Estimating Software Development Effort based on Use Cases – Experiences from Industry

Bente Anda[1], **Hege Dreiem**[2], **Dag I.K. Sjøberg**[1,3] **and Magne Jørgensen**[1,3]

[1]Department of Informatics
University of Oslo
P.O. Box 1080 Blindern
N-0316 Oslo
NORWAY
{bentea,dagsj,magnej}@ifi.uio.no

[2]Mogul Norway AS
Drammensveien 134
N-0277 Oslo
NORWAY
{hege.dreiem@mogul.com}

[3]Simula Research Laboratory
P.O. Box 1080 Blindern
N-0316 Oslo
NORWAY

Abstract. Use case models are used in object-oriented analysis for capturing and describing the functional requirements of a system. Several methods for estimating software development effort are based on attributes of a use case model. This paper reports the results of three industrial case studies on the application of a method for effort estimation based on *use case points.* The aim of this paper is to provide guidance for other organizations that want to improve their estimation process applying use cases. Our results support existing claims that use cases can be used successfully in estimating software development effort. The results indicate that the guidance provided by the use case points method can support expert knowledge in the estimation process. Our experience is also that the design of the use case models has a strong impact on the estimates.

Keywords Use cases, estimation, industrial experience

1. Introduction

Use case modelling is a popular and widely used technique for capturing and describing the functional requirements of a software system. The designers of UML recommend that developers follow a use case driven development process where the use case model is used as input to design, and as a basis for verification, validation and other forms of testing [11].

A use case model defines the functional scope of the system to be developed. The functional scope subsequently serves as a basis for top-down estimates[1]. A method for using use case models as a basis for estimating software development effort was introduced by Karner [13]. This method is influenced by the function points method and is based on analogous use case points. The use of an adapted version of the use case points method is reported in [3] where it was found that attributes of a use case model are reliable indicators of the size of the resulting functionality. Use case models have also been found well suited as a basis for the estimation and planning of projects in a software improvement project [16]. However, we have been unable to find studies that describe the use case points estimation process in details. This paper describes a pilot study on three system development projects. The aim of this paper is to provide a detailed description of the method used and experiences from applying it.

Our study was conducted in a software development company located in Norway, Sweden and Finland. The company has a total of 350 employees; 180 are located in Norway. Its primary areas of business are solutions for e-commerce and call-centers, in particular within banking and finance. The company uses UML and RUP in most of their software development projects, but currently there is neither tool nor methodological support in place to help the estimation process. The company wishes to improve the process of estimating software development effort. This is the origin of the process improvement initiative reported in this paper.

We compared estimates based on use case points for three development projects with estimates obtained by experts, in this case senior members of the development projects, and actual effort. Our results support findings reported elsewhere [3,13,16] in that use case models may be suitable as a basis for effort estimation models. In addition to supporting other studies, we have experienced that the guidance provided by the use case points method appears to reduce the need for expert knowledge in the estimation process.

UML does not go into details about how the use case model should be structured nor how each use case should be documented [17]. Therefore, use case models can be structured and documented in several alternative ways [19]. An experiment described in [2] indicated that the understandability of a use case model is influenced by its structure,

---

[1] In general, a top-down estimate is produced applying an estimation method to factors believed to influence the effort necessary to implement a system. The estimation method gives the total software development effort, which may then be divided on the different activities in the project according to a given formula. Adding up expected effort for all the activities planned in a project, on the contrary, produces a bottom-up estimate.

and our results show that the structure of the use case model has a strong impact on the precision of the estimates. In particular, we experienced that the following aspects of the structure had an impact:

- the use of generalization between actors[2]
- the use of included and extending use cases[3]
- the level of detail in the use case descriptions

An important prerequisite for applying a use case based estimation method is that the use cases of the system under construction have been identified at a suitable level of detail. The use case model may be structured with a varying number of actors and use cases. These numbers will affect the estimates. The division of the functional requirements into use cases is, however, outside the scope of this paper.

The remainder of this paper is organized as follows. Section 2 gives an overview of the use case points method. Section 3 describes related work and presents alternative methods and tools for estimation based on use cases. Section 4 describes the three development projects that were used as case studies and how data was collected from them. Our results are reported in Section 5. Lessons learned are reported in Section 6. Section 7 discusses threats to the validity of our results. Section 8 concludes and suggests directions for future work.

2. The Use Case Points Method

This section gives a brief overview of the steps in the use case points method as described in [18]. This estimation method requires that it should be possible to count the number of transactions in each use case. A transaction is an event occurring between an actor and the system, the event being performed entirely or not at all.[4] The four steps of the use case points method are as follows:

1. The actors in the use case model are categorized as *simple*, *average* or *complex*. A simple actor represents another system with a defined 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 graphical user interface or a web-page. A weighting factor is assigned to each actor category:

---

[2] Two actors can be generalized into a *superactor* if there is a large description that is common between those two actors.

[3] Common behaviour is factored out in included use cases. Optional sequences of events are separated out in extending use cases [17].

[4] Appendix A shows a use case from one of the development projects used in this study. The basic flow of events in the use case consists of 7 transactions. The use case is documented according to a template used throughout the company. The template ressembles those recommended in [6].

- Simple: Weighting factor 1
- Average: Weighting factor 2
- Complex: Weighting factor 3

The total *unadjusted actor weight* (*UAW*) is calculated counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the products.

2. The use cases are also categorized as *simple*, *average* or *complex*, depending on the number of transactions, including the transactions in alternative flows. Included or extending use cases are not considered. A simple use case has 3 or fewer transactions; an average use case has 4 to 7 transactions; and a complex use case has more than 7 transactions. A weighting factor is assigned to each use case category:
   - Simple: Weighting factor 5
   - Average: Weighting factor 10
   - Complex: Weighting factor 15

   The *unadjusted use case weights (UUCW)* is calculated counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The UAW is added to the UUCW to get the *unadjusted use case points* (*UUPC*).

3. The use case points are adjusted based on the values assigned to a number of technical factors (Table 1) and environmental factors (Table 2).

Table 1. Technical complexity factors

Factor	Description	Wght
T1	Distributed system	2
T2	Response or throughput performance objectives	2
T3	End-user efficiency	1
T4	Complex internal 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	Includes security features	1
T12	Provides access for third parties	1
T13	Special user training facilities are required	1

Table 2. Environmental factors

Factor	Description	Wght
F1	Familiar with Rational Unified Process	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	-1

Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means the factor is irrelevant for this project; 5 means it is essential.

The Technical Factor (TCF) is calculated multiplying the value of each factor (T1 – T13) in Table 1 by its weight and then adding all these numbers to get the sum called the *TFactor*. Finally, the following formula is applied:

$$TCF = 0.6 + (.01*TFactor)$$

The Environmental Factor (EF) is calculated accordingly by multiplying the value of each factor (F1 – F8) in Table 2 by its weight and adding all the products to get the sum called the *Efactor*. The formula below is applied:

$$EF = 1.4+(-0.03*EFactor)$$

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

$$UCP = UUCP*TCF*EF$$

4. Karner [13] proposed a factor of 20 staff hours per use case point for a project estimate, while Sparks states that field experience has shown that effort can range from 15 to 30 hours per use case point [21]. Schneider and Winters recommend that the environmental factors should determine the number of staff hours per use case point

[18]. The number of factors in F1 through F6 that are below 3 are counted and added to the number of factors in F7 through F8 that are above 3. If the total is 2 or less, use 20 staff hours per UCP; if the total is 3 or 4, use 28 staff hours per UCP. If the number exceeds 4, they recommend that changes should be made to the project so the number can be adjusted. Another possibility is to increase the number of staff hours to 36 per use case point.

3. Related Work

This section reports two experiences with estimation based on use case points. Two alternative methods and one tool for estimation based on use cases are described. Finally, use case points are compared to function points.

3.1 Reported Experiences with Estimation Based on Use Cases

Arnold and Pedross reported experiences from using use case points to measure the size of 23 large-scale software systems [3]. Their method for counting use case points was inspired by, but not identical to, Karner's method. Their experience was that the use case points method is a reliable indicator of the size of the delivered functionality. However, they observed that the analyzed use case models differed much in the degree of details and believed that the measured size may have differed according to this degree. They also found that free textual use case descriptions were insufficient to measure the software size.

Martinsen and Groven reported a software process improvement experiment aimed at improving the estimation process using a use case model in estimating a pilot project [16]. Before the improvement project, the requirements specification was only loosely coupled with the effort and cost estimates. The requirement specification was written in natural language, which was found too informal to be a good basis for the necessary revision of the cost estimate or for restricting the implementation within the cost estimate. Adopting use case modelling, the customer and developers had a common, documented understanding of the requirements. The pilot project experienced an overrun on the estimates, but the overrun was smaller than the average for previous, similar projects. Hence, they found use cases useful as a basis for estimation and planning.

3.2 Methods and Tools for Use Case Estimation

Alternative methods for estimation based on use cases are described in [7] and [20]. In [7] the use case model is a basis for counting function points, which in turn may be used to obtain an estimate of effort. In [20] the use case model is used to estimate the number of

lines of code (LOC) in the finished system. This number of LOC is subsequently used as the basis for an estimate.

These two methods appear more complex than the one we have used as they respectively make assumptions on the relationship between use cases and function points, and between use cases and the number of LOC in the finished system. These assumptions have not been tested. The advantage of these methods, however, is that they may exploit the extensive experience with estimation using function points or lines of code.

Optimize [22] is a tool that provides estimates based on use case models. Optimize measures the size of the problem counting and classifying scope elements in a project. The set of use cases in the project's use case model is one kind of scope element. Other possibilities are, for example, the project's classes, components and web-pages. Qualifiers are applied to each scope element. The complexity qualifier defines each scope element as simple or complex. The tool provides a set of default metrics, extrapolated from experience on more than 100 projects. The user can also customize metric data to produce estimates calibrated for an organization. Optimize organizes the scope elements and metric data to compute an estimate of effort and cost. We intend to evaluate this tool more thoroughly. So, far we have only tried it briefly on data from one of the development projects. Our impression is that the tool requires calibration to the particular organization to provide a reasonable estimate. Moreover, the cost of purchase and training makes it less accessible than the method with associated spreadsheet that we have used.

3.3 Use Case Points and Function Points

The number of function points measures the size of a software application in terms of its user required functionality [1]. Although the calculation of use case points has been strongly influenced by function points, there are several important differences leading to different strengths and weaknesses:

- The function point standards do not require that the input documents follow a particular notation. Use case points are based on the use case model. This means that it is easier to develop estimation tools that automatically count use case points; the counting is based on available documents (use case models). This is an important difference, since counting function points frequently requires much effort and skill.
- There are international standards on how to count function points. The concept of use case points, on the other hand, has not yet reached the level of standardization. Without a standard describing the appropriate level of detail in the requirement description, i.e., the use case model, there may be very large differences in how different individuals and organizations count use case points. Hence, it may currently be difficult to compare use case point values between companies. As reported in [12;14], even with a counting standard there may be significant differences in how people count function points.

4. Data Collection

Table 3 shows some characteristics of the three software development projects used in our case studies.

Table 3. Characteristics of three software development projects

Characteristic	Project A	Project B	Project C
Size	7 months elapsed time, 4000 staff hours	3 months elapsed time, 3000 staff hours	4 months elapsed time, 3000 staff hours
Software architecture	Three-tier, established before the project	Three-tier, known, but not established in advance	As project B
Programming environment	Java (Visual Café and JBuilder), Web Logic	MS Visual Studio	Java (Jbuilder), Web Logic
Project members	6 developers with 0 to 17 years experience	6 developers with 0 to 12 years experience	5 developers with 2 to 10 years experience, 4 consultants were involved part time.
Application domain	Finance	CRM (Customer relationship management within banking), part of a larger solution	Banking (support for sale of credit cards)

Our research project was conducted in parallel with project A during a period of seven months. Projects B and C, on the other hand, were finished before the start of our research. We collected information about the requirements engineering process and about how the expert estimates were produced. We also collected information about the use case models and actual development effort.

Data from project A was collected from the project documents, i.e., the use case model, iteration plan and spreadsheets with estimates and effort, and from several interviews with project members. Data from project B was collected from project documents, and from e-mail communication with people who had participated in the project. In this project the available documentation consisted of a detailed requirements specification with several use case diagrams and textual descriptions of use cases, project plan and time sheets recording the hours worked on the project. Data from project C was collected from project documents, including a requirements specification with brief textual descriptions of each use case, a use case model in Rational Rose with sequence diagrams for each use case, project plan and initial estimates, and from an interview with two of the project members. The collected data is shown in Table 4.

Table 4. Data collection in the three development projects

Data element	Project A	Project B	Project C
Requirements engineering	600 hours spent on requirements specification. Relatively stable throughout the project.	Effort not available. Some serious changes in the requirements during the project.	Effort not available. Stable requirements throughout the project.
Expert estimate	Produced by a senior developer with 17 years experience. The estimation process was influenced by the function points method; effort was estimated per screen.	Produced by a 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. Example of a use case in Appendix A. The customer reviewed the use case model and read through the use cases.	Included many small use cases (containing only 1 or 2 transactions). Contained many included and extending use cases as and a large number of actors.	Contained many included and extending use cases. Each use case was described by a brief textual description and a sequence diagram.
The use case estimation process	A senior member of the project team counted and assessed actors and use cases and assigned values to the technical and environmental factors. Values were inserted into a spreadsheet to produce an estimate. The estimation process took approximately one hour when the use case model was completed and well understood by the person performing the estimation.	The senior developer who had produced the initial expert estimate counted and assessed actors and use cases and assigned values to the technical and environmental factors. An alternative estimate was produced by the first author counting and assessing actors and use cases based on the textual requirements documents. A spreadsheet was used in the estimation.[1]	The project manager assigned values to the technical and environmental factors and also assessed the complexity of each actor. The first author counted use cases from the requirements document and a Rational Rose Model and assessed their complexity. A spreadsheet was used to produce an estimate.
Time sheets	Actual effort was computed from time sheets. The time sheets were structured to enable registering effort on each use case.	Hours were recorded according to some predefined activities. Actual effort was calculated adding up all the activities in the project.	As project B

[1]Two different estimates were produced due to different interpretation on how to count actors and usecases.

5. Results

The results are shown in Table 5. Despite of no customisation of the method to this particular company, the use case estimates are fairly close to the estimates produced by the experts.

Table 5. Expert estimate, use case estimate and effort (in hours)

Project	Expert estimate	Use case estimate		Actual effort
A	2730	2550		3670
B	2340	3320	2730	2860
C	2100	2080		2740

In projects A and C, the use case estimate ended up only slightly below the expert estimate but a bit below the actual effort. The use case estimate for project B is close to actual effort and somewhat higher than the expert estimate.

The first use case based estimate for project B (3320) was produced by the authors with information about actors and use cases given by the senior developer in the project. This estimate was very much higher than the original expert estimate, and it was also higher than the actual effort. We believe that this is because trivial actors were counted, such as printer and fax, and also included and extending use cases. We therefore decided to calculate a second estimate where actors and use cases were counted from the use case model. The actors that provided input to the system or received output from it were generalized into two superactors. Only those two were counted, not the individual actors. This reduced the number of actors from 13 to 6. The included and extending use cases were omitted. We used the same technical and environmental factors in the second estimate as in the first estimate. This resulted in an estimate of 2730 hours, which is very close to the actual effort on the project (2860 hours).

One reason why the use case estimate for project C ended up a bit below the actual effort may be that the project manager assigned too high values to the environmental factors regarding experience and capabilities of the team. For example, he assigned higher values than did the project manager of project B even though the two projects were conducted with similar teams regarding size and experience with software development. Using the same environmental factors as for project B, project C would get a use case estimate of 2597 hours which is very much closer to the actual effort.

The use case estimates for projects B and C were made after the completion of the projects. It was therefore easier to assess values for the technical and environmental factors than in a normal situation because the choice of values could be based on experience with the actual project. This indicates that the technical and environmental factors in the method are appropriate for this company, although there may be a need for some, but not extensive adjustments.

We consider these results promising for the use case points method. The expert estimates were produced by very competent senior developers with good knowledge of

both the technology and the application domain. The results were obtained without any particular calibraton of the method, so it is likely that the use case estimates can be improved. Independent of the method used for estimation, we must expect inaccuracies. Boehm states that these inaccuracies range up to 60 percent or more during the requirements phase [4]. The use case estimate for project A, the project with the largest difference, is 30 percent below actual effort.

Table 5 indicates a relationship between the use case estimate for a project and the effort needed to implement it. Based on this, we would expect a relationship between the size of each use case, measured in number of transactions and the actual effort on implementing the use case. The result of investigating this relationship is shown in Table 6.

Table 6. Size and effort for each use case in project A

Use case	Number of transactions	Use case points	Deve-loper	Iteration	Expert estimate	Actual effort
1. Fetch application	16	15	A	0 and 1	42 h	224 h
2. Simulate application	22	15	B	1 and 2	64 h	301 h
3. Automatic scoring	11	15	C	1 and 2	86 h	267 h
4. Change application	13	15	C	2	124 h	144 h
5. Assess credit-worthiness	31	15		2	170 h	
6. Produce documents	7	10	B + D	1 and 3	152 h	122 h
7. Register new application	14	15	All	2 and 3	936 h	647 h
8. Notification of application	5	10		3	132 h	
9. Transfer application to new responsible	9	15		3	82 h	

For each use case, Table 6 shows the number of transcations, the number of use case points, the developer (anonymized), in which iteration the use case was developed, the expert estimate and the actual effort. The system was developed in four iterations, in the first iteration (iteration 0) the architecture was established and in the subsequent iterations the system was constructed. The realization of the use cases was divided on these iterations.

The functionality described in use cases 5, 7, 8 and 9 was realized as one unit. The total corresponding effort was registered on use case 7. Hence, six cells in the table are empty.

The use cases contain between 5 and 31 transactions. The number of use case points for each use case is calculated according to the description in Section 2. Use cases 1 through 4 were implemented by a single developer; use case 6 was implemented by two developers; and all the developers participated on use case 7. For most of the use cases, the basic flow was implemented in one iteration. Those use cases were then completed by the implementation of alternative flows in a later iteration. Originally, the expert had

estimated effort per screen, but the screens were associated with use cases, so he managed to re-organize the estimates in order to show expected effort per use case. Actual effort shows effort on analysis, design and implementation for each use case and was calculated from the time sheets.

The sum of effort registered on the use cases is smaller than the total effort registered on the project because much of the effort was registered on activities that were not related to a particular use case.

Table 6 shows no relationship between the size of each use case measured as the number of transcations and the effort necessary to implement it. Possible reasons are:

- Estimates based only on the number of use cases, or on a division into simple and complex use cases, are equally precise to counting all the transactions. One way of investigating this further is to compare the size of each use case with the number of classes or lines of code necessary to implement it.
- Many factors influenced the registered effort for each use case, for example:
  - There were different levels of experience among the members of the project.
  - The use cases implemented in the last iterations could reuse design and code.
  - The structure of the time sheets was new to the project members, and they did experience some difficulties in registering effort exactly as intended.

With relatively few data, we are unable to correct for these confounding factors. Therefore, we need more data to further investigate the relationship between the size of a use case and the required effort to implement it.

6. Lessons Learned

This section presents a number of lessons learned from applying the use case points method.

6.1 The Impact of the Structure of a Use Case Model

When we applied the use case points method to the three projects, we experienced that the following aspects of the structure of a use case model had an impact on the estimates:

- The use of generalization between actors. The number of actors in a use case model affects the estimate. Our experience is that if the descriptions of two or more actors have a lot in common, the precision of the estimate is increased by generalizing the actors into a superactor and hence counting the actors only once.
- The use of included and extending use cases. Karner recommends that included and extending use cases should not be counted. Omitting such use cases for project B resulted in an estimate that was closer to the expert estimate and to the actual effort than if they were included. However, in project C we found it necessary to count the

included and extended use cases as very much of the essential functionality was described using these constructs. In our opinion there is definitely a need to investigate this further. Separating out functionality in included and extending use cases reduces the number of transactions in the use cases from which the functionality is separated out, hence the estimate is reduced, but the functionality will still have to be implemented. Optional functionality can be described either in an extending use case or in an alternative flow of events. This complicates the estimation process, because choosing an extending use case will result in a lower estimate than if an alternative flow is chosen.

- The level of details in the use case descriptions. The size of each use case is measured as the number of transactions. We experienced the following difficulties when counting transactions for each use case.
  - Almost all the use cases in project A were classified as complex. We believe that this indicates that the classification of complexity of the use cases should also include an alternative for very complex use cases, for example use cases with 15 or more transactions.
  - It is a challenge to decide the appropriate level of detail in each transaction when structuring a use case, as there are no metrics established to determine correct level of detail for all projects [15]. The level of detail in each transaction will affect the number of transactions, which subsequently has an impact on the estimate obtained with the use case points method.

6.2 Assigning Values to Technical and Environmental Factors

We experienced difficulties when trying to assign values to the technical and environmental factors because we lacked a basis for comparison. In some cases we had to guess what was meant by each factor and we had to try to recapture other projects with which this project could be compared. A particular problem here is that the environmental factors require an evaluation of the competency of the project team. People often have difficulties being neutral when they are asked to evaluate their own work. This may lead to problems if the project members themselves perform the use case estimation and have to assign values to the environmental factors. With more experience from using the method, it will be possible to reuse experiences from earlier projects and calibrate the method to fit the organization. However, this will require that use case models for different projects are structured with a consistent level of detail.

The choice of productivity rate for each use case point may also need calibration. We used a productivity rate of 20 staff hours pr. use case point, but we believe that this choice will depend on whether some activities are estimated outside the use case point method or not.

6.3 Time Sheets

We believe that the entities used in the estimation should correspond to the structure of the time sheets to enable feedback on the precision of the estimates to gradually improve the estimation process. In project A, the time sheets were organized as a table with five columns:

Activity, Use case, Functionality, Name of developer, Hours

Examples of activities are analysis and design, implementation and administration. There was, however, effort in the project that the developers found difficult to register on one particular use case, for example effort related to establishing the database and the client framework. It was some disagreement as to whether this effort should be registered on the first use case that was implemented or as a separate activity not related to a use case. The developers decided to do the latter.

We included all the activities performed after the use case model was completed in the actual effort. Some of the activities were untypical; that is, they would usually not be included in the organization's software development projects. Examples are effort on upgrading to a new version of a development tool and training in the development method and tools for new project members. We decided to include untypical activities also, because we believe that every project is special somehow. In our opinion, if the application of the use case model shall result in a complete top-down estimate, it must produce an estimate with some surplus time for unexpected activities.

6.4 Using Use Case Estimates

In our opinion, use case estimates should not substitute expert estimates. It seems sensible to combine models and human judgement [5]. We therefore believe that use case estimates can be used successfully in conjunction with expert estimates.

For estimators with little experience, the use case points method gives good support for estimation. Estimators may also find it useful to compare the unadjusted use case points for a system with unadjusted use case points from previous projects. Expert estimators may be strongly influenced by, for example, previous estimates or what the estimators believe will be the price to win [9;10], which in turn may result in large deviations from actual effort. The use of function points together with expert estimates has reduced such large deviations [8]. Use case points may be used to obtain the same effect. We also believe that the customers may more easily accept estimates if they know that an established method have been used to produce them.

7. Threats to Validity

In project A, the use case estimate and the expert estimate were produced in parallel. Hence, some of the information used in the expert estimate may have been reused in the use case estimate because of communication between the project members involved in producing the two estimates. We do not know whether this influence had any impact on the estimates.

Project B and C were finished before the start of the research project, so the actual effort was known when the use case estimates were produced. However, the project members who provided information to the estimation method were unable to make use of the information about the actual effort because they did not know the formula behind the use case estimate. Therefore, we believe that the actual effort had no impact on the use case based estimate.

In Project C there were no detailed textual descriptions of the use cases so the number of transactions in each use case was counted from sequence diagrams. Sequence diagrams typically describe the functionality at a lower level of detail than the textual use cases so there will usually not be equally many functions in a sequence diagram as there are transactions in the corresponding use case description. This means that the use case estimate for project C might have been slightly different if it had been produced from use case descriptions instead. After considering the actual sequence diagrams, we do not believe that this had a serious effect on the estimate in this case.

8. Conclusions and Future Work

We conducted three case studies on applying a method for estimating software development effort based on use cases, the use case points method. The results indicate that this method can be used successfully since the use case estimates were close to the expert estimates in our three case studies. In one case it was also very close to the actual effort. It is therefore our impression that the method may support expert knowledge. We intend to further study the precision of the use case point method compared with expert estimates. In our three projects the experts had much experience from similar projects. We will therefore conduct a study where the estimators have different levels of experience.

Moreover, our experience is that applying the use case point method in practice is not straightforward. For example, the choice of structure for the use case model has an impact on the estimates. There is consequently a need for further studies on the precision of the estimates when using the use case points method in different types of projects.

We also believe that it would be useful to investigate how the use case points method, which provides top-down estimates based on a measure of size, can be combined with other methods that provide bottom-up estimates. The purpose of using the estimation method investigated in this paper is to provide a complete estimate for all the activities in the project. Nevertheless, we believe that some of the activities in a development project

do not depend on size or use case points, for example, training and establishing a new programming environment. Therefore, such activities should be estimated in alternative ways and then be added to the use case estimate to provide a final estimate.

Another direction we intend to pursue is comparing the different methods for use case estimation described in Section 3 with regards to precision of the estimates and the effort needed to produce them. The use case points method requires use cases to be described at a level of detail where each transaction is specified. This is not always the case in practice. We therefore believe it is useful to investigate whether other methods for use case estimation are suitable for use case models with less detail. The way use case models are described in a company should guide the choice of method for use case based estimation, or vice versa, a specific method for use case based estimation should guide the way the use case models are described in the company.

Acknowledgements

We gratefully acknowledge the support from our industrial partner, Mogul, in particular Jon Ola Hove, Trond Andersen, Anne Hurlen, Skule Johansen, Helge Aarstein and Sigurd Stendal. The reported work was funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

References

1. Albrecht, A.J. Measuring Application Development Productivity. Proceedings of Joint SHARE, GUIDE, and IBM Application Development Symposium. 1979.
2. Anda, B., Sjøberg, D., and Jørgensen, M. Quality and Understandability in Use Case Models. In Proc. 13th European Conference on Object-Oriented Programming (ECOOP'2001), Jørgen Lindskov Knudsen (editor), June 18-22 2001, Budapest, Hungary, LNCS 2072, Springer Verlag, pp. 402-428.
3. Arnold, P. and Pedross, P. Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department. Forging New Links. IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 490-493. 1998.
4. Boehm, B.W. *Software Engineering Economics*. Prentice-Hall. 1981.
5. Blattberg, R.C. and Hoch, S.J. Database models and managerial intuition: 50% model + 50% manager, *Management Science*, Vol. 36, No. 8, pp. 887-899. 1990.
6. Cockburn, A. Writing Effective Use Cases. Addison-Wesley. 2000.
7. Fetcke. T., Abran, A. and Nguyen, T.-H. Mapping the OO-Jacobson Approach into Function Point Analysis. International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-23). IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 192-202. 1998.
8. Jørgensen, M. An empirical evaluation of the MkII FPA estimation model, Norwegian Informatics Conference, Voss, Norway. 1997.

9. Jørgensen, M., Kirkebøen, G., Sjøberg, D., Anda and B., Bratthall, L. Human judgement in effort estimation of software projects, Beg, Borrow, or Steal Workshop, International Conference on Software Engineering, Limerick, Ireland. 2000.
10. Jørgensen, M. and Sjøberg, D.I.K. Software Process Improvement and Human Judgement Heuristics, *Accepted for publication in: Scandinavian Journal of Information Systems.* 2001.
11. Jacobson, I., Christersson, M., Jonsson, P. and Övergaard, G. *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley. 1992.
12. Jeffery, D.R., Low, G.C. and Barnes, M. A comparison of function point counting techniques. *IEEE Transactions on Software Engineering*. Vol. 19, No. 5, pp. 529-532. 1993.
13. Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December 1993.
14. Kemerer, F.K. Reliability of Function Points Measurement. *Communications of the ACM*. Vol. 36, No. 2, pp. 85-97. February 1993.
15. Kulak, D. and Guiney, E. *Use Cases: Requirements in Context*. Addison-Wesley. 2000.
16. Martinsen, S.A. and Groven, A-K. Improving Estimation and Requirements Management Experiences from a very small Norwegian Enterprise. Improvement in Practice: Reviewing Experience, Previewing Future Trends. The European Conference on Software Process Improvement (SPI 98). Meeting Management, Farnham, UK. 1998.
17. OMG Unified Modeling Language Specification, Version 1.3. June 1999. (http://www.rational.com/media/uml/post.pdf).
18. Schneider, G. and Winters, J. *Applying Use Cases – A Practical Guide*. Addison-Wesley. 1998.
19. Sendall, S. and Stroheimer, A. From Use Cases to System Operation Specification. Third International Conference on the Unified Modeling Language (UML'2000), York, UK. LNCS 1939; Springer Verlag, pp. 1-15. 2000.
20. Smith, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper. 1999.
21. Sparks, S. and Kaspcynski, K. The Art of Sizing Projects, Sun World. 1999. (http://www.sunworld.com/sunworldonline/swol-12-1999/swol-12-itarchitect.html).
22. The Object Factory. Estimating Software Projects using ObjectMetrix, White paper. April 2000.

Appendix A

Use Case Specification : Transfer loan application to new responsible

1. Use Case Name: Transfer loan application to new responsible

1.1 Brief description

The use case describes how a person responsible for a loan application can transfer it to another responsible.

2. Flow of Events

2.1 Basic Flow

1. The responsible notifies the system that he wants to transfer a specific loan application to another responsible.
2. The system displays the name of the applicant and the reference number of the application.
3. The responsible verifies that the application is correct based on the name of the applicant and the reference number.
4. The system presents a list of groups of responsibles and users within each group.
5. The responsible may choose one group of responsibles and possibly one particular responsible.
6. The responsible requests the loan application to be transferred to the chosen (group of) responsible(s).
7. The system transfers the application to the chosen (group of) responsible(s).

The use case ends successfully.

2.2 Alternative Flows

2.2.1 The responsible cancels the transfer

The responsible can cancel the transfer at any time and the use case ends.

2.2.2 Additional notification by mail

After the 5th step:

5.1 The responsible indicates that the new responsible should receive an e-mail telling him that he has received a new application for consideration.

5.2 The system automatically produces an e-mail message to the new responsible.

The use case resumes at step 6.

3. Special Requirements

4. Pre-Conditions

4.1 The responsible must be logged on to the system

The system must be started and the responsible must be logged on correctly.

5. Post-Conditions

5.1 The application has a valid status after saving

The application should be saved in the database with a valid status and in a consistent state.

5.2 The application is assigned to one responsible or to a group of responsibles

The application should be assigned to one responsible or to a group of responsibles after the transfer is completed.

6. Extension Points