her about the number of use cases and their size and complexity. She counted the use cases from the use case model as I watched her. While we were talking, I took notes of the conversation.

I conducted two interviews with the project manager. They both lasted one hour. In the course of the first interview, the values for the technical and environmental factors were defined. The second interview established which features should be investigated in the feature analysis described in Chapter 8.

Setting scores for the technical and environmental factors was an iterative process. The project manager first assigned values to the specific factors. Later, he asked the advice of two leading developers, who refined the values. The project manager then decided that the environmental factors had been set too high, and made some adjustments.

In Case Study B, the interview with the mentor was both a semi-structured interview and an informal conversation, and lasted for one and a half hours. I had a short conversation with the project manager after interviewing the mentor, and asked how estimation had been done. One interview was conducted on the telephone with the project manager. I asked him about the estimation process, and about the estimates for each subsystem, also how many members there were on each team.

The results of these interviews, and the data collected for input to the method and tools are presented in Chapter 6, Sections 6.1 (Case study A) and 6.2, (Case study B).

4.3.2 The Students' Projects

Students' projects from a course in Software Engineering at the University of Oslo have also been studied in order to try out the use case points method on several more projects. These projects were small and not very complex, and the data were somewhat unreliable. Still, the results are useful for studying use case writing, and as a basis for defining guidelines for writing use cases.

I selected 10 projects to study. Most of the project data was collected from the project documents. However, I needed the values for the technical and environmental factors, and conducted semi-structured interviews with one student from each project group. Each interview lasted about 20 minutes.

The questions were the following:

- What sort of system is this?
- What is your role on the team?
- How many members are there on the development team?
- What sort of programming experience do the team members have?
- How did you organize work on the different tasks?
- How many team members did the programming work?
- How did you estimate effort?
- How did you register actual effort?

- Did you deliver all the modeled functionality?

The Tables 3.1 and 3.2 were used for setting scores for the technical and environmental factors. The students assigned the values. There was much insecurity here. Setting these factors requires some experience from past projects [SW98]. The final value were obtained through discussions between the students and the author.

The results of the interviews are presented in Section 6.1.2, 'Data Collection'.

4.4 Analysis

In the feature analysis, I used qualitative evaluation of method and tools order to select the most appropriate method or tool for use in the specific software company. I have applied the case study approach, using the method and the tools to compute estimates of both projects in Case Studies A and B. These projects were typical of the projects undertaken by the software company.

All the estimates for a project were compared to the expert estimate and the actual effort spent on the project. The results were used to find the estimation accuracy of the method or tool by using the *'Symmetrical Relative Error'* [MJ01] as a measure of accuracy. The results were used as input to the Evaluation Profiles of the tools.

The software company wanted an estimation method or tool that computed accurate estimates and was easy to learn and use. The project manager of Case Study A was the sponsor and initiator of the feature analysis, and provided the funds for the study. I have played the part of both the evaluator and the assessor. The technology users were the people in the software company who would use the selected method or tool. I chose the features to be investigated after interviewing the project manager of the project in Case Study A. I also set the scores for each feature. The results are presented in Chapter 8.

Chapter 5

Industrial Case Studies

In this chapter, I present two case studies which I conducted in a software company in order to investigate various aspects of the use case points method. Case Study A is described in Section 5.2, and Case Study B in Section 5.3.

My task was also to select an estimation method or tool that would be appropriate for the needs of the software company in question. The tools 'Optimize' and 'Enterprise Architect' described in Section 2.3, and the use case points method described in Section 3.2 were used to compute estimates of the development projects. The results of the method and tool evaluation are presented in Chapter 8.

5.1 Background

The two Case Studies A and B were conducted in a local branch of a major European software company, situated in Norway. The goal of the studies was to add to existing experience with the use case points method, to determine the appropriate level of detail in use cases written for estimation purposes, and to investigate if the Technical Complexity Factor TCF) can be dropped in the use case points method.

The software company in question wished to select a cost estimation method or tool suited to their specific needs. I was therefore given access to project data from two projects. Software development in the organisation is now mainly object-oriented, and the Rational Unified Process (RUP) is used in many development projects. It was therefore natural for the organisation to consider adopting a use case-driven approach to estimation.

The local branch of the company has a total of 600 employees. Among the main areas of business are Internet Application Development, Customer Relationship Management (CRM), and Extended Enterprise Applications (ERP). The department where the case studies were conducted uses UML and RUP in their development projects. There is a lack of methodological support in the estimation process, and estimates are based on expert knowledge and successive calculation. Successive calculation is a bottom-up approach to cost estimation. The principle is to control and reduce

uncertainties in estimates by splitting up development tasks into smaller tasks and estimating each task separately.

The projects in the two Case Studies A and B differed in many ways. In Case Study A, there were no textual use case descriptions, and I therefore found it necessary to search for approaches to sizing that extended the original use case points method. In Case Study B, some of the use case descriptions were lacking in detail, and sequence diagrams had to be used for counting transactions and defining complexity. Both projects are probably typical of much of today's software development, where not enough attention is given to writing fully dressed use case descriptions. Estimating effort therefore becomes more complex than when the use cases are written at an appropriate level of detail.

5.1.1 Information Meetings

Before I commenced my investigations in the software company, I was invited to a meeting with the project manager and supervisor of the development team in Case Study A, and two senior consultants. I explained the use case points method and that I needed data from real projects for my investigations. It was decided that I could have access to data from two projects, one that was finished, and one that was in the starting phase.After having received all the necessary documentation, I computed estimates of the projects. The data collection is described in Section 5.2.2.

A second meeting was held with the same participants where I presented the estimation results of Project A. Actual effort had not been revealed before I made the estimates, in order to avoid biases and effects of *anchoring and adjustment* - the insufficient adjustment up or down from an original starting value, or 'anchor' [Plo93]. The results shows that the estimates obtained with the use case points method were very close to actual effort. The project manager had underestimated the project, and to him, the results of estimates produced with the use case points method were very inspiring. He therefore applied for financial funds for a small project that was to further investigate the results of applying the use case points method to project data, and comparing these to estimates produced with the two commercial tools.

After the investigations were finished, I was again present at a meeting that involved a group studying the use of object-oriented analysis and design in the company. I presented the results of my investigations and the feature analysis described in Chapter 8. The use case points method had performed very well, and was accepted as an estimation method to be studied in further detail in the development department where the case studies had been conducted.

5.2 Case Study A

5.2.1 Context

The developed software was an Internet application for a major bank in Norway. The purpose of the software solution was to improve communic-

Phase	Estimate in hours	Actual effort in hours
1	3950	4971
2	3820	5072
Total	7770	10043

Table 5.1: Estimates of the 2 Phases in Project A

ation between customers and their contacts by enabling users to upload news, download documents and view current activities.

The development team consisted of five programmers and one technical supervisor who was responsible for analysis and testing. The application was built using a standard three-layer model with a presentation layer, application layer and a database layer, with Java2Enterprise on the application side. The development tools used were Websphere from IBM, and Dynamo from ATG. The project started on October 1st 2000, and finished on May 24th 2001. Early estimates were made using successive calculation. These estimates and the actual effort spent on the project were not revealed until the estimates produced by the use case points method and the tools were presented, in order to avoid biased results.

The project was divided into two phases. Phase 1 was concerned with requirements specification and technical issues, and was supposed to start in August 2000. However, it was delayed, and started on October 1st 2000. Actual staff hours spent on this phase were 4971. The preliminary estimate was 3950 staff hours. Phase 2 was the development phase. Actual staff hours spent on the project were 5072, and the preliminary estimate was calculated to 3820 staff hours.

These Figures are shown in Table 5.1. The project was grossly underestimated.

The application was developed in accordance with the client's new e-infrastructure. It soon became clear that the infrastructure was neither ready nor well documented, so much time was spent trying to find out details. Phase 1 therefore did not start until the beginning of October 2000. The original development plan, which was written at the end of August, was based on an incremental development method; IAD (Incremental Application Development) with two-week cycles. Owing to uncertainties concerning the framework, Phase 1 was not conducted in accordance with the IAD method or the planned schedule. The team had little experience with Java, two of the developers were novices, and the selected development technology was unknown to the whole team, so the time schedule turned out to be unrealistic. In Phase 2 the IAD method was used, and development proceeded according to the project plan. The activity breakdown is shown in Table 5.2.

5.2.2 Data Collection

This project was finished when the investigations started. Interviews were conducted with the project manager to establish the values of the technical and environmental factors, and with the supervisor to obtain the use case

Activity	Percentage	Staff hours
Programming	58	5831
Testing	22	2230
Data capture	5	493
Project Management	11	1.100
Planning	4	389
SUM	100	10043

Table 5.2: Breakdown into Activities of Project A

models and class models, and define use case and class complexity. The supervisor had three years of experience with UML and use case modeling.

The documentation that was made available to me were project plans, use case models, class diagrams, technical reports and the evaluation report. The use case model did not have extending and included use cases. The reason for this choice of design was to make the model more understandable to the customer. Unfortunately, there were no textual use case descriptions to be found in the use case model, and no use case documents. There were also no sequence diagrams. The team had made the use case model with stick figures and ellipses, and started coding right away.

The supervisor was responsible for the analysis phase, and for testing. Since the use cases did not have textual descriptions, I asked her how she would define use case complexity. She decided that the amount of reuse of code in each use case was a measure of the amount of work needed to be done. Reuse implies less programming effort. She therefore assigned complexity by using degree of reuse as a complexity measure:

- A simple use case has extended reuse of code,
- a medium use case has some reuse of code, and
- a complex use case has no reuse of code.

Looking back on this process, I concluded that complexity could also have be assigned in the way described in the tool 'Optimize'. This approach is to consider the amount of use case description which would have to be written to document the specific business activity. See Section 4.1.1. The supervisor could also have used the approach proposed for signing complexity in the tool 'Enterprise Architect', where amount of work coupled with how many database entities are involved is a complexity measure. See Section 4.2. However, none of these approaches are as prescribed by Karner, so they are really variations of sizing in the use case points method.

A question is whether the supervisor decided complexity on the grounds of the source code. If this had been the approach, applying the use case points method would have little meaning, as estimates should be made at the beginning of a project with the aid of the use case model. But as pointed out by Bente Anda *et al.*, the source code can be used to verify that use case complexity is assigned correctly, by comparing the size of each use case with the number of classes or lines of code necessary to implement it [ADJS01]. I did not, however, use this approach in this case.

Factor	Description	Value	Reason
T1	Distributed System	5	Yes
T2	Response adjectives	4	Response time is important
T3	End-user efficiency	5	Must be efficient
T4	Complex processing	3	Average complexity
T5	Reusable code	3	Some reuse planned
T6	Easy to install	4	Customer requirement
T7	Easy to use	4	Customer requirement
T8	Portable	0	No issue
T9	Easy to change	4	Changes will be made
T10	Concurrent	4	Interfacing with other systems
T11	Security features	3	Average
T12	Access for third parties	1	Not important
T13	Special training required	3	Some training needed

Table 5.3: Evaluation of the Technical Factors in Project A

Asking the supervisor how she had assigned complexity revealed that although she knew the source code, she stated that she had based her decisions on her knowledge of the amount of functionality in each use case coupled with her experience of how much work had to be done to implement it. This was the approach she would have used if she had been assigned the task of sizing the use cases at the beginning of the project. I decided that the method applied by the supervisor for assigning use case complexity may be used as a general approach when the use cases have little or no textual descriptions.

The technical and environmental factors were set by the project manager with the aid of two senior developers. Some of the technical factors were unclear. Factor T10 'Concurrent', presented a certain difficulty. This could for instance mean parallel processing, parallel programming, or whether the system is stand-alone or interfaces with several more applications. There are no guidelines in the use case points method explaining exactly what this factor is supposed to measure. The project manager decided that in this case it meant interfacing with other systems, and gave the factor a high score. The reasons given for setting the technical factors are shown in Table 5.3.

The second interview with the project manager lasted one hour and consisted of an informal talk of the results obtained so far, and a short interview about the preliminary estimation process. We established that the use case points method should be used without the technical complexity values, because of the difficulties with the counting rules, and because estimates produced without the technical complexity factor are not very different from estimates with the technical complexity factor. This talk also established the features to be investigated in the feature analysis. See Chapter 8.

5.2.3 Setting the Values for the Environmental Factors

The project manager set the factors based on the following reflections:

1. **F1: Familiar with Development Process**

   The team members were not very familiar with RUP, but used a different development process, IAD, Incremental Application Development. IAD was known to the team. Score 3.

2. **F2: Application experience**

   The technology used was new to most of the team members, but they all had some application experience. Score 3.

3. **F3: Object-oriented experience**

   Most of the team had experience with object-oriented analysis and design. The supervisor who was responsible for analysis knew Rational Rose. Score 3.

4. **F4: Lead analyst capability**

   The lead analyst was in this case the supervisor who had 3 years of experience with UML modeling. Score 4.

5. **F5: Motivation**

   After the initial period with unstable requirements, motivation was high. Score was set to 4.

6. **F6: Stable requirements**

   The requirements were unstable due to the insecurities on the customer's side. Score 2.

7. **F7: Part-time workers**

   This was not an issue, as there were no part time workers. Score 0.

8. **F8: Difficult programming language**

   The team had little experience with Java, two of the developers were novices. However, they had general programming experience. Score 3.

   These reflections and the scores are shown in Table 5.4.

5.2.4 The Estimates

The technical complexity factor, TCF, and the environmental factor, EF, are computed in the following manner:

$$\text{TCF} = 0.6 + (0.01*44) = 1.04$$
$$\text{EF} = 1.4 + (-0.03*16) = 0.92$$

Factor	Description	Value	Reason
F1	Familiar with IAD	3	Average experience
F2	Application experience	3	Average experience
F3	Object-oriented experience	3	Average experience
F4	Lead analyst capability	4	3 years of experience
F5	Motivation	4	Highly motivated
F6	Stable requirements	2	Quite unstable
F7	Part-time workers	0	No issue
F8	Difficult programming language	3	Average experienced team

Table 5.4: Evaluation of the Environmental Factors in Project A

Factor	Description	Weight	Value	Weighted value
T1	Distributed System	2	5	10
T2	Response adjectives	1	4	4
T3	End-user efficiency	1	5	5
T4	Complex processing	1	3	3
T5	Reusable code	1	3	3
T6	Easy to install	0.5	4	2
T7	Easy to use	0.5	4	2
T8	Portable	2	0	0
T9	Easy to change	1	4	4
T10	Concurrent	1	4	4
T11	Security features	1	3	3
T12	Access for third parties	1	1	1
T13	Special training required	1	3	3
TFactor				44

Table 5.5: Technical Factors in Project A

Factor	Description	Weight	Value	Weighted value
F1	Familiar with RUP	1.5	3	4.5
F2	Application experience	0.5	3	1.5
F3	Object-oriented experience	1	3	3
F4	Lead analyst capability	0.5	4	2
F5	Motivation	1	4	4
F6	Stable requirements	2	2	4
F7	Part-time workers	-1	0	0
F8	Difficult programming language	-1	3	-3
EFactor				16

Table 5.6: Environmental Factors in Project A

Effort	Expert estimate	With TFC	Without TCF	Optimize	EA
10043	7770	10831	10414	10856	10831

Table 5.7: All estimates for Project A

I made four estimates: two with the use case points method in a spreadsheet, with and without the technical complexity factor, one with the tool 'Enterprise Architect', and one with the tool 'Optimize'. The estimates are shown in Table 5.7. 'Effort' means actual effort spent on the project, 'With TCF' is the estimate made with the use case points method with the technical complexity factor, without TCF is the use case points method without the technical complexity factor, 'Optimize' is the effort produced by the tool 'Optimize', and 'EA' the estimate produced with the tool 'Enterprise Architect'.

5.2.5 Estimate Produced with the Use Case Points Method

Actor complexity was defined according to Karner's method. The two actors were both people acting through a GUI. The following input values were used:

2 complex actors, 18 simple, 41 medium and 4 complex use cases.

The UUCP are the unadjusted use case points obtained from adding the unadjusted actor weights, UAW, and the unadjusted use case weights UUCW:

$$UUCP = 2*3 + 18*5 + 41*10 + 4*15 = 566$$

The adjusted use case points were worked out:

$$UCP = UUCP*TCF*EF = 566*1.04*0.92 = 541.54 = 542$$

The UCP were multiplied by staff hours per use case point, which in this case is 20, owing to team experience and stability expressed in the environmental factors. See section 3.3. This yielded an estimate of 10831 hours.

The results of the estimation process are shown in Table 5.7.

5.2.6 Omitting the Technical Complexity Factor

An estimate was also produced omitting the TCF. The UCP were calculated in the following manner:

$$UCP = UUCP*EF = 566*0.92 = 520.72$$

Multiplying this figure with 20 staff hours per use case point yielded a total effort of 10414 staff hours. The difference between this estimate and the estimate made with the TCF was 417 hours, and the estimate without the TCF was closer to the actual effort of 10043 hours.

5.2.7 Estimate produced by 'Optimize:

For computing estimates with the tool 'Optimize', the number of analysis classes are needed as well as the number of use cases and their complexity. I applied the use case complexity that had been defined by the supervisor. There were therefore 18 simple, 41 medium and 4 complex use cases.

Unfortunately, there is no proven relationship between number of use cases and number of classes, so it is important to spend time during analysis finding the classes that implement the functionality described in the use cases [Fac]. There was no business concept model or analysis model, so the supervisor provided all the class models and a high level class diagram of the Enterprise Java Beans (EJB). The application had 16 session beans, and I decided to use these as input to the tool, because they implemented the functionality described in the use cases. I used the default value 'medium' for describing complexity, as there was no information about class size and complexity at this stage.

The estimate yielded a total effort of 5536 staff hours. This is about half of the estimate obtained with the use case points method. However, there is no activity called 'Project management' in the breakdown into specific activities. Including this activity in the estimate would have increased the total number of staff hours.

The project manager would have liked a tool functionality mapping the development hours back to specific use cases, in order to see the amount a work connected with a specific use case.

5.2.8 Estimate Produced by 'Enterprise Architect'

Because input to the estimation function in the tool is the same as in the use case points method, estimates produced with 'Enterprise Architect' are the same as those produced with the use case points method when the effort per use case point is set to 20 staff hours: 10831 staff hours. See Table 5.7. Staff hours per use case point must be defined for each specific project. I have therefore used the same value as in the use case points method.

5.2.9 Comparing the Estimates

Karner's method produces an estimate of 10831 hours. Actual effort spent on the project was 10043 hours. However, omitting the technical complexity factors produces an estimate of 10414 hours, which is closer to the actual effort. The tool 'Optimize' produces an estimate of 5536 hours, which is roughly half of the estimate produced using Karner's method. One of the reasons is that project management was not included as an activity. Another reason may be that the number of classes that implement the use cases is incorrect. When I discussed this issue with the supervisor, we agreed to count session beans, without knowing whether they actually implement all the functionality described in the use cases. In order to obtain an estimate more in accordance with the estimate obtained using Karner's method, it is necessary to include all the 130 classes, defining the 35 subclasses as 'small' in size with 'trivial' complexity, and the rest of the classes

as 'medium' sized with 'medium' complexity (tool default values), since little is currently known about the actual values. These values produce an estimate of 10856 staff hours.

However, this approach gives very biased results, as I already knew the value for the estimate computed by the use case points method, and adjusted the number of classes and their complexity to fit this value. Still, I did use all the classes as input, not just a selection. Whether this result really does indicate that all the design classes should be used as input, will have to be verified by trying out the same approach on several more projects.

In the use case points method, effort per use case is 20 hours, because the number of environmental factors in F1 through F6 that are above 3 added to the number of factors in F7 through F8 that are below 3 are 2. Therefore, the estimates produced with the use case points method in a spreadsheet are identical to the estimates produced by the tool 'Enterprise Architect', when effort is set to 20 hours per use case point.

Omitting the TCF results in an estimate closer to the actual effort. Also, it is not very different from the estimate with the TCF. I conclude that this indicates that the TCF does not improve estimates in general. All the estimates are shown in Table 5.7.

5.3 Case Study B

5.3.1 Context

The application in question was a real-time system developed as part of a large commercial system. A new system was to substitute the existing system, and the development task was to build an application that interfaced with existing internal systems. The application consisted of four subsystems, and these subsystems were developed by separate teams. The teams consisted of 3-4 members, and were composed of developers coming from both the customer and the software company. The project started in February 2001, and the dead-line for development was supposed to be November 1st 2001.

The modeling tool was Rational Rose, the programming language C++ for UNIX, and the development process was RUP. An iterative development approach was used. Programming started before all the design was finished. Team 4 started coding before analysis was completed, so the use cases were not written in full detail.

The project manager had assigned the task of making estimates to the team leaders of the separate teams. This decision was based on internal knowledge of who was good at coming up with reliable figures. Estimates were made using successive calculation.

The teams were supervised by a mentor, who had two and a half years of experience with use case modeling, Rational Rose, and UML when the project started. The mentor had been assigned the job on the project because she had knowledge of UML and use case modeling. The teams had very little or no experience with UML, so the role of the mentor was essential. She worked on the project until the start of the design phase.

The customer's developers had no experience with use case modeling, less programming experience than those who came from the software company, and no experience whatever with Rational Rose or RUP. The developers from the software company had some knowledge of object-oriented modeling, but only one had experience with Rational Rose. Learning RUP was one of the difficulties and frustrations the teams had to cope with. They were unfamiliar with producing great amounts of documentation, and wanted to get quickly down to programming. Learning Rational Rose was comparatively simple. Programming in C++ on the UNIX platform was a source of frustration to the teams, and also a potential risk, as the developers were unfamiliar with the tools and with UNIX.

This application was very different from the one in Case Study A, in the respect that it consisted of four subsystems developed by four different teams. Three of the subsystems were real-time systems, and system security was crucial.

The software solution was technically complex. The application had to interface with many existing internal systems. The mentor expressed a wish to be able to account for more of this complexity by including some factor to describe it. This shows that the technical complexity factor does not account well enough for the complexity of today's software systems, and this is one of the weaknesses of the use case points method.

5.3.2 Data Collection

The UML models for all the subsystems were sent by e-mail from the mentor on the project before I interviewed her. From these models I collected the following data: use case diagrams, textual descriptions of use cases, class diagrams, a few state transition diagrams and sequence diagrams. Technical and environmental complexity factors were set by the mentor in the course of one interview. The project manager supplied all the estimates after I had computed the first estimates, as I wished to avoid biased results.

Actor complexity and values for the technical and environmental factors were assigned by the mentor, who knew the system and all the teams well after working with them for several months.

The use cases had in part very technical textual descriptions. The actors were mostly other systems, also very technically complex. According to the mentor, one goal when modeling the use cases was to avoid special use case documents, so all the use case documentation was written in the use case model.

When assigning values to the technical factors, the mentor felt the need for some factor to describe that the application was very technically complex. Unfortunately, there are no ways to add to the lists of the technical or environmental factors. Again, the factor T10 'Concurrent' presented a problem. Since a factor was needed to describe that the four subsystems processed data in parallel, and that it was a real-time solution, this factor could be used. Multiple processors are used for real-time systems needing a lot of computational performance. Often large amounts of data must shift between data acquisition, data display, and processor boards. The

Factor	Description	Value	Reason
T1	Distributed System	0	No issue
T2	Response adjectives	5	Response time is crucial
T3	End-user efficiency	5	No 'bottle necks' tolerated
T4	Complex processing	5	Very technically complex system
T5	Reusable code	3	Some reuse planned
T6	Easy to install	0	No issue
T7	Easy to use	0	No issue
T8	Portable	4	Planned portability
T9	Easy to change	4	Requirements will change
T10	Concurrent	4	Multiple processing
T11	Security features	5	Security is crucial
T12	Access for third parties	4	Must give access
T13	Special training required	0	No issue

Table 5.8: Evaluation of the Technical Factors in Project B

data must move quickly and efficiently [Rul01].

Considering these issues, the mentor decided that in this case, 'Concurrent' meant multi-processing. The other technical factors were straightforward, and the mentor spent about 20 minutes assigning the values. The reasons for assigning the specific values are given in Table 5.8.

The mentor also assigned values for the environmental factors, sharing her reflections while I took notes. The scores and the reasons for each score are shown in Table 5.9 on the next page. The following is an explanation of the specific scores.

1. **F1: Familiar with RUP**

   The team members were unfamiliar with RUP. They wanted to learn more, but there was little time. The mentor had both knowledge and experience, but this knowledge was difficult impart. The result was that the teams used what they could of RUP. The score was set to 1.

2. **F2: Application experience**

   Some of the technology used was new to most of the teams, but they all had extensive application experience. Score 4.

3. **F3: Object-oriented experience**

   Only one person on all the teams apart from the mentor had experience with Rational Rose. This could have been a potential problem. As it turned out, learning the tool and modeling use cases went smoothly. Score 3.

4. **F4: Lead analyst capability**

   The mentor evaluated her own capability. She had two and a half years of experience with object-oriented analysis and design, and set the score to 4.

Factor	Description	Value	Reason
F1	Familiar with RUP	1	Unexperienced teams
F2	Application experience	4	Average experience
F3	Object-oriented experience	3	Average experience
F4	Lead analyst capability	4	Experience from several projects
F5	Motivation	4	Highly motivated
F6	Stable requirements	4	Yes
F7	Part-time workers	0	No issue
F8	Difficult programming language	3	C++ for UNIX

Table 5.9: Evaluation of the Environmental Factors in Project B

5. **F5: Motivation**

   After a period of resistance to using a new development method (RUP), which the teams characterized as 'office work only', things went smoother and the teams settled down to work, and were now doing very well. Motivation was therefore high, although it was not so in the beginning. Score was set to 4.

6. **F6: Stable requirements**

   The requirements phases lasted several months, and there were no changes up to the the moment when scores for the environmental factors were set. All the teams wrote requirements in collaboration with the customer. However, some major changes were expected. The score was therefore set to 4.

7. **F7: Part-time workers**

   This was not an issue, as there were no part time workers. But one problem was for all the developers to get time off from their daily tasks to participate on the projects, so the number of team members could vary somewhat from time to time. Score 0.

8. **F8: Difficult programming language**

   The development language was C++ for UNIX. This was seen as a potential risk at the beginning of the project. However, the developers grew comfortable with the programming environment after a while. Score 3.

The four subsystems and the teams were different in some respects, and I had expected that the technical factors would vary somewhat. However, when going through the details, the mentor assigned the same technical factors to all the subsystems. The environmental factors were the same for all teams, as the level of experience was the same. This factor demands that the team must be seen as a whole, even if one or several team member are more experienced than the others.

Factor	Description	Weight	Value	Weighted value
T1	Distributed System	2	0	0
T2	Response adjectives	1	5	5
T3	End-user efficiency	1	5	5
T4	Complex processing	1	5	5
T5	Reusable code	1	3	3
T6	Easy to install	0.5	0	0
T7	Easy to use	0.5	0	0
T8	Portable	2	4	8
T9	Easy to change	1	4	4
T10	Concurrent	1	4	4
T11	Security features	1	5	5
T12	Access for third parties	1	4	4
T13	Special training required	1	0	0
TFactor				43

Table 5.10: Technical Complexity Factors in Project B

5.3.3 Input to the Use Case Points Method and the Tools

I counted all the use cases and defined use case complexity for each system separately. Complexity was assigned by counting either use case transactions, transactions from sequence diagrams or implementing classes. The approach for each subsystem is described below, and is dependent on the level of textual detail in the use cases. Actor complexity was assigned by the mentor. The value for the actors and use cases were used as input to the use case points method, together with the values for the technical complexity and environmental factors. I used an Excel spreadsheet to compute the estimates.

The same input was used in the tool 'Enterprise Architect.' Since 'Enterprise Architect' is primarily a UML CASE-modeling tool, I had to enter all the use cases into the Use Case view of the analysis phase, which was rather tedious work. I then set the values for the technical and environmental factors, and the default value for staff effort per use case point to 28, to be in accordance with the value in the use case points method.

Input to the tool 'Optimize' was the number of use cases and their complexity, and the number of classes. I decided that there was no need to count complexity twice, even though the approach to defining use case complexity is slightly different than in the use cases points method. Classes are also defined by their size, which ranges from 'tiny' to 'huge' on an ordinal scale of five steps. I decided that if use case complexity was 'simple', I would set size to 'small'. Similarly, if use case complexity was 'medium', size would be 'medium', and if use case complexity was 'complex', size would be 'large'. I based this decision on the fact that I did not have enough information on the classes to differentiate this more. Besides, ideally the estimate should be computed using analysis classes, not design classes.

Factor	Description	Weight	Value	Weighted value
F1	Familiar with RUP	1.5	1	1.5
F2	Application experience	0.5	4	2
F3	Object-oriented experience	1	3	3
F4	Lead analyst capability	0.5	4	2
F5	Motivation	1	4	4
F6	Stable requirements	2	4	8
F7	Part-time workers	-1	0	0
F8	Difficult programming language	-1	3	-3
EFactor				17.5

Table 5.11: Environmental Factors in Project B

5.3.4 Estimates Produced with the Use Case Points Method

Estimates were made for each subsystem separately. The total size of the system was obtained by adding up these estimates. The results are shown in Table 5.12. The details of the estimation process are as follows:

5.3.5 Assigning Actor Complexity

Deciding the complexity of the actors was not straightforward, as the actors did not quite fit the actor definitions in the original method. Although there is a 'customer' actor in Subsystem 1, it does not interact with the system through a web page or GUI, but is an internal user with an automated subscription to internal information. This actor is in fact more like a system acting through a protocol. This actor was therefore characterized as 'average'. All the other actors are other systems, but with different degrees of complexity. Three of the subsystems are real-time applications, feeding each other with real-time data, and appearing as actors to each other. Since some of these actors are very complex, there should have been a complexity scale that could account for more levels of complexity than the 'simple', 'average' and 'medium' of the original use case points method. Since this is not the case, the most complex actor only gets 3 points, and this may be misgiving. The mentor assigned complexity to the actors after a round of discussion with myself as to which kind of systems the different actors interacted with.

5.3.6 Assigning Use Case Complexity

I counted use case steps and assigned use case complexity from the use case models and descriptions. For subsystems 1 and 3 this was straightforward. Subsystem 2 had use case descriptions that seemed to be lacking in detail to some extent. However, I could not be sure if this was the case, or if most the use cases were simply small. Complexity was assigned by counting the use case transactions, and verifying the count by viewing the class diagrams. Use case complexity can be assigned by counting the number of implementing analysis classes. This was done for Module 2 of Subsystem 4,

Subsyst.	UUCP	UC Estimate	Without TCF	Expert Estimate	Actual effort
1	163	4113	3994	3450	3723
2	161	4062	3920	2615	3665
3	145	3659	3553	3235	3835
4	123	3129	3038	3300	2710
Total	593	14965	14528	12600	13933

Table 5.12: Estimates made with the use case points method in Project B

since it had scanty use case descriptions. For Module 1 in Subsystem 4 the use cases had very little detail, and complexity was assigned with the use of sequence diagrams. Ideally, transactions should be counted at an earlier stage in development, directly from the use case descriptions to be in accordance with the prescriptions of the use case points method. In reality, most people write either detailed use cases or make sequence diagrams, seldom both. It is a question of time. So when use cases lack textual detail or there are no use case descriptions at all, counting transactions from sequence diagrams is a possible approach to sizing.

5.3.7 Computing the Estimates

By studying the use case models and the textual use case descriptions, I saw that the four teams had very different approaches to writing use cases. Team number 1 wrote detailed use cases where it was easy to count the transactions. Teams number 2 wrote less detailed use cases, Team 3 wrote very well structured use cases with detailed documentation of the purpose and functionality of each use case. Subsystem number 4 consisted of two modules. One module had use case descriptions that were not very detailed. The second module had no textual use case descriptions, therefore, complexity was defined by counting the classes that implemented these use cases.

There were several extending and included use cases. Karner recommended not counting these use cases, but in this application, much of the technically complex functionality was expressed in included and extending use cases. Not counting them might mean that important functionality would not be sized. I therefore decided to count all the extending and included use cases.

Actors and their complexity for all the sub-systems are shown in Table 5.13, use cases and their complexity in Table 5.14. The estimates with and without the technical complexity factor, actual effort, and expert estimates are shown in Table 5.12.

The estimation processes for the specific sub-systems are described in the next sections.

5.3.8 Subsystem 1

Subsystem 1 was not a real-time application. The team consisted of 4 developers. The documentation available was the use case model, with tex-

tual descriptions of the use cases written as use case steps, and a few state transition diagrams. There were also class diagrams, but no sequence diagrams. There were 19 use cases and 6 actors. The use case descriptions were detailed and captured all the functionality, so I assigned use case complexity by counting the transaction steps. The following use case example shows how this team wrote use cases:

Use Case example, Team 1

```
This use-case allows IT operations to do a
consistency check to verify that basis data updated
in xxxx-system is correctly transferred to the xxx database.

Pre-conditions:
1. xxxx -system is started.

Basic Flow:
Steps:
1. Get data from xxx.
2. Get extract from xxx database.
3. Run consistency check.
4. Report discrepancies.
5. Verify discrepancies.

Alternative Flow:
Described in the activity diagram "Verify Consistency".

Post-conditions:
2. The two databases are consistent
```

There are 5 use cases steps in this use case. The alternative flow is described in a state diagram. The digram shows two transitions, or transactions. The main success scenario has 5 steps, and the alternative flow has two. Counting these transactions gives a total of seven transactions. Use case complexity is therefore 'medium'. I counted transactions for all the use cases in the same manner. 2 of the use cases had alternative flows described in state diagrams, and one in an extended use case. There were 10 simple, 7 medium and 2 complex use cases.

The actors were other systems and one customer actor. However, most of the other systems were highly complex, being real-time systems feeding each other continuously with data. Security was crucial. It did not seem correct to characterize the 5 other systems as 'simple' actors, because of all the interfacing and processing between the systems. The mentor therefore

decided that there were 2 complex actors, 3 medium actors and 1 simple actor.

The actors and use cases were counted, sized, and the the values entered into a spreadsheet with the technical and environmental factors. Two estimates were then computed, with and without the technical complexity factor.

The UUCP are the unadjusted use case points obtained from adding the unadjusted actor weights, UAW, and the unadjusted use case weights UUCW:

$$\text{UUCP} = 1*1+ 3*2+ 2*3 + 10*5 + 7*10 + 2*15 = 163$$

The adjusted use case points were worked out and multiplied by staff hours per use case point, which in this case was 28, owing to team experience and stability expressed in the environmental factors. This yielded an estimate of 4133 staff hours with the TCF, and 3994 staff hours without the TCF. The estimate without the TCF was the closest to the actual effort. See Table 5.12.

5.3.9 Subsystem 2

Team 2 consisted of 3-4 developers. The subsystem was a real-time system and was modeled by 21 use cases, all described in use cases steps, and 4 medium and 1 complex actor. An example of use case writing in this team is the following:

Use Case example, team 2

```
Use case Get dynamic data

The use case executes when a received broadcast is
received.
Pre-Conditions:
The getStaticData use case has ended, and a broadcast
transaction has been received

Basic flow:
Scenario:
* Identify transaction
* Forward transaction to appropriate subroutine.

Alternative flow:
*If transaction is not of any value, ignore transaction.

Post-Condition:
An acknowledgment has been received
```

I counted the transactions from the use case steps. There were 12 simple and 9 medium use cases. The use cases looked as if they might not have enough detail. For instance step 2 'Forward transaction to appropriate subroutine', may include some functionality that is not seen. I therefore studied the class diagrams to count classes that implement these use cases, to verify if complexity was correct. A simple use case is implemented by less than 5 classes, and medium use case by 5 to 10 classes [SW98]. Most of the use cases were implemented by less than 10 classes, and many by less than 5, so I decided that use case complexity was assigned correctly. There were no sequence diagrams by which I could verify that the counts were quite correct, but this is one of the uncertainties of the method.

The UUCP are obtained from the formula

$$\text{UUCP} = 0*1+ 4*2+ 0*3 + 12*5 + 9*10 + 0*15 = 161$$

The estimate for subsystem 2 was 4062 staff hours with TCF, and 3920 staff hours without TCF. The estimate without the TCF was the closest to actual effort. See Table 5.12.

5.3.10 Subsystem 3

Subsystem 3 was a real-time system with 5 actors and 13 use cases. The development team consisted of 3-4 people. The use cases were written in detail, and contained extensive documentation and explanations of the use cases in the the textual descriptions. This system was one of the most complex subsystems, the other was Subsystem 4.

The model also included sequence diagrams for all the use cases. This made it possible to verify that the use case transaction counts were correct. When counting transactions from the sequence diagrams, I observed that they all had the same logging on and security issues. This means reuse, and these transactions should not counted for each use case, since the functionality is implemented only once. I compared the transaction counts from the use cases to the transaction counts from the sequence diagrams. These were practically the same, and yielded the same use case complexity. This also verified that I had counted transactions correctly for subsystem 4, module 1.

An example of use case writing in team 3 is as follows:

Use Case Example, Team 3

```
Use case Update Order.
The use case updates orders in the Order Book.

Pre-Conditions:
Assumes that a message of type update order is received.
```

```
Sequence :
1. Find correct order if it exists
2. Delete/update volume on the order
3. Calculate new value
4. Calculate sum
5. Get number of orders in OrderBookLine
6. Create 'OrderbookStatus' message

Post-Conditions:
The message is processed and updated in the order book,
and a valid message is made.
-----------------------------------------------------------
```

This use case has 6 transaction steps. Exceptions are handled in extending use cases, which are counted separately. This means that the use case is of medium complexity. The corresponding sequence diagram is shown in Figure5.1.

The sequence diagram has 9 transactions:

1. Handle Order
2. decodeMessage
3. updateOrder
4. getParticipants
5. calculateValue
6. calculateSum
7. getNumberofOrders
8. returnOrders
9. writeMessage

However, not all these transactions describe 'what' to do, but 'how' to do it. For instance step 2, 'decodeMessage' is too detailed, the same goes for 'returnOrders.' The functionality is expressed in 'getOrders'. I removed these two transactions and obtained 7 transactions, or a use case of medium complexity.

Considering the use case description again, I decided that use case step 2 'Delete/update volume on the order ' should really be written something like this:

- if it is a 'update order' message, then update order
- else include use case 'Delete Order'.

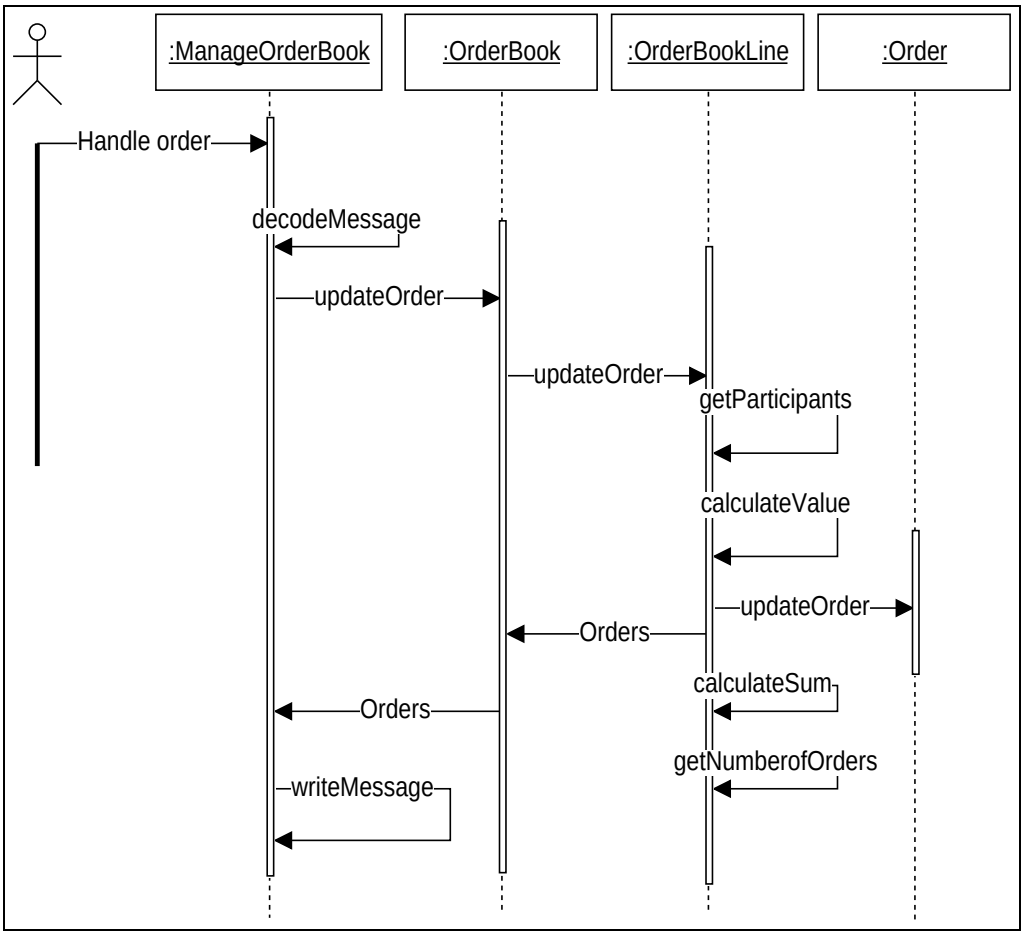

Figure 5.1: Sequence diagram for 'Update Order' Use case

The reason is that both the 'Delete Order' and the 'Update Order' functionality must be described, 'Delete order' by invoking the use case 'Delete Order.' This step should be counted as two steps. The use case would then have 7 transactions like the sequence diagram. The transaction 'getParticipants' in the sequence diagram can not be seen in the use case description, and is probably hidden in the functionality of some other step. A use case of medium complexity has from 4 to 7 use case steps. Using the sequence diagram therefore verified that complexity had been assigned correctly.

I counted the use case steps for all the use cases, and compared them to the transactions of the sequence diagrams. This process resulted in the following values:

Subsystem 3 consisted of 3 simple, 4 medium, and 6 complex use cases. The 5 actors were all of medium complexity since they were systems acting through protocols.

The UUCP were obtained from the formula

$$UUCP = UAW + UUCW$$
$$UUCP = 0*1+ 5*2+ 0*3 + 3*5 + 6*10 + 4*15 = 145$$

The estimate with TCF and effort per use case point 28 staff hours is 3659 hours, and without TCF 3553 hours. Actual effort was 3835 staff hours, and the expert estimate was 3235 hours. This subsystem was underestimated both by the expert estimate and the use case points method. The estimate with the TCF was the closest to the actual effort. See Table 5.12.

5.3.11 Subsystem 4

Team four consisted of 2-3 developers. Subsystem 4 was the most difficult to size, and there may be some insecurities regarding the results. **In module 1**, there were 6 use cases with some detailed descriptions :

Use Case example, team 4, module 1

```
------------------------------------------------------------
Use case GetValue
Receives values, and publishes values

Scenario:
 -Ask for values
 -Publish according to publication rules.
  (Only one publication rule is currently
  supported: Continuous publishing.)

Exceptions:
 -If XXXX is unable to receive messages, the message
  must be buffered
------------------------------------------------------------
```

Subsystem	Simple actors	Medium actors	Complex actors
1	1	3	2
2-RT	0	4	1
3-RT	0	5	0
4-RT	0	4	2

Table 5.13: Actors and their complexity in Project B

This was a real-time application, and I could see this use case had too little detail in the main success scenario. There were only three transactions, defining a simple use case. This did not seem convincing. I therefore counted transactions from the sequence diagrams in the way I had done for subsystem 3, omitting all the logging on and security issues, since they were the same for all the use cases. The use case 'GetValue' shown above had 6 transactions in the sequence diagram. Recounting in this manner yielded higher complexity than the first counts, when all the use cases were counted as simple. The module had 4 medium and 2 complex use cases.

Module 2 had very little textual description:

```
-----------------------------------------------------------
```

Use Case example, team 4, module 2

```
-----------------------------------------------------------
Use Case Detect Capacity
Detects if system is experiencing capacity problems
-----------------------------------------------------------
```

For this module, there were no sequence diagrams. There were four use cases, and I counted the classes that implemented them, using the class diagram. Ideally, analysis classes should be counted according to Schneider and Winters [SW98], but as there were none, I used the class diagram from the design.

There were 4 medium use cases.

The two modules had the same 6 actors, 1 simple, 3 medium and 2 complex.

The UUCP for the whole of subsystem 4 are:

$$\text{UUCP} = 1*1 + 3*2 + 2*3 + 0*5 + 8*10 + 2*15 = 123$$

The estimate with TCF was 3129 hours, and 3038 without the TCF. The estimate without the TCF was the closest to the actual effort.

5.3.12 Estimation Results

Two estimates were computed for each subsystem, one with the technical complexity factor, and one without. The estimates without TCF were a little lower than with TCF, as seen in Table 5.12. The estimates were added up to obtain the total estimate of effort.

Subsystem	Simple use cases	Medium use cases	Complex use cases
1	10	7	2
2-RT	12	9	0
3-RT	3	6	4
4-RT	0	8	2

Table 5.14: Use cases and their complexity in Project B

Subsystem	Use cases	Classes	Estimate	Expert Estimate	Actual effort
1	19	26	3092	3450	3723
2	21	57	5095	2615	3665
3	13	35	3245	3235	3835
4	10	43	3529	3300	2710
Total	63	161	14961	12600	13933

Table 5.15: Estimates produced by 'Optimize'

The expert estimate was 12600 hours. The use case estimate was 14965 staff hours with the TCF, and 14528 staff hours without the TCF. Actual effort was 13933 hours. The expert estimate was lower than the use case estimates, and the actual effort. The results show that omitting the TCF brings the estimates closer to the actual effort, and that the TCF has just a little influence on the estimates.

5.3.13 Estimate produced by 'Optimize:

Input to the tool were the number of use cases and their complexity defined by counting transactions, as in the use case points method, and the classes from the class diagrams for all the subsystems. Estimates were calculated for each of the subsystems, see Table 5.15. Use case complexity for each subsystem is as shown in Table 5.14. There were 25 simple, 30 medium and 8 complex use cases for the whole system.

The total number of use cases and classes and estimates are shown in Table 5.15. The total estimate was 14961 staff hours.

5.3.14 Estimate produced by 'Enterprise Architect'

The estimate produced by 'Enterprise Architect' using 20 staff hours per use case point was 10058 staff hours. This is much lower than all the other estimates. The reason for this is that that staff hours per use case point was set to 20. In the use case points method it was 28 staff hours per use case point. Using this value yields the same estimate as the use case points method: 149615 staff hours.

Effort	Expert est.	UC estimate	UC without TCF	Optimize	EA
13933	12600	14965	14528	14961	14965

Table 5.16: Estimates computed for Project B

5.3.15 Comparing the Estimates

All the estimates are shown in Table 5.16. 'Effort' means actual effort, 'Expert est.' is expert estimate, 'UC estimate' is the use case points method with TCF, 'UC without TCF 'is the use case points method without TCF, and 'EA' means 'Enterprise Architect'. The value obtained in 'Enterprise Architect' is the same as for the use case points method with the TCF when effort per use case point is the same.

The use case points estimate without the TCF, 14528 staff hours, is the closest to the actual effort of 13933 hours. The estimate produced by 'Optimize' is practically the same as the estimate produced with the use case points method with the TCF. the results are 14961 and 14965 staff hours.

The main difficulty with the tool 'Enterprise Architect' is deciding how to assign staff hours per use case point. It is necessary to compare with earlier projects. But often, no such projects exist, and assigning effort per use case point is therefore based on guesswork.

'Optimize' needs classes as input as well as the use cases, and is therefore more suited to making estimates later in the project when more data are available.

5.4 Threats to Validity

The use case points method accounts for all project activities like planning, analysis, design, project management, testing etc. in the total estimates. In the projects of both case studies, project management was not included in the actual effort. In Case Study A, this activity was simply not included when registering effort. In Case Study B, development of the four subsystems was part of a larger project. Project management was therefore registered for the project as a whole, and was not included in the actual development effort for the four subsystems.

The resulting estimates were very accurate in both case studies. But since project management was not counted, these results are somewhat uncertain. If project management had been part of the actual effort, both method and tools would have undersized the systems. However, as there was no information about the effort for project management, it is not possible to say how much the the systems would have been underestimated. The results shown in Table 5.12 were therefore used in the further studies.

The values for the technical and environmental factors were not assigned at the beginning of either project. Setting these scores when a project is finished or well on its way, means having knowledge that is not available at the beginning of a project. It makes the process easier, but ideally, these factors should be evaluated early in the process.

One of the main problems when sizing these systems was defining use case complexity from the use case descriptions. In both case studies, many use cases totally lacked text, others were incomplete. This is most likely a normal situation in industry, where there are no guidelines for use case writing. I have tried to amend these failings in various ways, like counting transactions from sequence diagrams, and number of classes that implement a specific use case. By doing so, I may have made mistakes.

In Case Study A, complexity was assigned by the technical supervisor, who defined degree of reuse as a measure of complexity. This approach is different from the approach described by Karner, so in effect it is a variation or extension of the use case points method. Although the resulting estimate is very close to actual effort, especially when omitting the technical complexity factor, it is a different way of sizing than that which is described in the original method.

Module 2 in subsystem 4 in Case Study B also lacked use case descriptions. Use case complexity was defined by counting the classes that implemented the use cases. Schneider and Winters recommend counting analysis classes [SW98]. However, the only available classes were in the class diagram in the design, and this may mean that too many classes were counted.

The conclusion is that when use cases are not written fully dressed and at a suitable level of detail, sizing is possible, but may be difficult and include uncertain factors.

5.5 Summary

In Case Study A, the results show that the use case points method computed an accurate estimate of effort, and that the estimate was improved when the technical complexity factor was omitted.

In Case Study B, the method and tools estimated the application quite accurately. As in Case Study A, the estimate without the TCF was the most accurate. Counting transactions and defining use case complexity was straightforward for two of the subsystem, whereas sequence diagrams and class diagrams had to be used to count transactions for the other two subsystems.

The results obtained from the two Case Studies A and B underline the importance of writing use case descriptions at an appropriate level of detail if use cases are to be used for estimation purposes. The trouble with sizing use cases that are lacking in textual detail became evident. Sizing becomes time-consuming, and there is a danger that the functionality in not expressed well enough, and that the system will be underestimated.

The estimate produced by the tool 'Optimize' was accurate when all the classes of the class diagrams were used as input. But this is contrary to the guidelines for input to the tool, where only the analysis classes should be used in addition to the use cases. In Case Study A, there were no analysis classes, and the session beans were first counted and used as input. This resulted in a very low estimate. In Case Study B, there was no analysis classes either, and all the design classes were used as input to compute a fairly accurate. This approach is not in accordance with the recommendations of the tool vendor, but was necessary in order to obtain fairly accurate

estimates.

The tool 'Enterprise Architect' computes the same estimate as the use case points method, because input is the same. With the use cases points method, effort per use case point is 20 person hours for project A, due to team experience and stability expressed in the environmental factors of the use cases points method. However, if the measured values for experience and stability are reduced, effort per use case point is increased to 28 hours in the use case points method, according to the suggestion of Schneider and Winters [SW98]. It is difficult to know how to adjust this value in 'Enterprise Architect'. Other projects need to be used as a baseline to be able to define effort per use case point, but such information is often lacking, either because there are no recorded historical project data, or because no similar projects exist. Converting use case points to staff hours in this tool may therefore often be based on guess-work.

Chapter 6

A Study of Students' Projects

This chapter describes the use case points method applied to students' projects. Estimates were computed of ten small projects with the use case points method, and compared to estimates made with 'Optimize'. The tool 'Enterprise Architect' was not used for these studies, as the estimates would be identical to estimates produced with the use case points method. The main goal of the investigations was to determine the appropriate level of detail in use case descriptions written for estimation purposes. Section 6.1 describes the projects and the data collection. Two sets of estimates were made. Section 6.2 shows the first estimates based on use case descriptions that had not been edited. Section 6.3 describes the process of editing the use cases and shows the new estimates. Section 6.4 describes the estimation process with the tool 'Optimize'.

6.1 Background

In order to add to existing experience with use case points method, I have applied the method to several students' projects. The data from these projects are not overly reliable, and the results are therefore not as distinct as the results from the industrial projects, which showed that the use case points method produced fairly accurate estimates. But the students were not experienced in use case modeling and writing, nor had any of them estimated a project before. They also recorded actual effort spent somewhat inaccurately. Still, the results are useful for establishing certain criteria that must be fulfilled in order to be able to use the use case points method successfully, like the suitable level of detail in the use case descriptions, and the influence of use case structure and actor generalisation on estimates.

I have chosen to study a selection of students' projects from the course 'Software Engineering' (in219) at the University of Oslo, Department of Informatics, in the Autumn term of 2000. The results of my investigations accentuate the importance of writing use cases at an appropriate level of detail if the use cases are to be used effectively for estimation purposes. The students wrote use cases very carefully according to guidelines, however, many of the use cases contained far too much detail, and this caused overestimation. On the other hand, some of the use cases were not detailed

enough, and the projects were underestimated.

These projects were in many ways much less complex than real-life projects. The values for the technical factors were therefore quite low, as there were no requirements demanding ease of use, ease of installation and so on. The main focus was on analysis, design and the implementation of some of the most important functionality.

6.1.1 Context

The students worked in project groups consisting of three to five members developing small software systems, using an incremental development process. The systems were of two different categories. One task was to build a questionaire for the Internet, the other a system for swapping shifts between nurses in a hospital ward. Part of the project task was to compute an early estimate of the time needed to build the system. This was done by estimating the necessary time needed for the different activities. The projects groups were also asked to record actual time spent on each activity, and to compare the estimates to the actual effort spent on the project.

Estimates were made using the PERT method, recording values for 'Most Optimistic', 'Most Pessimistic' and 'Most Probable' staff hours, choosing the 'Most Probable' value as the project estimate. The project work was to result in a set of increments and deliverables using object-oriented analysis and design. The documentation consisted of project plans, functional and non-functional requirements, use case models with textual descriptions, domain model, class diagrams, sequence diagrams and final reports. The source code was also available.

Out of the 31 projects on the course, I selected 10 projects to study, 5 of each category. These projects were the most complete, as the groups had registered total effort spent on the projects. Many of the other projects groups had not registered actual effort, so they could not be used in the study.

In the selected projects, the use case models and use cases descriptions were used to assign complexity to the actors and use cases, and to compute estimates using Karner's method in a spreadsheet. The results were compared to estimates produced by the tool 'Optimize'. Input to the tool was the use case model, where complexity was defined by counting use cases transactions, and the number of analysis classes in the domain model.

Two of the project groups, number 1 and 3, succeeded in delivering the 1st increment only, which accounts for the rather low number of hours spent on these projects. The work consisted mainly of planning, analysis, design and coding. There was little testing, project management and other tasks that are present in real projects, so effort per use case point is lower than in an industry project. Karner's suggested effort per use case of 20 hours is therefore far too much in this case.

The default value in the tool 'Enterprise Architect' is 10 staff hours per use case point. I sent an e-mail to the creator of the tool, Geoffrey Sparks, and asked him why this value was chosen. His answer was that this value reflects actual development effort only, not *'project management, extensive system testing, quality assurance, interfacing with the final customer or*

other housekeeping tasks.' This is indeed the case for the students' projects, so I decided to follow the recommendation of Geoffrey Sparks and set effort per use case point to 10 hours.

6.1.2 Data collection

I collected data from project plans, reports, analysis and design documents. I studied the use case models, use case descriptions, class diagrams, sequence diagrams, time sheets, estimates, and source code and defined input to the use case points method and 'Optimize'. The technical and environmental factors were assigned by students who had participated on the projects during the interviews.

Informal interviews were conducted with 10 students, one from each project. Each interview lasted about 20 minutes. The technical and environmental factors were set by one member in each team, through discussions with the author about the meaning of the specific factors and their values. The technical factors were more or less the same for all the projects. The environmental factors varied slightly for team members. But the differences were insignificant, because the project groups were composed of people with different skills. The environmental factors observe the team as a whole, so the differences were evened out.

The students were also asked how they had estimated, and how they had registered actual effort spent on the projects. None of the students had ever estimated a project before, and they were very uncertain of how to do it. As one student admitted:

'We just set some not improbable number as the Most Pessimistic value, and some other number as the Most Optimistic value, and figured out that the Most Probable value would be somewhere in between.'

When asked how they recorded actual effort, some of the students admitted to having based their figures on guess-work. The reason was that they were not used to writing time lists.

The two system types differed in several ways, for instance, the 'Questionaire' (Q) was a distributed system, the 'Shift' (S) system was not. The technical factors and hence the TCF were therefore different for the two types of systems. The technical factors were practically the same for all the projects belonging to one project category, and for the sake of simplicity, I have used the same values for all the estimates for one type of system. The technical factors for the 'Questionaire' (Q) projects are shown in Table 6.1 on the following page, and the factors for the 'Shift' (S) projects are shown in Table 6.2 on page 73. There were several uncertainties connected with understanding what the factors meant and setting the scores, so I decided that using the same values would not make much difference from trying to distinguish between the projects.

The environmental factors were much the same for all the teams, as the level of experience of the teams seen as a whole were more or less the same. On each team, there were at least one or two people who had experience with object-oriented analysis and design, UML, or programming. The same values for the environmental factors are therefore used for all the projects.

The reason for this choice was, apart from the fact that the project

Factor	Description	Weight	Value	Weighted value
T1	Distributed System	2	4	8
T2	Response adjectives	1	2	2
T3	End-user efficiency	1	2	2
T4	Complex processing	1	1	1
T5	Reusable code	1	1	1
T6	Easy to install	0.5	2	1
T7	Easy to use	0.5	4	2
T8	Portable	2	2	4
T9	Easy to change	1	0	0
T10	Concurrent	1	1	1
T11	Security features	1	3	3
T12	Access for third parties	1	0	0
T13	Special training required	1	0	0
SUM				25

Table 6.1: Technical Complexity Factors assigned project 'Questionaire'-'Q'.

teams had more or less the same average level of experience, that there were several uncertainties connected with these factors. It was not clear what all the factors meant and what the scores meant. I therefore decided it was better to use the same values for all the teams, and not introduce more uncertainty into the measured values.

The values for the environmental factors are shown in Table 6.3 on the facing page.

The first estimates were made without making any changes to the use cases. The results were very inaccurate. The use cases were therefore inspected closely and edited before estimates were computed a second time, with different values for use case complexity.

6.2 The Use Case Points Method - First Attempt

The estimates were computed without editing the use cases. Input to the use case points method were the number of actors and use cases and their complexity, and the technical and environmental factors. Person hours per use case point was set to 10.

The use cases were classified in complexity by counting the transactions in the use cases, or use case steps. I first counted these steps without inspecting the level of detail, and without making any changes to the use case descriptions.

Estimates of all the projects were made with and without the TCF, and compared with the actual effort. The results are shown in 6.6. 'UC estimate' means estimate computed with the TCF, 'UC est. without TCF' is the estimate without the TCF, 'Student est.' is the students' estimate, and 'Actual effort' is total effort spent on the project.

The number of actors, their complexity and the computed Unadjusted

Factor	Description	Weight	Value	Weighted value
T1	Distributed System	2	0	0
T2	Response adjectives	1	0	0
T3	End-user efficiency	1	4	4
T4	Complex processing	1	4	4
T5	Reusable code	1	3	3
T6	Easy to install	0.5	3	1.5
T7	Easy to use	0.5	3	1.5
T8	Portable	2	1	2
T9	Easy to change	1	2	2
T10	Concurrent	1	1	1
T11	Security features	1	1	1
T12	Access for third parties	1	0	0
T13	Special training required	1	0	0
SUM				19

Table 6.2: Technical Complexity Factors assigned project 'Shift'- 'S'

Factor	Description	Weight	Value	Extended value
F1	Familiar with RUP	1.5	2	3
F2	Application experience	0.5	2	1
F3	Object-oriented experience	1	2	2
F4	Lead analyst capability	0.5	2	1
F5	Motivation	1	4	4
F6	Stable requirements	2	5	10
F7	Part-time workers	-1	0	0
F8	Difficult programming language	-1	2	-2
SUM				19

Table 6.3: Environmental Factors assigned to students' projects

Project No.	Simple actors	Average actors	Complex actors	UAW
1-Q	0	1	2	8
2-S	0	0	3	9
3-S	0	0	3	9
4-Q	1	0	2	7
5-Q	1	0	2	7
6-S	1	0	2	7
7-Q	1	1	2	9
8-S	0	1	3	11
9-S	1	0	2	7
10-Q	1	1	2	9

Table 6.4: Number of actors and their complexity, first attempt

Project No.	Simple UCs	Average UCs	Complex UCs	UUCW
1-Q	0	6	0	60
2-S	1	7	0	75
3-S	1	6	0	65
4-Q	2	3	1	55
5-Q	1	0	3	50
6-S	0	5	5	125
7-Q	2	3	0	40
8-S	2	3	1	55
9-S	0	4	0	40
10-Q	4	4	0	60

Table 6.5: Number of use cases and their complexity, first attempt

Actor Weights, **UAW** are shown in Table 6.4. The number of use cases, their complexity and computed Unadjusted Use Case Weights, **UUCW** are shown in Table 6.5. The unadjusted actor weights, UAW, plus the unadjusted use case weights, UUCW, is an indication of the size of the application. The Unadjusted Use Case Points, UUCP, are computed by adding up the UAW and UUCP.

Studying the unadjusted use case weights shown in Table 6.5, one can see that project number 6 was more than three times as large as projects 7 and 9, and more than twice as large as project number 10. Although the systems are of different sizes owing to the amount of functionality they contain, these differences may be due to overestimation caused by too detailed use case descriptions.

Total effort spent on the projects was dependent on the the amount of functionality that was implemented. Not all the project groups succeeded in delivering all the modeled functionality.

The estimates shown in Table 6.6 on the facing page are much higher than the students' estimates, and also much higher than the actual effort. Although not all the functionality was implemented in projects number 1 and 3, it is evident that something is wrong with these figures. The

Project No.	UC estimate	UC est. without TCF	Student est.	Actual effort
1-Q	455	536	418	294
2-S	551	672	182	298
3-S	485	614	177	232
4-Q	421	496	335	371
5-Q	490	576	364	420
6-S	866	1096	519	580
7-Q	401	251	132	243
8-S	433	410	635	595
9-S	308	390	630	578
10-Q	469	552	520	490

Table 6.6: Estimates made with the use case points method, first attempt

.

students' data may be somewhat unreliable, but the number of use cases should give an indication of the size of the system. Some of the systems seemed to be sized incorrectly, especially projects 2, 6 and 7. I suspected that the trouble was the assigned use case complexity, because the use case descriptions had been written at a too low level of detail. For instance, project number 6 was overestimated by nearly 50 percent by the use case points method. Practically all the functionality was implemented. Since the use case estimate is computed from the functionality expressed in the use case descriptions, it was quite possible that use case complexity was set too high.

Examining the use case descriptions does indeed show that this is the case. First of all, one of the use cases in this project did not really describe functionality, but course of action the technical manager must take when the system sends an error message. This reduced the number of use cases to 9. Most of the use case descriptions contained steps describing functionality at too low a level, like 'Actor presses button, ' or 'user wishes to terminate.' This lead to counting too many use case steps, and thus too high complexity.

6.2.1 Editing the Use Cases

I went through all the use case descriptions thoroughly, removing use case steps like 'Actor presses button' or whole sequences like

1. System presents ID and password screen
2. User enters ID and password and clicks 'OK'
3. System validates user ID and password
4. System displays Personal Information screen
5. User types in first and last name and clicks 'OK'
6. System validates that the user is a known user

7. [...]

This is a common mistake. The writer has described too much about the user interface, making the use case a into a user manual instead of a requirements document [Coc00].

The same logging on procedure was described in several of the use cases. I decided it would be better to model a high level 'Log on' use case, and set as precondition in the other use cases that the user is logged onto the system. Another way to correct the use case is to describe the intentions of the use without proposing a specific solution [Coc00]. For instance:

1. User accesses system with ID and password
2. System validates user
3. User provides name
4. System validates that the user is a known user
5. [...]

In this way, the six steps were reduced to four.

6.2.2 A Use Case Example

The following example illustrates how a use case that looks OK, is too detailed.

Use Case example

```
Use Case Name:      Answer Questionaire

Primary Actor:      Internet user

Secondary Actors:   none


Pre-Conditions:     The Internet user has received a password
                    The web pages are up and running
Main success
scenario:           1. The Internet user logs onto the web page
                    using password
                    2. The user answers the questions
                    3. The user presses the 'Submit' button
                    4. The information is saved

Post Conditions:    The information is saved to disk or
                    The user is informed that information is
```

```
                    not saved

Alternate Flow:     2a. The password is wrong
                        The user is not logged on
                    3a. The user leaves questions unanswered
                        The answeres are saved
                    4a. The information cannot be saved
                        The user is informed


Used By:            none

Includes UC:        none
------------------------------------------------------------
```

Here, step number 3 ' The user presses the 'Submit' button is superfluous. Instead, there should have been a step 2: 'System validates user', as a use case describes how the user interacts with the system.

Project number 1 seemed to be quite accurately estimated, since not all the functionality was implemented. Projects number 2, 3, 4, 5, 6 and 7 were overestimated because the use cases were written at too low a level of detail. I stripped the use cases of unnecessary detail, and defined use case complexity a second time. In this way, use case complexity for several of the use cases was reduced from 'complex' to 'medium', or from 'medium' to 'simple', some even from 'complex' to 'simple'.

Projects number 8 and 9 were somewhat underestimated due to use cases with too little detail. This represents a different problem than too detailed use cases: One cannot simply add functionality when editing the use cases. Complexity must then be defined from counting analysis classes that implement the use case, or transactions in the sequence diagrams. I used the sequence diagrams as described in Section 5.3.10 on page 59 to define use case complexity for these projects.

Some of the actors were specialisations of others, and I therefore used generalisation to reduce the number of actors.

6.3 The Use Case Points Method - Second Attempt

Having gone through all the projects and corrected the number of actors and use cases as well as use case complexity, I made new estimates of all the projects. Estimates were produced with and without the technical complexity factor. The new estimates are shown in Table 6.9 on page 79. 'UCP' is estimate with the TCF, 'UCP minus TCF' means estimate without the TCF, 'Student est' means Students' estimate, and Functionality means how much functionality was implemented.

For projects 2, 3, 4, 5, 6 and 7 and 9, editing the use cases caused use case complexity to be reduced. In projects 2, 3, and 6 the number of use cases were reduced by one, and in project no. 7, the number of use cases were reduced by two. Project number 7 also had actors that were

Project No.	Simple actors	Average actors	Complex actors	UAW
1	0	1	2	8
2	0	0	3	9
3	0	0	3	9
4	1	0	2	7
5	1	0	2	7
6	1	0	2	7
7	1	0	2	7
8	0	1	3	11
9	1	0	2	7
10	1	1	2	9

Table 6.7: Revised number of actors and their complexity, second attempt

Project No.	Simple UCs	Average UCs	Complex UCs	UUCW
1	0	6	0	60
2	6	1	0	40
3	4	2	0	40
4	2	3	1	55
5	0	4	0	40
6	2	7	0	80
7	1	2	0	25
8	2	3	1	55
9	1	2	1	40
10	4	4	0	60

Table 6.8: Revised number of use cases and their complexity, second attempt

specialisations of another actor. After generalisation, the number of actors were reduced from 4 to 3.

The new number of actors, their complexity and the computed Unadjusted Actor Weights, UAW, are shown in Table 6.7.

The results of editing the use cases and defining complexity are shown in Table 6.8. All the estimates were closer to actual effort after the corrections were made, as shown in Table 6.9. For the projects 2, 4, 6, 7 and 10, editing the use cases brought the estimates very close to the actual effort spent.

6.4 Estimates Produced by 'Optimize'

Input to 'Optimize' was the number of use cases and their complexity, as well as the classes that implemented them, and class size and complexity. I counted all the classes in the class diagrams, because studying the estimates made of the projects in the the case studies indicated that all the classes must be used as input. I used the tool default value 'medium' for

No.	UCP	UCP minus TCF	Student est.	Effort	Functionality
1	455	536	418	294	not all
2	321	407	182	298	all
3	321	407	177	232	not all
4	421	496	335	371	all
5	320	376	364	420	all
6	570	722	519	580	all
7	231	256	132	243	all
8	498	631	635	595	all
9	340	431	630	578	all
10	469	552	520	490	all

Table 6.9: Revised estimates made with the use casepoints method, second attempt

No.	Use cases	Classes	Optimize	UC estimate
1	6	5	929	455
2	8	4	1013	321
3	6	6	989	321
4	6	5	1099	421
5	4	6	982	320
6	10	6	982	570
7	5	7	957	231
8	6	7	894	498
9	4	6	747	340
10	8	6	897	469

Table 6.10: First estimates obtained in the tool 'Optimize'

class complexity, as little is known about the classes at the beginning of a project. The estimates are shown in Table 6.10. They were compared to the estimates made with the use case points method with the TCF, as these were the closest to the actual effort.

The results obtained from the same project data used with Karner's method and the tool 'Optimize' do not correspond at all. The main reason is that estimates produced with the use case points method used a value of 10 person hours per use case point, which is half of what was proposed by Karner. This means that the default value of 90 hour per medium use case in 'Optimize' must be calibrated to fit these projects. But counting use cases and their complexity is not enough. In 'Optimize', analysis classes must also be used as input. Since effort per use case was halved in Karner's method to fit the students' projects, I suggest halving the metric values in 'Optimize' as well. Effort per medium use case was therefore 45 hours, per medium class 30 hours, and per medium subsystem 90 hours.

Using these values as input, with the same use case complexity as in the use case points method, quite different results were obtained. See Table 6.11. These values were compared to estimates computed by the use case

No.	Use cases	Classes	Optimize	UC estimate	Actual effort
1	6	5	464	455	242
2	7	4	541	321	298
3	6	6	452	321	232
4	6	5	549	421	371
5	4	6	390	321	420
6	9	6	496	570	580
7	3	5	332	231	243
8	6	7	521	498	595
9	4	6	392	340	578
10	8	6	527	469	490

Table 6.11: New estimates computed by 'Optimize'

points method with the TCF, and to actual effort. This time, the estimates were comparable. For project number 1 the estimates were practically the same, and for project number 8, the difference was only 23 hours. But all the other projects have differences ranging from around 40 to 130 hours, where 'Optimize' computed the higher estimate. The differences may have to do with assignment of complexity to the use cases. It is not quite clear how 'Optimize' works out total effort in accordance with the define use case size and complexity. But reducing complexity for one use case by one degree in project number 10, from small to trivial, reduces the estimate from 527 hours to 523 hours, a difference of only 4 hours. Reducing one of the use cases from medium to small in the use case points method reduces the estimate (with TCF) from 487 to 451 hours, a difference of 36 hours. Use case complexity alone does not play the same part in 'Optimize' as in the use case points method.

For projects number 5, 8 and 9, estimates made with 'Optimize' were closer to actual effort. These projects were underestimated in the use case points method, maybe due to too little detail in the use case descriptions.

6.5 Threats to Validity

As already pointed out, the data collected from these projects are in many ways less reliable than the data collected from the industrial cases studies described in Chapter 5, because many of the preliminary estimates were made *ad hoc* and actual effort was often recorded inaccurately. It is also possible that some of the results are unreliable because of the adjustments I made to the use cases, knowing what the actual effort was. I have also participated in assigning values to the technical and environmental factors, and may therefore have influenced the students I interviewed and discussed these issues with.

The students were quite good at estimating their own effort, but it may be argued that since many of the project groups admitted that they had not registered hours spent very accurately, the value of the preliminary estimate may have influenced the value of the actual effort. This is known

as 'anchoring and adjustment', where a given or expected value influences human judgment and decision making unconsciously [Plo93].

The students' projects did not include project management, testing and so on, therefore effort was set to 10 staff hours per use case point. This figure may be unreliable, because the students registered time for activities such as studying and group meetings.

6.6 Summary

6.6.1 Establishing the Appropriate Level of Use Case Detail

Writing use cases at a correct level of detail is essential if the use case points method is to be effective. Inspecting and editing the use cases is tedious work and may introduce errors.

The students' use case examples demonstrated the dangers of writing too much detail. For example, modeling twelve simple use cases gives an unadjusted use case weight (UUCW) of 60. 4 complex use cases give the same result. It is a choice of design whether one wishes to put a lot of functionality into one complex use case, or split functionality up into several simple use cases. For estimation purposes it makes no difference, as long as the level of detail is correct. But if the twelve use cases are written with too much detail, they will be defined as more complex than they really are. For instance, if they are assigned medium complexity instead of simple, the system will have an UUCW of 120. Even worse, if the use cases are defined as complex, the UUCW will be 180.

The dangers of detailing too much are obvious: it leads to overestimation. Likewise, too little detail leads to underestimation. Many of the students were not very familiar with UML. Although on the whole the use cases were well written, the overall tendency was to write too much detail. This became apparent after estimates had been computed the first time. I therefore had to edit the use cases to remove unnecessary use case steps, and compute new estimates using the new values. These estimates were closer to the actual effort spent.

Only a few of the project groups used extending and included use cases. Total functionality was therefore sized by counting all the use cases for most of the projects. Only in project number 3 did I remove an included use case. In this project, actual effort was very low, and not counting the included use case reduced the estimate computed both with the use case points method and the tool 'Optimize', and brought the estimates closer to actual effort. This again demonstrates my conclusion presented in Section 3.4.2, that there may be no precise rule for when to count included or extending use cases.

6.6.2 Estimates versus Actual Effort

Some of the use case estimates were fairly accurate when compared to actual effort spent on the project, although most of them differed quite a bit. The relative difference between the preliminary estimates made by the students and actual effort spent on the projects varied a great deal

between the different project groups. However, the students were unsure of the estimation process, not having estimated any projects earlier, and so it is uncertain how reliable these figures really are.

The students took other courses as well as the course in 'Software Engineering', so some of the project groups reported that the preliminary estimated effort was about as much time as they had to spend on project work on this course. The projects with the lowest students' estimates, number 2, 3 and 7, all ended up ended up spending somewhat more time than estimated. Project group number 7 stated in their final report that they had not really understood the task at the beginning, and had underestimated the amount of work to be done.

6.6.3 Omitting the Technical Factors

The results show that there is quite a big difference between estimates with and estimates without TCF. The reason is that the TCF < 1. (0.85 for the 'Q'-projects and 0.79 for the 'S'-projects). The estimates without the TCF are higher than the estimates with the TCF. As already pointed out, these projects are not very complex, and therefore, the TCF is low. However, that is also why they are not very realistic. In real projects, complexity is accounted for when implementing the system. This may lead to counting complexity twice: once in the TCF, and once when implementing the system [Sym91]. In the students' projects, this was not the case, as the systems were not very complex. I therefore assume that for projects low in complexity, including the TCF may bring the estimate closer to the actual effort. But modern software systems are for the most part highly complex, so the problem is not very realistic. I believe that for software projects of today, the TCF does not improve estimate precision, as discussed in Chapter 5.

6.6.4 Comparing Estimates produced by 'Optimize' and the Use Case Points Method

The tool 'Optimize' computed estimates that were higher than those obtained with the use case points method. This may be due to the assigned values for class complexity. The default class value 'medium' was used as input, since little was known about class complexity. This may be a source of error. If the classes were defined as 'small', the estimate for project number 10 would be 505 hours instead of 527. This would be much closer to the use case estimate of 487 hours, and the actual effort of 490 hours. A general problem with the tool 'Optimize' is deciding how to calibrate the metrics. I tried reducing the values for staff hours per use case, class and subsystem by half, because in the use case points method, staff hours per use case point was 10, half of the value proposed by Karner.

As I have already pointed out, these project data are unreliable, and the results may only indicate certain possibilities.

Chapter 7

Evaluating the Results of the Investigations

In this chapter, I analyze the results of the industrial Case Studies A and B and the students' projects.

In Section 7.1, the goals of the case studies are described. Section 7.2 shows estimation accuracy for the estimates of the industrial projects and the students' projects using the *Symmetrical Relative Error* as a measure of accuracy. In Section 7.3, the the effect of discarding the technical factors as a contributor to size is discussed. In Section 7.4 guidelines for setting the environmental factors are defined, and in section 7.5, I discuss the suitable level of detail in use case descriptions written for estimation purposes.

7.1 The Goals of the Investigations

The goals of the case studies and the studies of the students' projects were to the following:

- To investigate the general usefulness of the use case points method,
- to simplify the method by discarding the technical complexity factor,
- to provide guidelines for assigning the values to the environmental factors,
- to define the appropriate level of detail in use case descriptions written for estimation purposes, and to describe alternative ways of sizing the use cases,
- to select a cost estimation method or tool appropriate for the software company where the case studies were conducted, and
- to propose an extension of the use case points method.

Effort	Expert estimate	With TFC	Without TCF	Optimize	EA
10043	7770	10831	10414	10856	10831

Table 7.1: Estimates computed for Project A

Subsyst.	UC Estimate	Without TCF	Optimize	Actual effort
1	4113	3994	3092	3723
2	4062	3920	5095	3665
3	3659	3553	3245	3835
4	3129	3038	3529	2710
Total	14965	14528	14961	13933

Table 7.2: Estimates of the subsystems in Project B

7.2 Estimation Accuracy

In order to determine how accurate the estimates are, traditional software effort accuracy measures the MRE, Magnitude of Relative Error. The formula is: MRE =(Ace-Est)/Act, where Act is actual effort spent on the project, and Est is the estimate. The problem with MRE is that it does not lead to symmetric distribution, because no matter how much the project is underestimated, MRE can never be higher than 1. When overestimating, there is no limit to to the MRE value.

M. Jørgensen and D. Sjøbeg show that the Symmetric Relative Error (SRE) is preferable to the MRE when dealing with large estimation errors [MJ01]. The formulas are

a) SRE = (Act-Est)/ Act, Act =< Est
b) SRE = (Act-Est)/ Est, Act => Est

For Project A in Case Study A, the estimates made with the use case points method, with and without the TCF, and the estimate produced with 'Optimize' were used in the formulas above to compute estimation accuracy.

'Optimize' produced two estimates, one with all the implementing classes, and one with session beans. The estimate obtained with 'Enterprise Architect' was the same as for the use case points method with the TCF. The estimates were all above actual effort, apart from the 'Optimize' estimate with session beans. The results are shown in Table 7.3. 'With TCF' and 'Without TCF' means the use case points method with and without TCF, 'Optimize 1' is the estimate produced by 'Optimize' including all the classes. 'Optimize 2 'is the estimate where only the analysis classes were included. For project A, this meant the session beans. Project B in Case Study B had no analysis classes, so all the design classes were used as input.

For project A, the estimates and actual effort was used in the formulas, and the following SRE obtained:

- The use case points method with TCF: SRE = (10043-10831)/10043 = -0.078

Case Study	With TCF	Without TCF	Optimize1	Optimize2
A	-0.078	-0.037	-0.081	0.814
B	-0.074	-0.043	-0.074	

Table 7.3: Symmetric Relative Error for Projects A and B

- The use case points method without TCF: SRE = (10043-10414)/10043 = -0.037
- Optimize with all classes: SRE = (10043-10856)/10043 = -0.081
- Optimize with session beans: SRE = (10043-5536)/5536 = 0.821

For project B, the same was done. The estimates were above actual effort when Project B was seen as a whole, and not split up into subsystems:

- The use case points method with TCF: SRE = (13933-14965)/13933 = -0.074
- The use case points method without TCF: SRE = (13933-14528)/13933 = -0.043
- Optimize with all classes: SRE = (13933 -14961)/13933 = -0.074

The estimates for the project in Case Study A are shown in Table 7.1. The estimate for 'Enterprise Architect' was the same as for the use case points method. The estimates for Case Study B are shown in Table 7.2. The subsystems were sized and estimated separately. The estimates were above actual effort, except for the estimate produced with 'Optimize' using session beans as input classes, which was below actual effort.

The SRE for both case studies is shown in Table 7.3. The accuracy is approximately the same for both estimates produced with the use cases points method without the TCF. The most accurate estimate was the one produced with 'Optimize' in Case Study B, but this was due to the fact that all the implementing classes were used as input. This is contrary to the tool instructions. The same is true for Case Study A. All the classes had to be used as input, otherwise, the project was grossly underestimated. This again indicates that using analysis classes only is not enough, and that all the design classes should be used as input.

The four subsystems of project B were also studied separately.

- For Subsystem 1, Act>Est for the estimate without the TCF, and Act<Est for the estimate with the TCF.
- For Subsystem 2, Act=>Est for the estimate without the TCF, and Act<Est for the estimate with the TCF.
- For Subsystem 3, Act<Est for estimates with and without the TCF.
- For Subsystem 4, Act<Est for both estimates.
- In the tool 'Optimize', Act<Est for Subsystems 2 and 4, and Act>Est for Subsystems 1 and 3.

Subsystem	With TCF	Without TCF	Optimize
1	-0.026	0.004	0.204
2	-0.029	0.000	0.393
3	0.048	0.079	0.181
4	-0.159	-0.121	-0.221

Table 7.4: Symmetric Relative Error for the subsystems in Project B

Project No.	UCP with TCF	UCP without TCF	Optimize	Effort
2	321	407	541	298
4	421	496	549	371
5	320	376	390	420
6	570	722	519	580
7	231	256	332	243
8	498	631	521	595
9	340	431	392	578
10	469	552	527	490

Table 7.5: Estimates made with Karner's method of students' projects

The SRE is shown in Table 7.4.

For the students' projects, the estimates with the TCF were used, as explained in Section 6.6.3. Only the 8 projects that had implemented all the functionality were used in the analysis. The estimates and actual effort for these projects are shown in 7.5.

For the use case points method with the TCF, the projects with estimates above actual effort are projects number 2 and 4. The projects with estimates below actual effort are 5, 6, 7, 8, 9 and 10.

For the use case points method without the TCF, the projects with estimates above actual effort are projects number 2, 4, 6, 7, 8, and 10. The projects with estimates below actual effort are 5 and 9.

For 'Optimize', the projects with estimates above actual effort are projects number 2, 4, 7 and 10. The projects with estimates below actual effort are projects 5, 6, 8 and 9.

The SRE for the students' projects is shown in Table 7.6. The distribution of the symmetrical relative error for the two projects in the case studies and the eight students' projects are shown in Figures 7.2 and 7.1. For the projects in the case studies, the values of estimates without the TCF were used. For 'Optimize', the estimates with all the design classes as input were used. 'Enterprise Architect' computes the same estimates as the use case points method when staff hours per use case point is the same, so the accuracy for this tool is the same as for the use case points method. In Case Study B, the SRE for each of the subsystems is used.

For both method and tools, the estimates are on the whole very accurate. The use case points method is the most accurate, with 5 estimates out of 13 with an SRE around + - 0.0. The tool 'Optimize' produces accurate estimates when all the classes are used as input, not just the analysis classes. These

Project No.	UCP with TCF	Optimize
2	-0.077	-0.815
4	-0.013	-0.480
5	0.308	-0.366
6	0.018	-0.076
7	0.052	0.077
8	0.194	0.169
9	0.676	0.142
10	0.064	0.474

Table 7.6: Symmetric Relative Error (SRE) for the students' projects

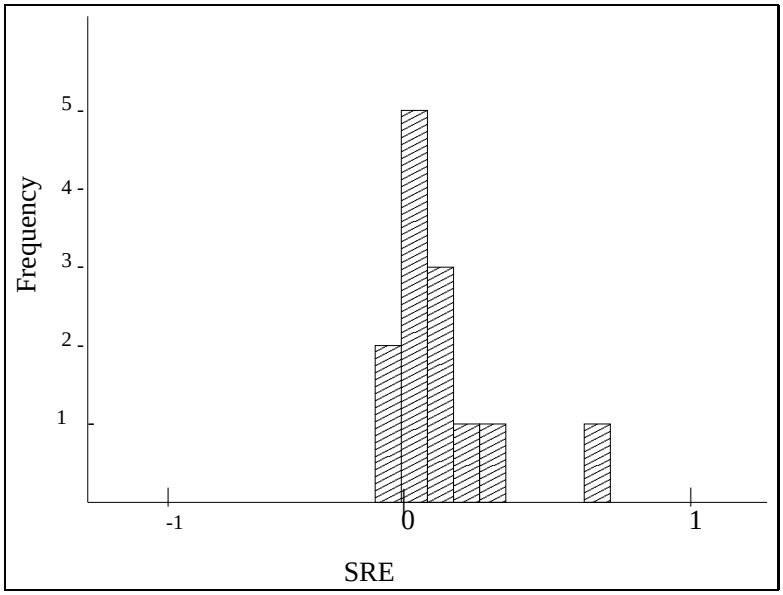

Figure 7.1: SRE distributions for the use case points method

results were used in the feature analysis described in Chapter 9.

7.2.1 Threats to Validity

As discussed in Section 5.4, the estimates produced with method and tools were very accurate when compared to actual effort, but since project management was not included in the actual effort for any of the projects, these results may be somewhat unreliable. However, the resulting estimates were used for computing the SRE since it was impossible to speculate about the total effort for project management. The students' projects did not include project management, testing and so on, therefore the value 10 staff hours per use case point was used for computing total effort. This figure may also be unreliable, because other activities such as studying and group meetings were included in the actual effort.

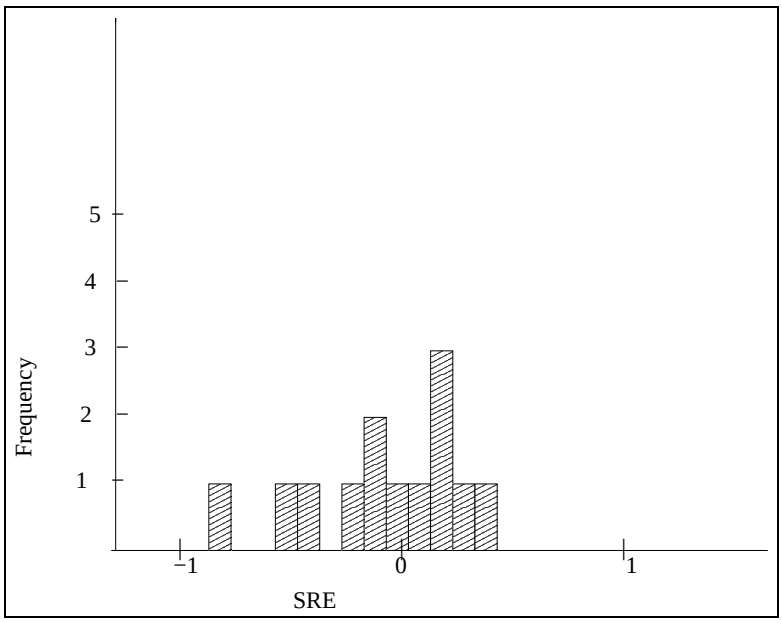

Figure 7.2: SRE distributions for Optimize

7.3 Discarding the Technical Complexity Factor

The main reason for discarding the technical complexity factor in the use case points method as a contributor to software size, is that the technical complexity factor of the MkII Function Point Analysis was dropped a couple of years ago, because it became clear that it was meaningless in modern software development [Sym01]. Use cases correspond to the logical transactions of Mk II FPA. See Section 2.2.2. The use case points method is based on the MKII function points method, and uses several of the of the same technical factors as MkII FPA.

One of the reasons that the technical complexity factor has been discarded in the MK II method, is that early in a project's life-cycle, requirements are often stated in a general form such as 'Ease of use'. At this stage it may seem that such a requirement should be handled by the technical complexity factor. However, by the time the requirements are agreed in detail, much of this 'ease of use' requirement will have influenced the information processing content of the various logical transactions, and so this requirement will be taken into account in the MkII Functional Size. The logical transactions to be counted therefore correspond to the finally agreed set of logical requirements allocated to the software being sized. So including the technical factors in the counts at an early stage may lead to double counting [MkI].

Besides, research has shown that omitting the technical adjustment factors in the estimates obtained with the tradition function points method produces practically the same results as when including them [KK97]. As the adjusted counts do not improve estimates, simple counts may be as ef-

fective as more complex size models when predicting effort. This is useful because the basic counts are known reasonably accurately earlier than the weighted counts. Many function point users restrict themselves to simple counts. Using simple counts improves counting consistency. Research has shown that the difference in counting function points of the same system differed by an average of 12.2 percent. Simple counts reduce this error [KK97].

I therefore assumed that it is possible to drop the technical complexity factors in the use case points method, and I will show in several examples using project data from different companies that this is indeed a possibility.

7.3.1 Estimation Results Obtained without the Technical Complexity Factors

The results from the Case Studies A and B show that the technical complexity factor does not improve estimate precision. Omitting the TCF for the students' projects did not give the same results; the difference between the estimates with and without the technical complexity factor was much larger. However, these projects were not complex, and therefore not very realistic.

To verify that omitting the TCF makes little difference to estimates in real software projects, I have made estimates with and without the TCF using project data from the two projects in Case Studies A and B described in Chapter 5, and three projects described by Bente Anda *et al.* [ADJS01]. These projects are here termed C, D and E.

The research described in [ADJS01] was conducted in a software development company that is situated in Norway, Sweden and Finland. The company has 350 employees, and of these, 180 are located in Norway. The primary business development tasks are solutions for e-commerce and call-centers, in particular within banking and finance. The company uses UML and RUP in most of their development projects, but currently there is neither tool nor methodological support in place to help the estimation process.

The research was conducted in parallel with project C during a period of seven months, while projects D and E were finished before the start of the research. Information was collected about the requirements engineering process and how the expert estimates were produced, and about the use case models and actual development effort.

The data collection for projects A and B is shown in Table 7.7 on the next page, and for projects C, D and E in Table 7.8. Estimates based on use case points had been made for the three development projects described by Bente Anda *et al.*[ADJS01]. These estimates were compared to expert estimates and actual effort, and to the estimates and actual effort from projects A and B.

All the estimates are shown in table 7.9. As seen from these results, there is not much difference between the estimates with and without the TCF, indeed, for projects A and B estimates without the TCF were closer to actual effort. For project D, the use cases estimates with and without

Data Collection		
Data element	**Project A**	**Project B**
Requirements engineering	Effort 10043 hours Very unstable requirements in Phase 1.	Effort 13933 hours. Stable requirements.
Expert estimate	Produced by project manager with 0 years experience using successive calculation	Produced by 4 project teams separately using successive calculation. Team experience from 1 to 7 years.
The use case model	No included or extending use cases. Many small use cases.	Many small use cases. Many included and extending use cases.
The use case estimation process	The technical supervisor counted actors and use cases and assigned values to the technical and environmental factors.	The mentor assigned values to the technical and environmental factors. expert estimate counted use cases

Table 7.7: Data Collection in projects A and B

the TCF are closer to the actual effort than the expert estimate is. This indicates that leaving out the technical complexity factors has little or no influence on the estimates.

7.3.2 Omitting the Environmental Factors

The environmental factors were omitted to see what happened to the estimates. There is a possibility that the unadjusted use case points, UUCP, may be used for early estimating, since they are an indicator of size. To calculate effort, the UUCP are multiplied by 20 staff hours for projects A, C, D and E. One may of course use a different value calibrated to the specific organisation, but for the sake of simplicity, have used the value proposed by Karner. For Project B, 28 staff hours were used, as this was the value in the estimate. The results are shown in table7.10. The estimates produced without the environmental factors are higher than estimates with environmental factors, whether the TCF are included or not. They are not very accurate. Leaving out the environmental factors therefore does not seem to make much sense. These factors must be included in the estimates. But the factors must be specified to avoid the insecurities that often are encountered when people are asked to assign values to them.

7.4 Assigning Values to the Environmental Factors

In order to discuss the usefulness of the environmental factors, it is necessary to look more closely at the content of the specific factors, their

Data Collection			
Data element	**Project C**	**Project D**	**Project E**
Requirements engineering	600 hours spent on requirements specification. Relatively stable requirements.	Effort not available. Changes in requirements	Effort not available Stable requirements.
Expert estimate	Produced by senior developer with 17 years experience.	Produced by senior developer with 7 years experience	Produced by three developers with between 6 months and 9 years experience
The use case model	No included or extending use cases	Many small use cases. Many included and extending use cases. Many included use cases.	Many included and extending use cases.
The use case estimation process	Senior member of the project counted actors and use cases and assigned values to the technical and environmental factors.	The senior developer who had produced the expert estimate counted use cases and actors and assigned values to the technical and environmental factors.	The project manager assigned values to the technical and environmental factors and assigned complexity to actors.

Table 7.8: Data Collection in projects C, D and E

Project	With TCF	Without TCF	Expert Estimate	Actual Effort
A	10831	10414	7000	10043
B	14965	14528	12600	12000
C	2550	2447	2730	3670
D	2730	3038	2340	2860
E	2080	1963	2100	2740

Table 7.9: Impact of TCF on Estimates

Project	With TCF and EF	Without TCF and EF	Actual Effort
A	10831	11120	10043
B	14965	16576	13933
C	2545	2660	3670
D	2730	3100	2860
E	2080	2600	2740

Table 7.10: Impact of ECF on Estimates

weights, and how to set the scores. Assigning values to the environmental factors may be difficult, because there is often no basis for comparison. Setting the scores requires some experience, so the usefulness of the technique is based on having one or two similar projects as a baseline. The environmental factor requires an evaluation of the project team, and people often have difficulties being neutral when asked to evaluate their own work.

Psychological research done by Amos Tversky and Daniel Kahneman has established that people use heuristics or general rules of thumb when making judgments. The advantage is that heuristics reduce time to make reasonably good judgments, and often, heuristics produce fairly good estimates. But in some situations, they lead to systematic biases. Also, expectations can strongly influence perception. When people have experience with a particular situation, they often sees what they expect to see [Plo93].

B.Anda *et al.* describe how two project manager assessed different values to the environment factors regarding experience and capabilities of their teams, although the two projects had similar teams regarding size and experience with software development. They concluded that if project members themselves perform the use case estimation and assign values to the environmental factors, there may be problems with the estimates [ADJS01]. I therefore conclude that there is a need for distinct guidelines for setting these scores.

I believe that if the use of environmental factors are to have any value, it must be made clear what exactly is meant by each of the factors in order to make them measurable. If this is not the case, different analysts will produce different sizes from the same requirements specifications, as pointed out by Bente Anda *et al.* [ADJS01].

The scoring rules for the environmental factors consist of a set of general rules as well as a set of specific rules. To make sure that the values obtained from the environmental factors are valid, it is necessary to be specific about the scoring rules. There was some insecurity when trying to define the values in both Case Studies and students' projects. I therefore propose a specific set of rules for determining the environmental factors. This will simplify the task of setting scores, and make them more concise.

The definitions of the environmental factors and how to set the scores are the result of the discussions with the project manager of the project in Case Study A, the mentor of the project inCase Study B and the students. I noted their observations as described in Sections 5.1.4, 5.2.4 and 6.1.2, and gathered what I decided to be enough information to be able to define special rules for how to set the scores. These scores should be refined for use in specific companies.

7.4.1 General Rules for The Adjustment Factors

The following general rules are described by Symons [Sym91].

- Not present, or no influence if present = 0
- Insignificant influence = 1
- Moderate influence = 2

- Average influence = 3
- Significant influence = 4
- Strong influence, throughout = 5

The scale from 0 to 5 is an ordinal scale, therefore, it is necessary to define the meaning of the different values. From a psychological point of view, it is natural to assume that 'average' means 'middle' or 'median'. However, the intervals on an ordinal scale are not even, but most people do not consider measurement theory when presented with the task of setting scores. It is therefore highly probable that without guidelines, different people will set different scores for the same factor under the same circumstances.

7.4.2 Special Rules for The Environmental Factors

The environmental factors measure team productivity, which means in the current project, not in the last assignment. The problem with this measure is that one has to assess individual experience, and from this determine experience level of the whole team. A project manager may compensate for less skilled team members by assigning mentors with higher skills, as in Case Study B.

Design teams with more than 1 year of experience are, on average, 25 percent more productive than less experienced teams. Productivity does not improve much after approximately 2 years of experience, so the definition of 'experienced' is more than two years of experience with the specific programming language [KLD97].

The meaning of the environmental factors and their scores are the following:

1. **F1: 'Familiar with development method used'.**

   In the original list of factors proposed by Karner, this factor is termed as 'Familiar with RUP' (Rational Unified Process). However, experience from industry shows that although ideally, the projects were supposed to use RUP, in practice this is often not the case. One or more team members may use elements from RUP, the rest may be totally unfamiliar with RUP. On the other hand, the team may use other development processes with which they are already familiar. Therefore, it is better to measure team experience with the method that is actually being used on the project, or scores may be set too low, and estimates will be too high.

   - Score 0: The team is unfamiliar with the development process.
   - Score 1: The team has theoretical knowledge of the development process.
   - Score 2-3: One or more team members have used the method once or a few times.
   - Score 3-4: At least half the team members have experience using the method on different projects.

- Score 5: The whole team has experience using the method on several different projects.

2. **F2 Application experience.**

   This factor indicates experience with the type of application being built, or experience with different types of applications.

   - Score 0: All the team members are novices
   - Score 1-2 : A few of the team have some experience, for instance 1 to 1 1/2 years, the rest are novices.
   - Score 3: All team members have more than 1 1/2 year of experience.
   - Score 4: Most of the team have 2 years experience
   - Score 5: All the team members are experienced

3. **F3 Object-Oriented experience.**

   This factor measures team experience with object-oriented analysis and design (OOAD). Although it contains some of the aspects of the first two factors, it is still significantly different in that it also measures modeling experience, for instance use case modeling in the analysis phase, and class and component modeling in the design phase.

   - Score 0: The team is totally unfamiliar with OOAD
   - Score 1: All the team members have less than one year of experience
   - Score 2-3: All the team members have 1 to 1 1/2 years of experience
   - Score 4: Most of the team members have more than 2 years of experience
   - Score 5: All developers are experienced (more than 2 years).

4. **F4 Lead analyst capability.**

   The factor measures experience with requirements analysis and modeling.

   - Score 0: The lead analyst is a novice
   - Score 1-2: Experience from a few projects
   - Score 3-4: At least 2 years of experience from several projects
   - Score 5: At least 3 years of experience with a variety of projects

5. **T5 Motivation.**

   This factor describes total team motivation.

   - Score 0: Not motivated.
   - Score 1-2 : Little motivation
   - Score 3-4: The team is motivated to do a good job.

- Score 5: The team is very motivated and inspired

6. **T6 Stable requirements.**

   The factor measures the degree of changing requirements and insecurities about the meaning of the requirements.

   - Score 0: Very unstable requirements, constant changes.
   - Score 1-2: Unstable requirements. Customer demands some changes made at various intervals.
   - Score 3-4: Overall stability. Minor changes needed.
   - Score 5 : Stable requirements throughout.

7. **T7 Part-time workers.**

   Having to train new people or not having a stable team influences productivity negatively.

   - Score 0: No part-time workers
   - Score 1-2: A few (20 percent) part-time workers
   - Score 3-4: Half the team are part-time workers
   - Score 5: All the team are part-time workers

8. **T8 Difficult programming language.**

   This factor may indicates experience with the primary development tools as well as the chosen programming language.

   - Score 0: All the team members are experienced programmers
   - Score 1: Most of the team have more than 2 years experience
   - Score 2: All team members have more than 1 1/2 year of experience.
   - Score 3: Most of the team members have more than 1 year of experience
   - Score 4: A few of the team have some experience, for instance 1 year, the rest are novices
   - Score 5: All the team members are novices

7.5 Writing Use Cases for Estimation Purposes

In practice there seems to be much disparity in how different people understand and write use cases. Alistair Cockburn found over 18 different definitions of a use case [Coc97]. Other practitioners have reported finding up to 32 different interpretations [Rul01]. As different individuals apply different rules even within one project, as was the case in Case Study B, the results of applying use cases for sizing software can be ambiguous.

A use case that does not describe the transaction steps properly is not suited for sizing in the way described in the use case points method. Detail

must be added or subtracts in order to obtain an unambiguous description, or use cases must be assigned complexity in some other manner.

In order to write effective use cases for estimation purposes, it is important to describe all the functionality without describing too much or too little detail. Describing the user's movements in operating the system's user interface is a common mistake. If there is a step 'User presses button', the writer has chosen an action that is too small. This leads to counting transactions that do not describe functionality, but the user interface. The description of movements in the user interface belongs to design of the user interface, not to the requirements document [Coc00]. See the students' example in Section 6.2.1.

Sometimes use cases are written as seen by the system looking out at the world and talking to itself. The sentences have the appearance " Get ATM card and PIN number. Deduct amount from account balance." This kind of use case writing was seen in Case Study B, subsystem 4: "Ask for values. Publish according to publication rules." Instead, the use case should be written as seen from the user's view:

- The customer puts in the ATM card and PIN
- The system deducts the amount from the account balance

The use case in subsystem 4 of Case Study B should therefore be written like this:

- The user asks for values
- The system determines the publication rules for the values
- The system publishes the values according to the publication rules

A use case step will describe a goal achieving action, for instance:

- An interaction between two actors ('Customer enters address')
- A validation ('System validates PIN code'
- An internal change ('System deducts amount from balance')

Well-written use cases should have from 3 to 8 steps in the main success scenario. See Section 3.4.2. Alistair Cockburn concludes that if the use case is longer than 10 steps, one has probably included user interface details or written the action steps at too low a level [Coc00]. I therefore propose the general rule of thumb that a use case that has more than ten steps in the main success scenario should be shortened, either by merging steps, or by separating out functionality. One important reason why use cases should not be too long is that the complexity scale in the use case points method has only three levels: simple, medium and complex. This does not account for use cases with a great many transactions. This problem was encountered by Bente Anda *et al.* who felt the need for a complexity scale with more levels in a project that had modeled functionality with many very complex use cases [ADJS01]

See Appendix A for use cases examples.

7.6 Verifying the Appropriate Level of Use Case Detail

In Case Study A, there were no use case descriptions at all, and in Case Study B, one of the subsystems lacked use case descriptions. Use case complexity therefore had to be counted from sequence diagrams for two of the subsystems in case study B. But counting transactions from sequence diagrams may lead to errors. There is an alternative solution to this problem. P. Grant Rule and Charles. A Symons suggests that the faults in the use case descriptions can be counter-balanced by incorporating the MkII Function Point Analysis (FPA) concept of the 'logical transaction' into the use case description [Sym01] [Rul01].

The following use case description is adapted from a real project.

Use Case example

Use Case Name:	UC1.1.1 Display Previous Order List
Primary Actor:	Order System
Secondary Actors:	none
Pre-Conditions:	Customer has requested a list of previous Orders
Main success scenario:	Order System requests an Order collection using enter Customer criteria from Order Information Order Information System returns a collection of Orders Order System displays list of Orders with information to enable a choice to be made
Post Conditions:	List of Orders satisfying selection is displayed on Web Page
Alternate Flow:	none
Activity Diagram:	
Used By:	none
Uses:	The Use Case uses Request Previous Order from Order database (UC1.3.1) and Send

```
                    Previous Order from Order database
                    (UC1.3.2)

Issues:             none

Version Number:     1.0000.
```

[Rul01].

This use case is not written as a list of transactions, and in order to count the use case steps, it must be re-written in some way.

To make sure that the use case is described at a suitable level of detail, one may replace or supplement the main success scenario and alternative flow parts of the use case template by the logical transactions of the MkII method. This approach may be useful if part of the system is described with use cases that are not written with the necessary level of detail, and therefore cannot be used for estimation purposes without some transformation. This may be a safer way than to use sequence diagrams for transaction counting.

For instance, the use case description above can be converted into logical transactions and MkII function points in the following manner:

```
-------------------------------------------------------------
Example use case expressed as measurable logical transactions:
-------------------------------------------------------------
Id :                        1
Stimulus:                   Select function from menu
Input fields:               None                        = 0
Object Class Referenced:    none                        = 0
Response:                   Display empty screen
                            <Select Customer>
Response fields:            None                        = 0
-------------------------------------------------------------

Id :                        2
Stimulus:                   Query Customer Details
Input fields:               Customer_Id                 = 1
Object Class Referenced:    Customer                    = 1
Response:                   Primary route:
                            Display screen
                            <Customer Details>
                            Alternate route:
                            Display screen
                            <Select customer>
Response fields:            Primary route:
                            First_Name
                            Last_Name
                            Organisation
                            Address
```

```
                              Phone
                              Fax                          = 7

                              Alternate route:
                              Error message
-------------------------------------------------------------



Id :                          3
Stimulus:                     Select list orders
Input fields:                 Customer_Id                  = 1
Object Class Referenced:      Customer Order               = 2
Response:                     Display screen
                              <Orders List>

Response fields:              Primary route:
                              Order_id
                              Order_date
                              Product
                              Quantity
                              Order_Status
                                                           = 6
                              Alternate route:
                              Error message
-------------------------------------------------------------
Id :                          4
Stimulus:                     Select specified order
Input fields:                 Order_id                     = 1
Object Class Referenced:      Order product                = 2
Response:                     Display screen
                              <Order Details>
Response fields:              Order_id
                              Order_date
                              Product
                              Quantity
                              Order_Status
                              Product_Description
                              Product_Price
                              Order_Value                  = 8

-------------------------------------------------------------

The results are converted into MkII function points:

Sub-Totals:          3       5         21

MkII Weights:        0.58   1.66      0.26

Contributions:       1.74    8.3      5.46
```

```
Functional Size   = 1.74 + 8.3 + 5.46 = 15.5 MkII fp
-------------------------------------------------------------
```

In the use case example above, the main success scenario consists of a list of activities that are not written as use case steps. Transforming the use case into logical transactions as shown above results in 4 transactions, when counting primary and alternative routes. The use case is therefore of medium complexity, when applying the complexity measures of the use case points method.

Grant Rule and Symons propose using the MkII function point count for estimating the effort needed to be done on this use case. If this approach is chosen, computing a total estimate of effort means using a mixture of two methods, the MkII FPA and the use case points method, which may be inconvenient. However, some experts recommend transforming use case points into function points for computing staff effort, because there is extensive experience with converting function points to estimates of effort [Lon01]. As discussed earlier, the approach of Schneider and Winters can result in very inaccurate estimates under certain conditions. See Section 3.3.

7.7 A Word about Quick Sizing with Use Cases

As described in Section 3.4.2 on page 25, most well-written use cases have 3 to 8 steps [Coc00]. This may be due to limitations in the human brain [Mil56]. I have therefore made the following observations:

If most use cases have 3 to 8 steps, this means that use cases are usually of simple or medium size and complexity. The projects in the case studies both had 63 use cases. Project A had 18 simple, 41 medium and 4 complex use cases. Project B had 25 simple, 30 medium and 8 complex use cases. If people generally limit the length of the use cases to approximately 8 steps, this means that more or less every system is modeled with mostly medium sized use cases, some simple use cases and a few complex use cases. One could therefore get a very rough idea of the size of the system by just counting the use cases. Studying the use cases in the projects described in this thesis shows that there were, roughly speaking, one and a half or twice as many medium sized use cases as simple use cases, and very few complex use cases. The exception was in Case Study B, where there were some more variations. In Subsystems 1 and 2, there were roughly the same number of simple and medium use cases, in Subsystem 3 there were twice as many medium as simple use cases. In addition, there were the same number of complex as simple use cases. In Subsystem 4 there were no simple use cases, but mostly medium. This means that the division between simple and medium use cases in a typical software system could be somewhere in the range of 50/50 to 30/70. Since there are usually few complex use cases, these do not have to be taken into account.

In this way, it is possible to get an idea of the size of the system very early. A system with 60 use cases would for instance have roughly from 30

simple and 30 medium use cases, to 18 simple and 42 medium use cases. This means that the estimate is in the range between

30*5 + 30*10 UUCW = 150 + 300 = 450 UUCW and
18*5 + 42*10 UUCW = 90 + 420 = 510 UUCW.

The unadjusted actor weights, UAW, are added to get the unadjusted use case points, UCP.

This approach is too inaccurate to compute a reliable estimate of effort, but the main idea is to get a rough idea of the size of the future system at a very early stage, especially if the use cases are lacking in textual detail, and it may not yet be clear how they should be sized.

7.8 Summary

As seen from the results presented in Table 7.9, omitting the technical factors does not make much difference to the estimates, it may even improve estimates. It may therefore be possible to discard the TCF as a measure of size. The use case points method could then be simplified by restricting the use of adjustment factors to the environmental factors. These factors play a significant part in the method, and can be used effectively with the proposed guidelines for scoring rules proposed in Section 7.4.

As with MkII function points, some of the technical factors describe requirements that may be accounted for in the use case descriptions. It is therefore a danger that complexity may be counted twice; once when defining use case complexity, and again in the TCF. Adding these figures may produce a too high count for the adjusted use case counts, and this leads to overestimation. Omitting the TCF may eliminate this insecurity.

Omitting the EF made a big difference in Project A, where the estimate without the adjustment factors was less close to actual effort. In Project D, the difference between the estimates and actual effort was not significant. The estimate without the TCF and EF was 140 hours higher, and the estimate with the TCF and EF 130 hours lower than actual effort. In Project E, the project manager had assigned too high values to the environmental factors, which accounts for the difference in estimates [ADJS01]. Omitting the environmental factors in this project brought the estimate much closer to the actual effort, which proves that these factors really were set too high, something which lead to underestimation.

The results show that as opposed to the technical complexity factors, the environmental factors play a significant part in the estimates. Setting the scores too high leads to underestimation.

Use cases must be written at a suitable level of detail if they are to be used effectively for estimation purposes. If descriptions are lacking in detail, and there is doubt as to if all the functionality has been correctly described, the scenarios may be replaced or supplemented with the logical transactions of the MkII function points method. One may then count transactions and use them for defining use case complexity.

In order to get an overall idea of the size of the future system, a 'Quick estimation' approach may be used. This approach is too inaccurate for

producing estimates of effort, but gives a rough idea of the amount of work that must be done.

Chapter 8

Evaluation of Method and Tools

This chapter presents a feature analysis of the use case points method and the tools 'Optimize' and 'Enterprise Architect'. The use case points method is evaluated on the basis of the extension without the TCF, and the guidelines described in Chapter 7. The goal of the feature analysis was to identify which method or tool best fits the requirements of the software company described in Chapter 5, and to select the most appropriate tool or method.

8.1 Determining the Features

The software company needed a method or tool for computing accurate estimates at an early development stage. The requirements for such a method or tool were gathered from interviews with the project manager of Project A in Case Study A. The method or tool must be easy to learn, easy to use, compute reliable estimates quickly, the results must be easy to interpret, and there must be a help manual. The tool should also not be overly expensive.

The features to be evaluated are therefore:

1. Quality of documentation
2. Help manuals
3. Ease of use
4. Estimation accuracy
5. Ease of input
6. Ease of interpretation
7. Training Effort

8. Learning curve

9. Purchasing Costs

If the relative importance of the features can be assessed, this importance assessment can be used as a weighting factor. Kitchenham and Jones suggest the following weights [KJ97]:

- Mandatory features (M):10
- Highly desirable (H): 6
- Desirable (D): 3
- Nice to have (N): 1

I have defined the following scores, using the general rules of Charles Symons described in Section 7.4.1 as guidelines. The scores are on a ordinal scale from 0 to 5, where 0 means 'not present', 1 means insignificant presence, 2 means moderate presence, 3 means average presence, 4 means significant presence and 5 means presence throughout. The scores are added up and compared to the maximum score to see which method or tool is most appropriate.

8.2 Evaluation Profiles

I spent much time experimenting with the tools, using the project data from the Case Studies A and B, and from the students' projects.

I noted the following when I used the method and tools for computing the estimates:

- **The use case points method** can be used with a simple spreadsheet. The method computes accurate estimates for different types of projects. It is quickly learned, there is no training effort. Estimates are computed in a short time, depending on how much time must be spent defining use case complexity. There is no tool other than a spreadsheet, so there are no purchasing costs.

- The tool **Optimize** takes time to learn, as it is composed of five subtools, all with many input parameters. The documentation states that no formal training is necessary, but that is not my observation. Estimates are computed quickly if all the input parameters are known. A nice feature is the possibility to import models from CASE tools, for instance from Rational Rose. Entering all the use cases and classes manually is tedious. But one still has to define size and complexity for each element, using the guidelines described in Chapter 2. Input may therefore be tedious, and defining use case and class complexity is not straightforward. On-line help is available, as well as several white papers for down-loading on the Internet. A quick-tour of the tool can be downloaded from the web pages. The tool is expensive.

- **Enterprise Architect** is primarily a UML modeling tool with an estimation function. The modeling features have not been evaluated, only the estimation function. The underlying estimation model is the use case points method as developed by Karner. The tool therefore computes the same software size as the original use case points method. Staff hours per use case point can be adjusted to obtain an estimate of effort. The advantage of the tool is that it may be used to model and design the software, and compute an early estimate directly from the use case model. However, use case complexity has to be entered manually. But if a different modeling tool is used, all the use cases and actors must be entered manually. This is tedious and time-consuming. The technical and environmental factors must be defined, and cannot be omitted. This presents a problem if one decides to leave the technical factors out, as I recommend. However, other adjustment factors than the default factors can be defined and given weights. The tool is inexpensive.

I defined three sets of features to be examined:

1. The Learnability feature set,
2. The Usability feature set and
3. The Comparability feature set.

8.2.1 The Learnability Feature Sets

In the Learnability feature set, the following features are evaluated:

- Quality of documentation. This includes all the documentation for the method or tool.
- The Learning Curve defines how long it takes to learn to use the tool effectively.
- User manual. This feature describes how useful the user manual is, if there is one.
- On-line help. A nice feature to have.
- Low training effort. This feature registers the need for formal training.

The tools have the following learning characteristics:

- The use case points method is explained in Schneider and Winters [SW98]. Karner's work is difficult to obtain. The documentation is therefore somewhat lacking in detail. The main problem is assigning values to the technical and environmental factors. The method is learned quickly, and is mastered after having tried it out on a couple of smaller projects.
- Optimize has very good documentation on the web-pages. White papers and an online quick tour can be downloaded. A user manual comes with the tool. Learning the tool takes a lot of time. Formal training may be necessary.

Feature	Importance	Weight	Score	Total score
Quality of documentation	H	6	4	24
Learning Curve	H	6	4	24
User manual	H	6	3	18
On-line help	N	1	0	0
Low training effort	D	3	4	12
Total				78

Table 8.1: Evaluation Profile for the Learnability set. The Use case points method

Feature	Importance	Weight	Score	Total score
Quality of documentation	H	6	4	24
Learning Curve	H	6	2	12
User manual	H	6	4	24
On-line help	N	1	0	1
Low training effort	D	3	2	6
Total				67

Table 8.2: Evaluation Profile for the tool 'Optimize'. Learnability features

- Enterprise Architect has a help function in the tool, describing all the features. If one knows the use case points method, it is easy to understand the estimation function of the tool, and which input to use. The main problem is assigning values to the technical and environmental factors. I have only evaluated the estimation function of the tool. For this, no formal training is necessary.

The scores obtained for the Learnability feature set are shown in Tables 8.1, 8.2, and 8.3. The total weighted score is 110. The use case points method obtained 78 out of 110 points, which is 71 percent. Optimize obtained 67 out of 110 points, or 61 percent, and Enterprise Architect obtained 70 out of 110 points, or 64 percent.

Feature	Importance	Weight	Score	Total score
Quality of documentation	H	6	4	24
Learning Curve	H	6	3	18
User manual	H	6	3	18
On-line help	N	1	0	1
Low training effort	D	3	3	9
Total				70

Table 8.3: Evaluation Profile for the tool 'Enterprise Architect'

8.2.2 The Usability Feature Sets

In the Usability feature set, the following features are evaluated:

- Overall ease of use. This feature is scored by evaluating how easy the tool is to use on the whole: finding the correct menus, defining complexity for use cases and classes according to the tool guidelines, and printing out reports or results.
- Ease of input. This feature defines how easy it is to enter actual input to the tool, and how flexible the tool is, for instance if it is possible to omit unnecessary input.
- Calibration ease. This feature describes how easy it is to modify the underlying metrics, both manually as well as understanding how to modify the metrics in a meaningful way.

Deciding the scores for the method and tools is based on the following observations:

- The use case points method is used with a spreadsheet, with which most people are familiar. Defining use case and actor complexity is described by Schneider and Winters [SW98], and the weights for each complexity category are set. Guidelines for the environmental factors are described in Section 7.4.2. This makes for simpler counting rules. Calibrating the value for staff effort per use case point to fit the organisation/project can be done according to the recommendations of Schneider and Winters [SW98], or using the approach I describe in Section 9.2.
- 'Optimize' consists of 5 sub-tools and several menus. Both use cases and classes are defined by size as well as complexity, and each of these categories have scales of 5 steps, ranging from 'tiny' to huge', and 'trivial' to 'complex'. It can be difficult to distinguish between 'several extension points' and 'many extension points' when defining complexity. It is not always clear how one should calibrate the metrics. It is necessary to have some projects as baseline for comparison. It is also difficult to decide how many input classes that are necessary. The tool documentation states that analysis classes should be used, but estimates are often too low when this approach is used. The result of the estimation process can be printed as an Excel-report, or in a text file. Graphs show effort compared to team size and experience. The estimation tool shows the total effort in staff hours and months, with breakdown into separate activities such as planning, analysis, design, build, and testing. All these features are easy to understand and interpret. The tool has an attractive user interface.
- The estimation tool in Enterprise Architect computes estimates by importing the use case model from the modeling part of the tool. If one has modeled in a different tool, it is not possible to import the use cases, and they have to be entered manually. This is tedious. All the technical and environmental factors must be entered manually for each use case. If there are many use cases, like in the projects of Case

Feature	Importance	Weight	Score	Total score
Overall ease of use	M	10	4	40
Ease of input	H	6	4	24
Calibration ease	H	6	4	24
Ease of interpretation	H	6	4	24
Total				112

Table 8.4: Evaluation Profile for the Usability feature set. The Use case points method

Feature	Importance	Weight	Score	Total score
Overall ease of use	H	6	3	18
Ease of input	H	6	3	18
Calibration ease	H	6	3	18
Ease of interpretation	H	6	4	24
Total				78

Table 8.5: Evaluation Profile for the Usability feature set. Optimize

Studies A and B, this is also tedious work. The technical and environmental factors must be included in the estimates. There is no way to omit them. The default value for staff hours per use case points can easily be altered, but there are no guidelines for how to define this value. The results show the total number of unadjusted use case points, adjusted use case points and total effort in staff hours. Effort per category of use case complexity is also shown. Reports can be printed out in Excel files.

The scores for the Usability feature set is shown in Tables 8.4, 8.5, and 8.6. The total weighted score is 140. The use case points method obtained 112 out of 140 points, which is 80 percent. Optimize obtained 78 out of 140 points, or 56 percent, and Enterprise Architect obtained 90 out of 140 points, or 64 percent.

Feature	Importance	Weight	Score	Total score
Overall ease of use	M	10	3	30
Ease of input	H	6	3	18
Calibration ease	H	6	2	24
Ease of interpretation	H	6	4	24
Total				90

Table 8.6: Evaluation Profile for the Usability feature set. Enterprise Architect

8.2.3 The Comparability Feature Sets

In the Comparability feature set, the features are measurable and can be translated into numbers. the following features are evaluated:

- Installation ease. This feature is scored by evaluating how quick it is to install and upgrade the tool. This can be measured by the time it takes to download the tool and install it.
- Estimation accuracy. This feature defines how accurate the estimates are. Input is based on the Symmetric Relative Error (SRE) as shown in section 7.2.
- Low cost. The cost is a number.

Deciding the scores for the method and tools is based on the following observations:

- The use case points method is used with a spreadsheet, and installation and upgrading is therefore not a separate issue, and there are no special costs. The method computes fairly accurate estimates, which is given in total staff hours for the whole project.
- The tool 'Optimize' is easy to install and upgrade, it is downloaded from the Internet in a few minutes with a double ISDN line, but purchasing costs are high. The price is available on the web pages of the tool vendor. The tool computes inaccurate estimates for larger projects unless all the design classes are used as input. The SRE from section 7.2 is based on input to the tool of all the design classes. Using only analysis classes as input computes very inaccurate estimates. However, since this is what the documentation recommends, the tool must receive a low score for 'estimation accuracy'.
- The tool 'Enterprise Architect' is downloaded from the Internet in a couple of minutes with a double ISDN line, and is easy to install and upgrade. It is a relatively new tool, and there may be bugs. I found a bug in an earlier version, and sent an e-mail to the creator of the tool. The bug was fixed in the new version. Estimates are very accurate if the same values for effort per use case point is applied as in the use case points method. However, the problem is deciding this value. There are no guidelines in the tool. The default value of 10 staff hours per use case point accounts for analysis, design and build only. However, the tool is inexpensive. The prices are available on the web pages of the tool vendor.

The scores for the Comparability feature set in Tables 8.7, 8.8, and 8.9. The total weighted score is 50. The use case points method obtained 44 out of 50 points, which is 88 percent. Optimize obtained 17 out of 50 points, or 34 percent, and Enterprise Architect obtained 34 out of 50 points, which is 68 percent.

Feature	Importance	Weight	Score	Total score
Installation ease	N	1	5	5
Estimation accuracy	H	6	4	24
Low cost	D	3	5	15
Total				44

Table 8.7: Evaluation Profile for the Comparability feature set. The Use case points method

Feature	Importance	Weight	Score	Total score
Installation ease	N	1	4	4
Estimation accuracy	H	6	2	12
Low cost	D	3	1	3
Total				17

Table 8.8: Evaluation Profile for the Comparability feature set. Optimize

Feature	Importance	Weight	Score	Total score
Installation ease	N	1	4	4
Estimation accuracy	H	6	3	18
Low cost	D	3	4	12
Total				34

Table 8.9: Evaluation Profile for the Comparability Feature set. Enterprise Architect

Method/tool	Learnability	Usability	Comparability	Total
The UCP method	71	80	88	75
Enterprise Architect	64	64	68	65
Optimize	61	56	34	54

Table 8.10: Evaluation Profile in percentages

8.3 Evaluation

The use case points method received a total of 224 out of 300 points, which is 75 percent. For 'Optimize' the score was 162 out of 300, or 54 percent, and for 'Enterprise Architect' the score was 194 out of 300, or 65 percent. The evaluation scores expressed in percentages of the total weighted scores for all the features and total score are shown in Table 8.10. The results show that the tool 'Optimize' is inferior to the use case points method.

'Optimize' is a pure cost estimation tool, and it requires calibration to a specific organisation to compute fairly reliable estimates. This tool produces estimates that are more inaccurate than the use case points method does with a spreadsheet. It needs the input of analysis classes to compute a fairly acceptable estimate, but if the important issue is to compute an estimate of effort at a very early development stage, for example from the use case model, this tool is not appropriate. It is more suited for computing estimates later in the project, when more information about the design phase is available. Also, the purchasing cost is high.

There is little point in investing in the tool 'Enterprise Architect' if all one needs is a cost estimation tool, as the same estimate can be made with a spreadsheet. But if one is considering a modeling tool, the estimation function is a nice feature. Another advantage is the low cost.

The use case points method received the highest scores in the Evaluation profiles. The method was therefore selected as a cost estimation method to support expert estimation in the software company. It will be used on future projects in order to further investigate estimation accuracy.

Chapter 9

An Extension of the Use Case Points Method

In many development projects, use cases are written in ways that are not ideal for estimation purposes, as described in Chapter 7.

A single application may consist of several sub-systems developed by different teams, all writing use cases in different ways. The effect of this situation was demonstrated in Case Study B. This means that it may be more difficult to size certain parts of a system than other parts. Using the same approach to sizing for all the subsystems may produce inconsistent counts and inaccurate estimates.

But as the use case points method seems to work well for different kinds of applications, it is therefore important to be able to use the method even if the use cases are not written in an ideal way. I therefore propose an extension of the use case points method, with specific guidelines for assigning values to the environmental factors.

9.1 Alternative Counting Rules

9.1.1 Omitting the Technical Complexity Factor

In order to obtain more consistent counts, the counting rules can be simplified by omitting the technical complexity factor. My studies have shown that omitting the technical complexity factor does not make much difference to the estimates, in fact, it often improves estimates.

9.1.2 Alternative Approaches to Assigning Complexity

Counting transactions from well-written use cases is a simple task. Trying to define use case complexity from use cases where the main success scenario is the equivalent of 'do things' is a challenge.

There are several alternative ways of defining complexity, and one may have to combine some of the following approaches to arrive at a correct measure:

1. If there are no textual descriptions or activity diagrams, sequence diagrams can be used to count transactions. This presents a certain danger, because these diagrams describe a lower level of functional decomposition, and counting these transactions may lead to overestimation. One may also count analysis classes as proposed by Schneider and Winters [SW98]. The drawback is that counting classes means that there is some design done, and ideally, estimates should be computed from the requirements specification described by the use cases in the analysis stage.

2. Another approach to defining use case complexity is a combination of the one used in Case Study A, where degree of reuse was used to define complexity, and the guidelines for defining use case complexity described in the tool 'Enterprise Architect':

   - If there is extended design reuse, and the use case has a simple user interface, it is classified as simple.
   - If there is some reuse, and involves more interface design, the use case is of medium complexity.
   - If there is no reuse, and involves a complex user interface, the use case is complex.

3. If there are textual descriptions, and there is any doubt about having described the correct level of detail, this can be verified by transforming the use case descriptions into the logical transactions of the MkII method, as described in chapter 8. The use case transactions can be counted from the primary and alternative flows in the logical transactions. If the use cases are too detailed, they must be edited by removing use case steps that do not describe functionality, but act as a user manual. The drawback is that this is time consuming and demands creative thinking. If the use cases have too little detail, one of the other approaches must be used.

9.1.3 Converting Use Case Points to Staff Hours

The values for staff hours per use case point proposed by Schneider and Winters [SW98] can cause drastic increases in estimates when a factor is varied by one point or even half a point only. Although the figures proposed by [SW98] may be based on experience, they may lead to unexpected results. The project manager on the project in Case Study A observed that small variations made meaningless increases to estimates.

Adding the environmental factors as described in Section 3.4 gives 20 staff hours per UCP if the value is less than 3, 28 if it is 3 or 4, and 36 if it is more than four, but in the last case, Schneider and Winters recommend making changes to the project [SW98]. The leap from 20 to 28 staff hours may be based on experience, but it may also give meaningless results. I therefore propose that this value should be calibrated to the specific organisation.

9.2 Guidelines for Computing Estimates

The rules for sizing an application are therefore the following:

1. Count all the actors, and assign complexity to each according to Karner's method:

   A simple actor represents another system with a defined Application Programming Interface, API, an average actor is another system interacting through a protocol such as TCP/IP, and a complex actor may be a person interacting through a GUI or a Web page. A weighting factor is assigned to each actor type.

   - Actor type: Simple, weighting factor 1
   - Actor type: Average, weighting factor 2
   - Actor type: Complex, weighting factor 3

   Add up the counts to get the Unadjusted Actor Weights, **UAW.**

2. Count all the use cases, and assign complexity to each using either Karner's method, or one or several of the methods described above,

   - Simple: 3 or fewer transactions, weighting factor 5
   - Average: 4 to 7 transactions, weighting factor 10
   - Complex: More than 7 transactions, weighting factor 15

   Add up the counts to get the unadjusted Use Case Weights, **UUCW.**

3. Add these together to obtain the Unadjusted Use Case Points

4. Set the Environmental factors by using the guidelines described in Section 8.2.2, compute the EFactor

5. Use Karner's formula

   EF= 1.4+(-0.03*EFactor)

6. Calculate the *adjusted use case points* using the formula

   UCP = UUCP*EF

7. Convert the total effort into staff hours by using the approach of Schneider an Winters described in Section 9.1.3 to account for team experience, or calibrate the value for staff effort per use case point by using data from past projects in the specific organisation.

Chapter 10

Conclusions and Future Work

In this Chapter, the conclusions of the investigations are presented in Section 10.1, and ideas for future work on estimation with use cases in Section 10.2.

10.1 Conclusions

The analysis of the Case Studies shows that the use case points method and the extension I propose can be used to size different kinds of software.

Estimating software projects with use cases is still in the early stages. The nearly 10 year old use case points method proposed by Gustav Karner has not become popular, and little research has been done to establish the general usefulness of the method. The method has not been updated since its creation, although it has been used in modified forms together with function point counts [AP98].

Karner tested his method on a few small projects only, therefore, it is necessary to try out the method on several larger projects in order to gather more reliable data on estimation results. I have contributed to existing experience with the method by trying it out on the two projects in Case Studies A and B, and ten students' projects.

Research conducted by Bente Anda *et al.* [ADJS01] showed how the use case points method can compute accurate estimates of effort. However, these projects were relatively small, with person effort of approximately 2500 staff hours. I have applied the method to two projects that were approximately four times as large in order to investigate whether the method can be used to size larger projects with real-time functionality.

I have also studied 10 small students' projects. The projects ranged from approximately 300 to 600 staff hours of development effort. The main purpose was to study the use case descriptions, and determine how they can best be written for estimation purposes.

10.1.1 The General Usefulness of the Use Case Points Method

The two projects described in the Case Studies A and B in Chapter 5 differed in several ways. The project in Case Study A was an Internet application.

The project in Case Study B was a real-time system with critical security issues.

One could therefore expect somewhat different results as to the accuracy of the estimates produced with the use case points method, merely due to the difference in project type. However, for both projects, the results proved practically the same as for the studies conducted by Bente Anda *et al.* [ADJS01]. The conclusion is that the use case points method can be used to size different kinds of software of varying size.

The method may therefore be well suited to support expert estimation, especially if the estimator is inexperienced or has little knowledge of the application domain.

10.1.2 Omitting the Technical Complexity Factor

The traditional function points methods like FPA and MkII FPA on which the use case points method is based, have discarded the technical adjustment factor because it does not improve counts. Omitting this factor gives more consistent counts because of simpler counting rules. There are fewer sources of error.

It has also become clear that the technical complexity factor does not measure size, and that there is a danger that complexity is accounted for twice. I therefore assumed that the Technical Complexity Factor could be dropped in the use case points method also. I computed estimates for all the projects described in this thesis: the two projects in the Case Studies, the three projects described by Bente Anda *et al.*, and the 20 students' projects described in Chapter 6. The results showed that there was little difference between estimates produced with the technical complexity factor, and estimates produced without the technical factor. Estimates without the technical factors were also on the whole more accurate. Considering the uncertainties that people encounter when trying to set the values for the different factors, and the fact that the technical complexity factor is a concept created more than twenty years ago and has been discarded by function point users, I conclude that the technical complexity factor may be dropped. Doing so produced more concise counting rules and more dependable counts in the estimates.

10.1.3 Writing Use Cases for Estimation Purposes

One of the main difficulties one encounters when applying the use case points method, is that the writing of use cases is not standardized. Software practitioners write use cases in very different ways and at different levels of detail. Since defining use case complexity is dependent on textual use case descriptions or state or activity diagrams, it is important to agree on a company standard for structuring and writing use cases. Letting software engineers write use cases in their own way increases the number of errors, makes estimating with use cases more difficult, limits the possibility of use case reuse, and decreases the overall efficiency of the project [Ber97].

If there are no company standards for the writing of use cases, different teams may write use cases of very different form and structure for different subsystems of a larger system, as was seen in Case Study B. Sizing these sub-systems therefore introduces uncertainties as to the consistency of the use case counts, and the accuracy of the estimates. When there is not enough detail in the use case descriptions, or textual descriptions are totally lacking, the use case points method may be used only if it is possible to define complexity some other way, for instance by counting analysis classes or sequence diagrams. Other approaches are considering the amount of reuse, and/or the number of interfaces and database entries on which the use case touches as a measurement of complexity.

Unfortunately, few software practitioners have an adequate sense of the proper level of detail that should be associated with a given use case. A survey of the use cases at a given site will most likely reveal that some use cases have so little detail in them that they are unacceptably ambiguous, while others are so detailed that the slightest change to the requirements will cause them to be rewritten [Ber97]. In order to use the use case points method effectively and compute estimates in a short time early in the development phase, the use cases must be written at a suitable level of detail, and be structured in a way that makes it easy to count use case transactions. Otherwise, other sizing approaches must be used. This is time consuming, and may also introduce errors into the counts. I have therefore studied various writing styles, and have given examples of how use cases should be written for use case estimation purposes. See Appendix A. I propose that these examples may be used as general guidelines for defining company standards for use case writing.

Defining and verifying the correct level of textual detail is not straightforward. The minimum level of detail in the use case descriptions must list all the functionality as use case steps, from which one can read number of transactions. If there is doubt as to the appropriate level of detail, the use case may be translated into the logical transactions of the MK II function points method to verify if all the functionality is covered by the use case description. This may be a useful approach if there are difficulties with sizing one of many subsystems in a large application, as in Case Study B. On the other hand, this is time consuming, and should be unnecessary once company guidelines for use case writing have been established, and are followed.

10.1.4 Specification of Environmental Factors

The environmental factors that are used as cost drivers in the use case points method are not always clearly understood. I have therefore specified the meaning of the specific factors, and defined guidelines for specific counting rules in order to obtain more consistent counts.

10.1.5 Evaluation of the Method and Tools

The use case points method is quickly learned and can be applied with the aid of a spreadsheet. An estimate is computed in a short time, maybe

even less than one hour if one is familiar with the method. It has therefore been interesting to compare the accuracy of estimates produced with this method to estimates produced by commercial cost estimation tools. I have conducted a feature analysis in order to select the most appropriate method or tool for the specific software company. The results showed that the use case points method computes more accurate estimates than the tools, and received the highest score in the feature analysis.

The tool 'Enterprise Architect' uses the same input as the use case points method if the TCF is included, and therefore produces the same estimates when staff effort per hour is the same in both cases. However, it can be difficult to determine staff hours per use case point if there are no projects to compare with. If one does not need a UML modeling tool, there are no advantages to this tool.

A difficulty with the tool' Optimize' is that classes must be used as input in addition to use cases. In the tool guidelines, it is stated that analysis classes should be used. My experience is that all the implementing classes from the class diagrams must be used as input to produce an estimate that is fairly close to actual effort. Otherwise, estimates are very inaccurate. This means that some design must be done, and ideally, an estimate should be computed at an even earlier stage, during the analysis phase. The time and effort spent on learning to use the tool, as well as the cost of purchase, makes it less attractive than a simple spreadsheet for computing estimates with the use case points method.

But it is not easy to get software practitioners to adapt new methods and tools [KLD97]. An advantage of the use case points method is that it is based on Function Point Analysis and Mk II Function Point Analysis, which have been in widespread use for years. The method therefore carries weight and may be accepted by companies using the object-oriented approach. The method is not dependent on an expensive tool or training, and the results are easy to interpret.

The results of the feature analysis was presented to the software company in November 2001. The company will investigate the use of the extended use case points method presented in Chapter 9 by applying it to several projects, in conjunction with expert estimates and other cost estimation methods.

10.2 Future Work

The work on this thesis shows that more experience with the use case points method is needed in order to establish the general usefulness of the method. The method should be applied to very large projects developing embedded software, real-time applications, and systems that are rich in algorithmic complexity.

One of the difficulties of the method is converting use case points to staff hours. The approach described by Schneider and Winters can create unreliable estimates. More research is needed to find reliable solutions to this problem.

Appendix A

Use Case Templates

The following use cases template is taken from Alistair Cockburn [Coc00]. This use case was written for requirements purposes, so it is a fully dressed, black-box, system use case. It contains Primary actor, scope, stakeholders and interests, Pre- and post conditions, guarantees , main success scenario and extensions. All these points are not necessary, but a use case should at least contain a primary actor, the goal to be achieved, pre- and post conditions, a main success scenario and extensions.

```
USE CASE EXAMPLE
-------------------------------------------------------
USE CASE 1: BUY STOCKS OVER THE WEB

Primary Actor: Purchaser

Scope: Personal Advisors / Finance package ("PAF")

Goal: Buy stocks

Stakeholders and Interests:
Purchaser - wants to buy stocks, get them added
to the PAF portfolio automatically.
Stock agency - wants full purchase information.

Precondition: User already has PAF open.

Minimal guarantee: Sufficient logging information
that PAF can detect that something went wrong
and can ask the user to provide details.

Success guarantee: Remote web site has
acknowledged the purchase, the logs and
the user's portfolio are updated.
```

```
MAIN SUCCESS SCENARIO:
----------------------
1. User selects to buy stocks over the web.
2. PAF gets name of web site to use from user.
3. PAF opens web connection to the site, retaining
   control.
4. User browses and buys stock from the web site.
5. PAF intercepts responses from the web site,
   and updates the user's portfolio.
6. PAF shows the user the new portfolio standing.

EXTENSIONS:
-----------
2a.  User wants a web site PAF does not support:
2a1. System gets new suggestion from user,
     with option to cancel use case.

3a.  Web failure of any sort during setup:
3a1. System reports failure to user with advice,
     backs up to previous step.
3a2. User either backs out of this use case,
     or tries again.

4a.  Web site does not acknowledge purchase,
     but puts it on delay:
4a1. PAF logs the delay, sets a timer to ask
     the user about the outcome.
4a2. (see use case Update questioned purchase)

5a.  Web site does not return the needed
     information from the purchase:
5a1. PAF logs the lack of information, has the user
     Update questioned purchase.
```

In the main success scenario, there are 6 use case steps, with extension points in steps 2,3,4 and 5.

The following use cases are adapted from the students' projects and modified slightly. They contain a primary actor, a goal, pre- and post conditions, main success scenario and extensions. These are the minimum requirements for a well-written use case.

```
USE CASE EXAMPLE 1- System 'Questionaire'
-----------------------------------------------------

USE CASE 1: Publish Questionaire

Primary Actor: Administrator, System

Goal: To publish a project on the Internet.

Precondition: The questionaire has been saved

MAIN SUCCESS SCENARIO:
----------------------
1. The adminstrator selects 'Publish Questionaire'
2. The system displays the projects
3. The user selects the project in question
4. The system verifies that the project has not
   already been published
5. The system publishes the project and returns the URL
   to the Web page.

EXTENSIONS:
-----------
2a. There are no projects ready for publishing
 2a1. An error message is sent
4a. The project has already been published
 4a1. An error message is returned
```

In the main success scenario, there are 5 use case steps, with extension points in steps 2, and 4.

```
USE CASE EXAMPLE 2 - System 'Swap Shifts'
-----------------------------------------------------------------

Use Case Name: Verify New Shift

Primary Actor: Nurse2

Secondary Actors: System

Pre-Conditions: Nurse1 has requested to swap shifts
with Nurse2

Main success scenario:
----------------------
1. Nurse2 selects 'Swap Shifts'
2. The system displays the request from Nurse1
3. Nurse2 either accepts or rejects the request
4. The system updates the information
5. The system sends a message to Nurse1, confirming
   or rejecting the swap

Extensions:
2a. There are no registered shifts
3a. Nothing is chosen
 3a1. the user is prompted to try again

Post Conditions: The shifts are updated
```

Appendix B

Regression-based Cost Models

A brief presentation of regression based cost models is given here to show the forerunner of the function points methods and the use case points method.

Early cost models used regression techniques. Data were collected from past projects, and by examining relationships among the attribute measures captured, software engineers hypothesized that some factors could be related by an equation. The basic equation was then adjusted by other, secondary cost factors [FP97].

Transforming the linear equation

log E = log a + b log S

yields an exponential relationship of the form:

$$E = aS^b$$

If size were a perfect predictor of effort, every point of the graph would lie on the line of the equation, with a residual error of 0, but in reality, there is a significant residual error.

The next step in regression based modeling is identifying the factors that cause variations between predicted and actual effort. A factor analysis helps identify these parameters. Weighting factors are assigned to these parameters, and they are added to the model as cost drivers. The weights are applied to the right hand side of the effort equation, yielding a model of the form

$$E = aS^bF$$

Barry Boehm's original COCOMO model contained 15 cost drivers, for which Boehm provided relevant multiplier weights.

Bailey and Basili suggest a cost model from your own data, which minimizes the standard error estimate in the data. The approach was tried out

Methodology(METH)	**Complexity(CPLX)**	**Experience(EXP)**
Tree charts	Customer interface cplx	Programmer qualifications
Top-down design	Application cplx	Programmer machine exp
Formal documentation	Program flow cplx	programmer language exp
Chief programmer teams	Complex processing	Programmer language exp
Formal training	Database cplx	Team exp
Design Formalisms	External communication cplx	
Code reading	Customer-initiated program -	
Unit development folders	- design changes	

Table B.1: Technology adjustment factors

on 18 large, similar projects written in FORTRAN [FP97] . The basic effort equation from this data is

$$E = 5.5 + 0.73S^{1.16}$$

A technology factor is then used, based on attributes such as formal training, programmer qualifications programmer application experience etc. The Table B.1 shows the technology adjustment factors for the Bailey-Basili model. Each entry is given a score from 0 (not present) to 5 (very important), judged by the project manager.

Each column is summed, and the sums are used in a least-squares regression to fit the equation

$$\text{Adjusted effort} = \text{a METH} + \text{b CPLX} + \text{c EXP} + \text{d}$$

It is the technique that is transferable, not the model itself [FP97].

Appendix C

Software Measurement

Software measurement has become essential in software engineering. Measurements are made in order to establish whether requirements are consistent and complete, if the design is of high quality, and to compute estimates of effort. A brief presentation of the basics of measurements and measurement theory is therfore included here.

C.1 Measurement and Measurement Theory

Measurement is the process whereby numbers or symbols are assigned to dimensions of entities in such a manner as to describe the dimensions in a meaningful way. An entity may be a thing, an event, a person, an software application or a development process [NCW]. Measurement makes concepts more visible and therefore more understandable and controllable. Measurements are used to describe the attributes of entities. A measure is therefore the number or value assigned to an entity by a mapping from an observed system to a given mathematical system, in order to characterize an attribute [FP97].

Values are also obtained by other means than direct measurement. An **estimate** is a non-measured value. It is an assessment of an attribute value obtained by an estimation method. The estimate may be based on guesswork, for instance by looking at a code listing and guessing the number of lines of code. Otherwise, it may be based on a predictive formula, for instance by counting the number of transactions in the analysis model of a future system, and using that information in a formula to predict effort needed to construct the system [FP97].

Measurement theory consists of rules that define when and how to measure, how to analyze and depict data, and how to tie the results to questions to be investigated [FP97].

C.2 Software Metrics

Software metrics is a term that describes many activities, all involving software measurement. Some of these activities are cost estimation, productivity measures and models, data collection, quality models and measures, structural and complexity measures and evaluation of methods and tools. Much software metric work has lacked the rigour associated with measurement in other engineering diciplines [FP97].

C.3 Measurement Scales

When considering measurement units, one needs to understand the different measurement scale types implied by the particular unit. The most common scale types are nominal, ordinal, interval and ratio:

- **The nominal scale** is a set of categories into which an item is classified. The categories are not ordered. Categories can not be used in formulas.
- **The ordinal scale** is an ordered set of categories. Often used for adjustment factors in cost models based on a fixed set of scale points such as very high, high, average, low, very low. Scale points can not be used in formulas.
- **The interval scale** consists of numerical values where the difference between each consecutive pair of numbers is an equivalent amount, but there is no real zero value. Addition and subtraction are acceptable.
- **The ratio scale** is like the interval scale, but includes absolute zero representing total lack of an attribute. All arithmetic can be meaningfully applied. [FP97] [Kit95].

Understanding scale types makes it possible to determine when statements about measurements make sense. Measures often map attributes to real numbers, and it is natural to want to manipulate these numbers by adding, averaging, performing logarithms and statistical analysis. But analysis is constrained by the scale type, and it is possible to perform only those calculations allowed for the particular scale type [FP97].

Much of the measurement work on this thesis has been done by

- listing categories like use cases, classes, sequence diagrams etc. These categories are on a nominal scale.
- Counting the number of use cases, actors, classes, and transactions and assigning complexity to them. The counts that are computed are on a ratio scale. For instance 4 use cases of medium complexity (score 10) gives the count

  4 * 10 = 40 UUCW (Unadjusted Use Case Weights).

- Assigning values to entities and scores to specific factors, for instance simple, average and complex use cases and classes. These values or scores are on an ordinal scale.

However, calculations that violate the principles of measurement theory are frequently done when to do so proves useful. For instance, some of the formulas used for calculating function points and use case points are meaningless according to measurement theory [KK97]. The formula for computing the adjusted Use Case Points:

UCP = UUCP * TCF * EF

shown in Section 3.2.2, is not consistent with measurement theory, because the counts are on a ratio scale and the scores for the adjustment factors are on an ordinal scale [KK97]. But such formulas are often used in practice.

Bibliography

[ADJS01] Bente Anda, Hege Dreiem, Magne Jørgensen, and Dag Sjøberg. *Estimating Software Development Effort based on Use Cases - Experience from Industry. In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language.* Springer-Verlag. 4th International Conference, Toronto, Canada, October 1-5, 2001, LNCS 218, 2001.

[AP98] Martin Arnold and Peter Pedross. Software size measurement and productivity rating in a large-scale software development department. *Forging New Links. IEEE Comput. Soc, Los Alamitos, CA, USA*, 1998.

[Ber97] Edward W. Berard. Be careful with 'use cases'. *The Object Agency, Inc*, 1997.

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

[CD00] John Cheesman and John Daniels. *UML Components, A simple Process for Specifying Component-based Software.* Addison-Wesley, 2000.

[Coc97] Alistair Cockburn. Structuring use cases with goals. *Humans and Technology*, 1997.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases.* Addison-Wesley, 2000.

[Fac] The Object Factory. *Estimating Software Projects Using Object-Metrix.*

[FAN97] Thomas Fetke, Alan Abran, and Tho-Hau Ngyen. Mapping the oo-jacobsen approach into function point analysis. *The Proceedings of TOOLS*, 23, 1997.

[Fow97] Martin Fowler. *UML Distilled.* Addison-Wesley, 1997.

[FP97] N.E. Fenton and S.L. Pfleeger. *Software Metrics. A Rigorous and Practical Approach.* Cambridge University Press, 1997.

[JCO92] Jacobsen, Christerson, and Overgaard. *Object-oriented Software Engineering: A Use Case-driven Approach.* Addison-Wesley, 1992.

[JEJ95] I. Jacobsen, M. Ericsson, and A. Jacobsen. *The Object Advantage: Business Process Reengineering With Object Technology*. Addison-Wesley, 1995.

[Jon] Capers Jones. What are function points? www.spr.com.

[Kit95] Barbara Kitchenham. Measurement for software process improvement. *C-FP-003 Issue 1 draft B*, 1995.

[Kit98] Barbara Kitchenham. Evaluating software engineering methods and tools part 9: Quantative case study methodology. *Software Engineering*, 23, 1998.

[KJ97] Barbara Kitchenham and Lindsay Jones. Evaluating software engineering methods and tools, part 7: Planning feature analysis evaluation. *Computing and Control Engineering Journal*, 1997.

[KK97] Barbara Kitchenham and K. Känsälä. Inter-item correllation among function points. *National Computing Centre Ltd, UK and VTT, Finland*, 1997.

[KLD97] Barbara Kitchenham, L. Linkman, and D.Law. Desmet: A methodology for evaluating software engineering methods and tools. *Computing and Control Engineering Journal*, 1997.

[KP98] Barbara Kitchenham and Lesley M. Pickard. Evaluating software engineering methods and tools, part 9: Quantitative case study methodology. *Computing and Control Engineering Journal*, 1998.

[Lon01] David Longstreet. Use cases and function points. Copyright Longstreet Consulting Inc. www.softwaremetrics.com, 2001.

[Mil56] George A. Miller. The magical number seven, plus minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63, 1956.

[MJ01] D. Sjoberg M. Jorgensen. A simple effort prediction interval approach. 2001.

[MkI] *The MkII Counting Practices Manual*, volume version 1.3.1. www.gifpa.co.uk/library.

[NCW] Ralph D. Neal, Richard J. Coppins, and H. Roland Weistroffer. The assignment of scale to object-oriented software measurement.

[Plo93] Scott Plous. *The Psychology of Judgement and Decision Making*. McGraw-Hill, Inc, 1993.

[Ric01] Charles Richter. *Designing Flexible Object-Oriented Systems with UML*. Macmillan Technical Publishing, 2001.

[Rul01] P.Grant Rule. Using measures to understand requirements. *Software Measurement Services Ltd*, 2001.

[Ser01] Software Measurement Services. Cosmic ffp. www.gifpa.co.uk, 2001.

[SK] Steve Sparks and Kara Kaspczynski. The art of sizing projects. *Sun World*.

[Smi99] John Smith. The estimation of effort based on use cases. *Rational Software White Paper*, 1999.

[Spe01] The UML Specification. *Version 1.4*. www.omg.org, 2001.

[SW98] Schneider and Winters. *Applying use Cases*. Addison-Wesley, 1998.

[Sym91] C.R Symons. *Software Sizing and Estimating, MKII FPA*. John Wiley and Sons, 1991.

[Sym01] Charles Symons. Come back function point analysis (modernised) - all is forgiven! *Software Measurement Services Ltd*, 2001.