Effort Estimation of Use Cases for Incremental Large-Scale Software Development

Parastoo Mohagheghi[1], Bente Anda[2], Reidar Conradi[1, 2]
[1]*Department of Computer and Information Science, NTNU, NO-7491 Trondheim, Norway*
[2]*Simula Research Laboratory, P.O.Box 134, NO-1325 Lysaker, Norway*
parastoo@idi.ntnu.no, bentea@simula.no, conradi@idi.ntnu.no

Abstract

This paper describes an empirical study on effort estimation in incremental development of a large industrial system with modification of software from a previous release.. An estimation method based on use cases, the Use case Points method, was modified in several aspects: use cases were broken down into smaller ones, classification rules were modified, both total and the modified steps of the use cases were counted, the technical and environmental factors were omitted, and a maximum person-hour per UCP was used. While use cases use cases can be used as a measure of changes in functionality between releases, there are changes in quality attributes and changes that are made later by issuing change requests that are not covered by use cases Therefore,, we used an Equivalent Modification Factor (EMF) for other changes of software not reflected in the use cases. The original method does not explicitly define which phases of development are covered by the estimates. We made estimates for all activities before system test and used an Overhead Factor (OF) equal to 2 to cover all development. Data from one release were used to develop the estimation method and the .method produced an estimate for the successive release that was only 17% lower than the actual effort. Results of the study show that the estimation method can be calibrated for such a context and produce relatively accurate estimates.

1. Introduction

Effort Estimation is a challenge in every software project. The quality of estimation will impact costs and expectations on schedule, functionality and quality. While expert estimations are widely used, they are difficult to analyze and the estimation quality depends on the experience of experts. Consequently, rule-based methods could be used in addition to expert estimates to improve these. Traditionally, software size estimated in the number of Source Line of Code (SLOC), Function Points (FP) and Object Points (OP) are used as input to cost models, e.g. COCOMO and COCOMO II [Boehm95]. Because of difficulties in estimating SLOC, FP or OP, and because modern systems are often developed using UML, UML-based software sizing approaches are proposed that can be applied in different stages of software development. Examples are effort estimation methods based on use cases [Karner93] [Smith99] and software size estimation in terms of FP from UML class diagrams and sequence diagrams [Uemura99].

The *Use Case Points* (UCP) estimation method introduced in 1993 by Karner estimates effort in Person-Hours (PH) based on external use cases that mainly specify functional requirements of a system [Karner93]. Use cases are assumed to be developed from scratch, be detailed enough but not too detailed [Ribu01] and typically have less than 10-12 transactions or steps. Use cases and actors are classified as simple, average or complex and Use Case Points are assigned to each of them. A set of technical and environmental factors are also proposed as cost drivers to arrive at a total sum of Adjusted Use Case Points. This sum is multiplied by the number of Person-Hour (PH) needed to implement each UCP (PHperUCP) to estimate the needed effort. The method has earlier been used in several industrial software development projects (small projects compared to this study) and in student projects. There have been promising results [Anda01] [Anda02], the method being more accurate than expert estimates in industrial trials.

Most software is now developed incrementally or evolutionary. There is some inconsistency in how these terms are used. We use these terms interchangeably since requirements (and thus software) evolve in both approaches, either preplanned or not. Project management and iteration planning in these projects needs an estimation method that can estimate the effort for each release based on changes in requirements. Furthermore, software is built on a previous release that should be modified or extended. There are few empirical studies on estimation of incrementally developed software projects.

This paper presents results of an empirical study on effort estimation of use cases in a large industrial system that is developed incrementally. Use cases in this study contain several complex main and alternative flows that are typically modified between releases. We broke each use case down into several ones before classifying these as in the original UCP method. We also applied the method on modified steps in each use case to estimate effort needed to implement the new or modified requirements specified in use cases. The effort needed to modify a previous release was estimated by applying a formula from COCOMO II for modification of software [Boehm95] [Boehm04] and defining a factor called *Equivalent Modification Factor* (EMF). This effort is typically spent on implementing and optimizing the non-functional requirements (perfective changes in these), and on preventive and corrective changes. The technical and environmental factors were omitted from the method by assuming an "average" project. Instead, we used a relatively high number of PHperUCP (36 here) to account for the complexity of the product. Estimates were made for all activities before system test. We multiplied the estimates by two to cover all project activities (project management, configuration management, system test, software process adaptation and other minor activities), based on experience and the effort breakdown profile of the projects. The adapted and extended effort estimation method was developed using data from one release and produced good estimates for the successive release.

We concluded that high-level use cases can predict the effort needed to realize a system, but the challenge is to define rules that account for the level of detail in use cases and how to estimate effort when these are incrementally updated. The accuracy of the estimates depends on the classification rules, the value of PHperUCP, the EMF for software modification and the activities that are assumed to be covered by the estimates. Nevertheless, the method is rule-based, can be improved in some iterations, and can be used as a supplement to expert estimates to produce early top-down estimates. We notice an increasing interest in estimation methods based on use cases, and the study contributes in evaluating the UCP method for incremental development of a large-scale system, identifying the factors that should be included in incremental effort estimation and the factors that impact the estimation accuracy.

This paper is organized as follows. Section 2 presents the UCP method and the elements of the COCOMO II method that are used in this study. Section 3 introduces the context. The research questions are formulated in Section 4. Section 5 presents the extended estimation method and Section 6 gives the estimation results. The results are further discussed in Section 7 and the research questions are answered. The paper is concluded in Section 8.

2. The underlying estimation methods

2.1. The Use Case Points (UCP) estimation method

A use case model defines the functional scope of the system to be developed. Attributes of a use case model may therefore serve as measures of the size and complexity of the functionality of a system. A recent study by Chen et al. on UML sizing metrics using 14 (small) eServices products confirmed that SLOC was moderately well correlated with the number of external use cases [Chen04].

The UCP estimation method [Karner93] is an extension of the *Function Points Analysis* and *MK II Function Points Analysis* [Symons91]. Table 1 gives a brief introduction of the six-step UCP method. In Table 1, *WF* stands for Weight Factor, *UAW* is the Unadjusted Actor Weights, *UUCW* is the Unadjusted Use Case Weights, *UUCP* is the Unadjusted Use Case Points, *UCP* is the adjusted Use Case Point, *PH* is Person-Hours and *E* is effort in PH.

There are 13 *technical Factors* (e.g. distributed system, reusable code and security) and eight *environmental Factors* (e.g. object-oriented experience and stable requirements). The weights and the formula for technical factors is borrowed form Function Points method proposed by Albrecht [Albrecht79]. Karner himself, based on some interviews of experienced personnel, proposed the weights for environmental factors, and the formula for environmental factors seems to be calculated using some estimation results.

In step 6, Karner proposed 20 PHperUCP based on three projects. Schneider et al. proposed 28 PHperUCP if the values for the environmental factors indicate negatives [Schneider98] and Russell proposed 36 PHperUCP for a complex project [Russell04].

Table 1. The UCP estimation method

Step	Task	Output
1	Classify use case actors: a) Simple, WF = 1 b) Average, WF = 2 c) Complex, WF = 3	UAW = ∑ (#Actors in each group*WF)
2	Classify use cases: a) Simple (3 or fewer transactions), WF = 5 b) Average (4 to 7 transactions), WF = 10 c) Complex (more than 7 transactions), WF= 15	UUCW = ∑ (#use cases in each group*WF)
3	Calculate the unadjusted UCP (UUCP).	UUCP = UAW + UUCW

4	Assign values to the technical and environmental factors (0..5), multiply by weights (-1..2) and calculate the weighted sums (TFactor and EFactor). Calculate TCF and EF as shown.	TCF = 0.6 + (0.01 * TFactor),
		EF = 1.4 + (-0.03 * EFactor)

5	Calculate the adjusted UCP.	UCP = UUCP * TCF * EF
6	Estimate effort in person-hour.	E = UCP * PHperUCP

Table 2 shows examples where the method is applied to three industrial projects in a company in Norway with 9-16 use cases each for new development. The application domain was banking [Anda01]. The system in this study is 20 times larger than these examples measured in UCP and 50 times larger in effort.

Table 2. Some examples on PHperUCP

Project	UCP	Est. Effort	Actual Effort	Actual PHper UCP
A	138	2550	3670	26.6
B	155	2730	2860	18.5
C	130	2080	2740	21.1

2.2. The Constructive Cost Model (COCOMO) and incremental development

COCOMO (the Constructive Cost Model) is a well-known estimation method developed originally by Barry Boehm in 1970s [Boehm95]. COCOMO takes software size and a set of factors as input and estimates effort in person-months. The basic equation in COCOMO is:

$$E = A*(Size)^B \qquad \text{EQ.1}$$

E is the estimated effort in person-month, *A* is a calibration coefficient and *B* counts for economy or diseconomy of scale. Economy of scale is observed if effort does not increase as fast as the size (i.e. $B<1.0$), because of using CASE tools or project-specific tools. Diseconomy of scale is observed because of growing communication overhead and dependencies when the size increases. COCOMO suggests a diseconomy of scale by assuming $B>1.0$. COCOMO also includes various cost drivers that fall out of the scope of this paper.

Modern software is developed incrementally or evolutionary. In both approaches, each iteration delivers a working system being an *increment* to the previous delivery or release. The Spiral method, the Rational Unified Process (RUP) and recent agile methods like eXtreme Programming (XP) are examples of software processes with incremental covering (or discovering) of requirements.

Benediktsson et al. analyzed the COCOMO II model to explore the relation between effort and the number of increments [Benediktsson03]. They concluded that incremental development will need less effort than the waterfall model when the diseconomy of scale is significant.

Although there is some evidence that incremental development reduces the risks for schedule overruns [Moløkken04], we have not found any other empirical study on the relation between incremental development and effort. One challenge is to estimate the degree of change between releases of a software system. Eick et al. reported that during maintenance of a 100 MLOC system, 20-30% of source code is added or deleted every year, with a slightly decreasing change rate over time [Eick01]. Another study by Lehman et al. reported that 60-80% of software modules are modified in the beginning of the maintenance phase, but this rate is again reduced over time [Lehman98]. It is interesting to evaluate these results when software is both maintained and evolved between releases. Another challenge in estimation of incrementally developed projects is to count for modification of software delivered in previous releases. Boehm et al. write that there are non-linear effects involved in module interface checking, which occurs during design, coding, integration and testing of modified code. The same conditions apply in incremental or evolutionary development. Only the understanding effort may be less, since development is typically done by the same organization and with relatively stable staff.

COCOMO includes a model to estimate Equivalent new KSLOC (ESLOC) from Adapted KSLOC for software reuse. COCOMO II adds some non-linear increments to the model [Boehm04]. The model is given as:

*ESLOC=ASLOC*AAM* EQ.2

*AAF = (0.4*DM+0.3*CM+0.3*IM)*

AAM=0.01[AA+AAF*{1+(0.02*SU*UNFM)}], for AAF<=50 or*

AAM=0.01[AA+AAF+(SU*UNFM)], for AAF>50*

The abbreviations in EQ.2 stand for:

AAF=Adaptation Adjustment Factor

DM=percentage of Design Modification CM=percentage of Code Modification

IM=percentage of the original Integration effort required to integrate the adapted software into an overall product

AAM=Adaptation Adjustment Modifier

AA=Assessment and Assimilation increment, [0..8]

SU=Software Understanding increment, [10 (self-descriptive code)..50 (obscure code)]

UNFM=programmer's unfamiliarity with software, [0 (completely familiar)..1 (completely unfamiliar)]

Thus if software is reused without modification (black-box reuse), DM, CM and IM are zero, but there is cost related to assessment (AA). The cost will increase with the modification degree. Note that AAM can exceed 100%; i.e. reuse may cost more than developing from scratch if the cost of assessment or understanding is high, or if the reused software is highly modified. These factors are necessarily subjective quantities.

In [Boehm04], the above formula is used to develop a cost estimation model for product line development. We are going to use it for modifying software of a previous release in incremental development.

3. The company context

The system in this study is a large distributed telecom system developed by Ericsson. It is characterized by multi-site development, development for reuse since some software components are shared with another product and multi-programming languages (mostly non object-oriented programming languages but also minor parts in Java). The software size measured in equivalent C code exceeds 1000 KSLOC (Kilo SLOC). The system is developed incrementally and the software process is an adaptation of RUP. Each release has typically 5-7 iterations, and the duration of iterations is 2-3 months. The architecture is component-based with most components built in-house. Several Ericsson organizations in different countries have been involved in development, integration and testing of releases.

At the highest level, requirements are defined by use cases and supplementary specifications (for non-functional requirements, e.g. availability, security and performance). The use case model in the study includes use case diagrams modeled in Rational Rose, showing actors and relations between use cases, while flows are described in documents called Use Case Specification (UCS). Each UCS includes:

– One or several main flows: Main flows are complex and have several steps with several transactions in each step. There may be cases when several flows are equally desired. In these cases, there may be several main flows as well.
– One or several alternative flows.
– Some use cases also have exceptional flows. These describe events that could happen at just any time and terminate a flow. Exceptional flows are described in a table, which gives the event that triggers an exceptional flow, action and the result.
– A list of parameters and constraints, such as counters or alarms.
– Use cases may *extend* or *include* another use case as well.

Each release may contain new use cases. Usually, previous use cases are also modified or extended with new or modified steps, flows or parameters. What is new or modified in each use case is marked with bold and blue font in the UCS, not distinguishing between these two types of changes. In the remainder of this paper, we use the terms UCS and use case interchangeably.

4. Motivation of the study and research questions

In this system, expert estimations are used in different phases of every release in an inside-out style: effort is estimated for activities such as design, coding and some testing in a bottom-up style, and the total effort is estimated by multiplying this estimated effort by an Overhead Factor (OF). OF varies between 1.0 and 2.5 in different estimations. The reasons behind using this factor is the experience that activities such as system test fill whatever time that is available, and the size of some activities such as project management and configuration management is proportional to the size of system. Expert estimations done by technical staff tend to be over-optimistic. The UCP method can, on the other hand, be applied also by non-technical staff and is rule-based, allowing adaptation and improvement. We consequently decided to extend and evaluate this estimation method. Already when the UCP method was introduced to the project leaders to get their permission for the study, it was considered interesting. A project leader used it in addition to expert estimates by considering the amount of changes in use cases comparing to the previous release. The UCP method in its original form estimates effort needed to develop use cases from scratch, but it is not clear which activities (phases) are covered and it is not tested on large systems.

We have formulated the following research questions for this study:

RQ1: Does the UCP method scale up for a large industrial project?

RQ2: Is it possible to extend the UCP method for incremental changes in use cases?

RQ3: How to calculate effort needed to modify software from a previous release?

RQ4. Does the method produce reasonable results in this industrial setting?

5. The extended UCP estimation method

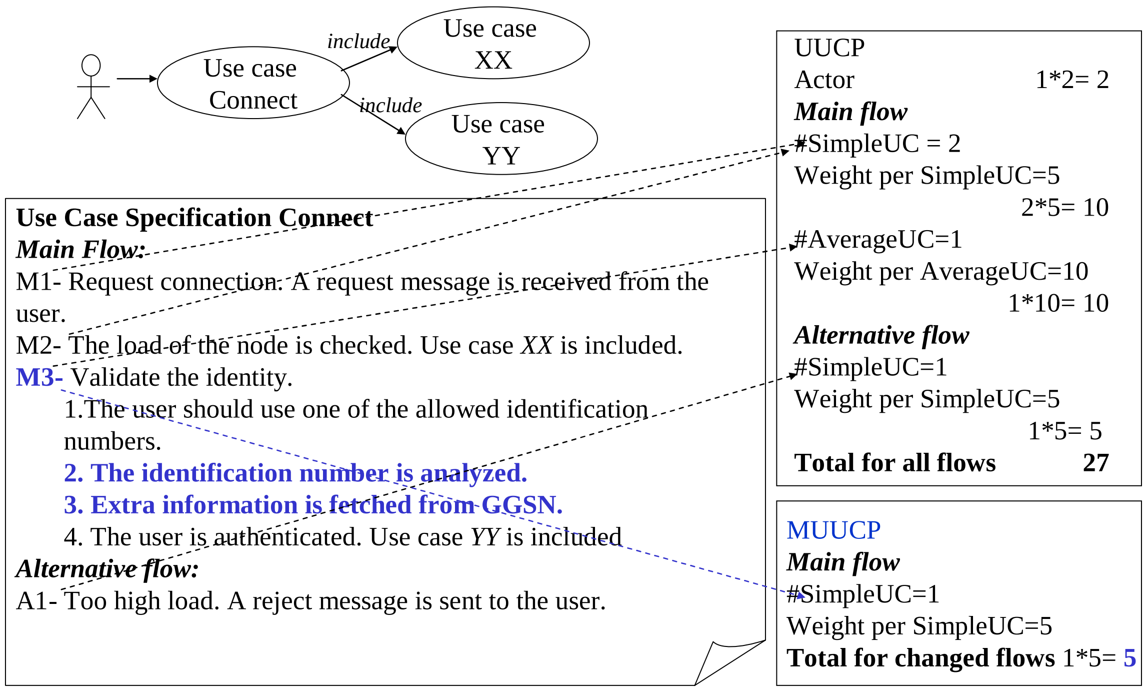
This section describes how the UCP method has been adapted to the industrial context and is extended for incremental development. The new rules are summarized in Table 3. Additional information on each step is given below.

Step 1. Actors. An actor may be a human, another system or a protocol. However, the classification has little impact on the final estimation result and all actors are assumed as average. Modified actors are also counted and MUAW is the Modified UAW.

Step 2. Counting the UUCW and MUUCW (Modified UUCW). We started to count UUCW for release 1 using the method described in Section 2.1. All use cases in this study would be classified as complex. Nevertheless, the total UUCP would be still very low for all the 23 use cases (23*15=345 UUCP). Comparing the complexity of these use cases with previous projects convinced us that we have to break use cases down into smaller ones as described in Rules 2.1 to 2.4. Rewriting use cases is too time-consuming while counting flows and steps is an easy task. Use cases should then be classified as simple, average or complex. A first attempt to follow the rule described in Section 2.1 resulted in most use cases being classified as simple (66%), and very few as complex. But the complexity of transactions does not justify such distribution.

Table 3. The extended UCP estimation method

Step	Rule	Output
1	1.1. Classify all actors as Average, WF = 2.	UAW = #Actors*2
	1.2. Count the number of new/modified actors.	MUAW = #New actors*2
2	2.1. Since each step in the main flow contains several transactions, count each step as a single use case. 2.2. Count each alternative flow as a single use case. 2.3. Exceptional flows, parameters, and events are given weight 2. Maximum weighted sum is limited to 15 (a complex use case). 2.4. Included and extended use cases are handled as base use cases. 2.5. Classify use cases as: a) Simple (2 or fewer transactions), WF = 5, b) Average (3 to 4 transactions), WF = 10, c) Complex (more than 4 transactions), WF= 15.	UUCW = ∑(#use cases in each group*WF) + ∑(Points for exceptional flows and parameters)
	2.6. Count points for modifications in use cases according to rules 2.1-2.5.	MUUCW = ∑(#New/modified use cases in each group*WF) + ∑(Points for new/modified exceptional flows and parameters)
3	3.1. Calculate UUCP for all software.	UUCP = UAW + UUCW
	3.2. Calculate MUUCP for new/modified software.	MUUCP = MUAW + MUUCW
4	Assume average project.	TCF = EF = 1
5	5.1. Calculate UCP.	UCP = UUCP
	5.2. Calculate MUCP.	MUCP = MUUCP
6	6.1. Estimate effort for new/modified use cases.	E1 = MUCP * PHperUCP
	6.2. Estimate effort for other changes of software.	E2 = (UCP-MUCP) * EMF * PHperUCP
	6.3. Estimate total effort.	E = E1 + E2



Use case Connect
include
Use case XX
include
Use case YY
Use Case Specification Connect
Main Flow:
M1- Request connection. A request message is received from the user.
M2- The load of the node is checked. Use case XX is included.
M3- Validate the identity.
1.The user should use one of the allowed identification numbers.
2. The identification number is analyzed.
3. Extra information is fetched from GGSN.
4. The user is authenticated. Use case YY is included
Alternative flow:
A1- Too high load. A reject message is sent to the user.
UUCP
Actor 1*2= 2
Main flow
#SimpleUC = 2
Weight per SimpleUC=5
2*5= 10
#AverageUC=1
Weight per AverageUC=10
1*10= 10
Alternative flow
#SimpleUC=1
Weight per SimpleUC=5
1*5= 5
Total for all flows 27
MUUCP
Main flow
#SimpleUC=1
Weight per SimpleUC=5
Total for changed flows 1*5= 5

Figure 1. Example of counting UUCP and MUUCP for a use case

We give an example of a use case called *Connect* in Figure 1. Two transactions in the main flow are modified or are new, and are classified as a new use simple case. Thus this use case is modified by 5/27=0.19. In Figure 1, M1 is described in one transaction, but it includes verifying that the received message is according to the accepted protocols. M2 refers to an included use case. M3 has four transactions where none of these is a single transaction and includes another use case as well. Therefore, we chose to classify the use cases according to the Rule 2.5. So M1 and M2 would be classified as simple, while M3 would be an average use case.

Karner proposed not counting so-called including and extending use cases, but the reason is unclear. We have applied the same rules to all the use cases.

Step 4. TCF and EF. Assigning values to technical and environmental factors are usually done by project experts or project leaders, based on their judgment [Anda01] [Ribu01]. The technical factors reflect some non-functional requirements, but the authors of the above papers conclude that the technical factors can be omitted (or be set to 1) without significant consequences for the estimate, since the impact of TCF is small. The environmental factors may have a large impact on the estimate, but the formula should be validated. We decided to simplify the method by assuming an average project, which gives TCF and EF approximately equal to 1.

Step 5. The adjusted UCP and MUCP will be equal to the unadjusted ones.

Step 6. We assume that there are two mechanisms that consume effort in our model: E1 estimates effort for realizing new and modified requirements as specified in use cases (or *primary* changes), while E2 estimates effort for other changes (*secondary* changes). We discuss E2 in more details here.

As discussed in Section 2.2, there is an overhead for changing software of a previous release. In addition to changes related to functionality in E1, there are several other reasons for why software is modified:

1. **Perfective functional changes not specified in use cases**: Functionality is enhanced and improved between releases by initiating change requests. There may also be a ripple effect of changes in use cases.
2. **Perfective and corrective non-functional changes**: Quality is improved between releases (performance, security, reliability etc.) and the share of changes due to quality requirements is high, but these changes are not reflected in use cases. We performed a study of change requests for Rel.1 and two previous releases [Mohagheghi04b]. Improving quality requirements is by

modifying software that is already implemented. Some effort is also spent on modifying software to correct defects.
3. **Preventive changes** to improve design and reduce software decay also consume effort.

We decided to use the model proposed in EQ.2 as a first trial to estimate EMF. For our model, in the simplest form we propose:

- AA=2, we assume low search, test and evaluation cost for software developed in-house.
- SU=30 for moderate understandable software.
- In [Mohagheghi04a], we reported that source code is approximately modified by 55% between releases, and this can be used as mean value for CM. DM is usually less than CM and is set to 30 here, which is slightly over the mean value of changes in use cases. IM is set to be 65, i.e. slightly over CM [Boehm04]. These values give AAF=48.
- UNFM=0.2 for mostly familiar with code

Thus, EMF (AAM in EQ.2) will be equal to 0.56. We have not found any empirical studies that contain such a factor in incremental development. This factor gives equivalent UCP for other modifications. The above values can be calibrated and adjusted more, In 6.2, we multiply EMF with the size of use case points that are not included in 6.1. Alternatively, we could multiply it with SLOC from the previous release and use EQ.1 to estimate effort for this part. Since we wanted to valuate the UCP method, we used use case points also in this step.

In this project we decided to compensate for not counting the environmental factors and for the large number of complex use cases, by using the maximum recommended number of PHperUCP that is 36. However, estimates should specify which phases (activities) of a development project are covered by them. As later described in Section 6, we found that our estimates cover development before system test and should be multiplied by an OF equal to 2 to account for all phases.

6. Estimation results

The estimation method was developed based on the use cases of one release and was later tested on the use cases of the successive release. Of the 23 original use cases in Rel.1, seven use cases were not modified, one use case was new, while 15 use cases were modified. Rel.2 had 21 use cases: two use cases were not modified, one use case was new, while 18 were modified. Note that 3 use cases are missing in Rel.2 (the sum should be 24). Two use cases were merged in other use cases, while one use case is removed from our analysis since development was done by another organization and we do not have data on effort here. We inserted the number of use cases, actors, exceptions and parameters in spreadsheets in Microsoft Excel, counted the UUCP and MUUCP, and estimated the effort following the rules in Table 3. Table 4 shows that after break-down of the use cases (UC), we ended up with 288 use cases in Rel.1 and 254 use cases in Rel.2.

Table 4. No. of use cases in each class

Rel.	Simple UC	Average UC	Complex UC	Mod. Simple UC	Mod. Average UC	Mod. Complex UC
Rel. 1	170	83	35	57	18	2
Rel. 2	95	100	59	81	16	11

The distribution of use cases has changed towards more average use cases in Rel.2. According to Cockburn most well-written use cases have between 3 and 8 steps (transactions), consequently most use cases will be of medium complexity, some are simple and a few are complex [Cockburn00]. Our results only verify this for one release.

The estimates with 36 PHperUCP were almost half the effort spent in Rel.1 for all activities. Therefore we compared our releases with the examples discussed before in other aspects. In the projects *A* and *B* in Table 2, estimates have been compared with the total effort after the construction of the use case model. These projects' effort distribution is very different from the releases of this system, as shown in Tables 5 and 6. The *Other* column in Table 5 covers deployment and documentation, while in Table 6 it covers CM, software process adaptation, documentation and travel. These profiles will vary depending on tools, environment and technologies. In our case, development before system test (also including use case testing added by Ericsson) only counts for half the effort. The existing estimation method in the company estimates effort needed for development before system test and multiplies this by an Overhead Factor (OF) to cover all project activities. We concluded that the 36 PHperUCP covers only development before system test. Based on empirical data presented in Table 6, it should be multiplied by approximately 2 to estimate the total effort.

Table 5. Percentages of actual effort spent in different activities in example projects

Project	Development before System Test	System Test	Other	Project Mngt
A	80%	2%	5%	13%
B	63%	7%	3%	27%

Table 6. Percentages of actual effort spent in Rel. 1 and 2 in this study

Rel.	Development before System Test	System Test	Other	Project Mngt
Rel.1	49%	25%	15%	10%
Rel.2	55%	18%	15%	11%

For confidentiality reasons, we cannot give the exact figures for estimations and effort. However, our estimations were 21% lower for Rel.1 and 17% lower for Rel.2 than the spent effort, which has been around 100 person-labor year; i.e. indeed very accurate for large projects. For comparison, the effort to develop the first release was 2-3 times this number. The expert estimations for Rel.2 were 35% lower than the actual effort and thus the method has lower relative error in this case than expert estimations.

7. Discussion of the results and answers to research questions

We consider the data on effort as reliable and we have had access to all the use cases. The estimation was done by us (the first author was an employee of Ericsson at the time) and one person spent around a week on classification of use cases. The second author had previous experience with the UCP method and had applied it on several projects. Although the method was presented to a few project leaders, we could not involve them in the adaptation work due to internal reorganizations.

The empirical study of Chen et al. [Chen04] and earlier studies on the UCP showed that use cases can be used as a measure of the size of a system, and steps in use cases can be used as a measure of their complexity. The results show that the adapted and extended UCP method produced reasonable estimates with the following adaptations:

1. **Large use cases**: we broke each use case down to several smaller ones. The method was only tested on small systems before.
2. **Level of detail in use cases**: we modified the UCP classification rules, justified by the complexity of transactions.
3. **Technical and environmental factors**: these were omitted. Since the method is adjusted to a context and is used on successive releases of the same system, these factors are redundant.
4. **Incremental modification of use cases**: we calculated effort for new and modified transactions. We did not distinguish between these two types of changes in use cases.
5. **PHperUCP**: we used the largest value proposed by other studies, i.e. 36. The results showed that it covers effort for specification (after use case modeling), design, coding, module testing and use case testing.

We also added two elements to the estimation method:

1. **The Equivalent Modification Factor (EMF)**: this factor is used to estimate equivalent UCP for modification of software from a previous release, and is set to 0.56.
2. **The Overhead Factor (OF)**: this empirically derived factor is used to estimate the total effort based on effort for development before system test. It is set to 2.

EMF, PHperUCP and OF rely on empirical observations and our judgments can be subject of further adjustments. All estimation methods are imprecise, because the assumptions are imprecise. The method was adapted using data from Rel.1, but it even gave better results for Rel.2. Each estimate should also come with a range, starting with a wider range for early estimates. Use cases are updated in the early design stage (the inception and elaboration phases in RUP), which gives a range of 0.67*E* to 1.5*E* according to COCOMO II [Boehm95].

The impact of E2 is large on the total effort (65% of effort in Rel.1 and 55% in Rel.2), due to the value of EMF and the assumption that in addition to portions of the software affected by changes in the use cases, the rest of the software is modified for other reasons. Parts of software may become more stable after a few releases or be used more in a black-box style without modification and E2 can decrease. But software also decays and there is cost related to refactoring and redesign. The study also showed that use cases are approximately modified by 23-31% (counted in UCP), while SLOC is typically modified by 50-60% between releases. Boehm et al. also propose that CM is usually larger than DM [Boehm04]. Here CM shows to be larger than changes in use cases. We also add the impact of modifications not specified in use cases and the effort to maintain and improve quality. Empirical studies of these factors can help us to understand the nature and rate of incremental evolution.

The UCP method showed flexibility in adapting to the context, but there are many assumptions in the original method and in our extensions of it. In other words, the study has revealed those factors in the method that needed adaptation for large-scale incremental development. Future studies can help to understand the practice of incremental evolution, how the method works on other types of systems and which parameters can be generalized to certain types of systems, like the EMF, PHperUCP and OF.

We answer the research questions as follows:

RQ1: Does the UCP method scale up for a large industrial project? It did when we broke down the use cases as reflected in Rules 2.1-2.5 in Table 3. The method depends on the level of details in use cases. One alternative is to include examples of typical use cases in the method, such as those defined in [Cockburn00].

RQ2: Is it possible to apply the UCP method to incremental changes in use cases? We did this by counting changes in use cases. The method is straightforward and Rules 1.2, 2.6, 3.2, and 6.1 in Table 3 show how to calculate effort for new/modified use cases.

RQ3: How to calculate effort needed to modify software from a previous release? We used the COCOMO 2.0 formula for adapted software, calculated EMF and applied it on UUCP for the rest of the software. The advantage is that EMF may be adapted to the context.

RQ4. Evaluation of the method: The adapted and extended UCP method fitted well into the adapted RUP process and produced reasonable results.

We also observe the impact of size and complexity of the system in the high value of PHperUCP, the overhead factor and the effort needed to maintain software (reflected in EMF). The study also the question on whether the value of PHperUCP depend on the effort break down profile and should this factor be included in the model?

8. Conclusions

An effort estimation method based on use cases has been developed and tested on a large industrial system with incremental changes in use cases. One main assumption is that use cases may be used a measure of the size of a system, and changes in these may be used as a measure of changes in functionality between releases. The method is suitable when guessing SLOC is difficult such as development with COTS or in multiple programming languages. It is also suitable when development is going to be outsourced. The method does not depend on any tools and can promote high quality use cases

The main contributions of the study are:

1. **Testing the UCP method on a large industrial project.** The proposed changes for breaking down use cases should be applicable to other projects by comparing their use cases to some example ones and calibrating the rules in Table 3 to their context. For a cost model to scale up, additional factors such as a higher value of PHperUCP and OF may be necessary. Although there is a relation between UCP and effort, this relation is not linear, as shown here.
2. **Extending the UCP method for incremental development** by accounting for modification of software from a previous release and counting changed transactions in use cases. The EMF factor is added and the impact of quality requirements in software evolution is discussed.
3. **Simplifying the UCP method.** The UCP method worked well without the technical and environmental factors. The impact of these factors is accounted for in other factors.
4. **Showing the impact of effort breakdown profiles on estimation results.**

An estimation method for use cases depends on some up-front requirement work and use cases that are well structured and described at a proper level of detail. Furthermore, use cases essentially express functional requirements. The influence of non-functional requirements should be included in the technical factors, the number of PHperUCP or be added to the estimated effort as done here. We have achieved a better understanding of software evolution in incremental development. The study also shows that several factors could be integrated in an effort estimation model such as information on the rate and impact of change requests and defects, degree of modification of SLOC and the impact of quality requirements.

There are few empirical studies on estimation of incrementally developed projects. The proposed factors (EMF, OF and PHperUCP) and modifications in the method could be subject of other studies.

9. Acknowledgements

The study was performed in the context of INCO (INcremental and COmponent-based Software Development), a Norwegian R&D project in 2001-2004 [INCO01], and as part of the first author's PhD study. We thank Ericsson for the support.

References

[Albrecht79] Albrecht, A.J., "Measuring Application Development Productivity". *Proc. IBM Applic. Dev. Joint SHARE/GUIDE Symposium*, Monterey, CA, 1979, pp. 83-92.

[Anda01] Anda, B., Dreiem, D., Sjøberg, D.I.K., and Jørgensen, M., "Estimating Software Development Effort Based on Use Cases - Experiences from Industry". In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools, 4th International Conference, 2001, *Springer-Verlag LNCS* 2185, pp. 487-502.

[Anda02] Anda, B., "Comparing Effort Estimates Based on Use Cases with Expert Estimates". *Proc. Empirical Assessment in Software Engineering* (EASE 2002), 2002, ?? 13 p.

[Benediktsson03] Benediktsson, O., Dalcher, D., "Developing a new Understanding of Effort Estimation in Incremental Software Development Projects". *Proc. Intl. Conf. Software & Systems Engineering and their Applications* (ICSSEA'03), Volume 3, Session 13, 2003, 10 p.

[Boehm95] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R., "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. *USC center for software engineering*, 1995. http://sunset.usc.edu/publications/TECHRPTS/1995/index.html

[Boehm04] Boehm, B., Brown, W., Madachy, R., Yang, Y., "Software Product Line Cycle Cost Estimation Model". *Proc. the ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*, 19-20 August 2004, Redondo Beach CA, USA, *IEEE CS Order No. P2165, pp. 156-164.*

[Chen04] Chen, Y., Boehm, B.W,, Madachy, R., Valerdi, R., "An Empirical Study of eServices Product UML Sizing Metrics". *Proc. the ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*, 19-20 August 2004, Redondo Beach CA, USA, pp. 199-206.

[Cockburn00] Cockburn, A., "*Writing Effective Use Cases*". Addison-Wesley, 2000.

[Eick01] Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., Mockus, A. "Does Code Decay? Assessing the Evidence from Change Management Data". *IEEE Trans. SE*, 27(1):1-12, Jan 2001.

[INCO01] http://www.ifi.uio.no/~isu/INCO/

[Karner93] Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21, December 1993.

[Lehman98] Lehman, M.M., Perry, D.E. Ramil, J.F., "Implications of Evolution Metrics on Software Maintenance. *Proc. ICSM 1998*, 16-19 Nov. Maryland, USA, pp. 208-219.

[Mohagheghi04a] Mohagheghi, P., Conradi, R., Killi, O.M., Schwarz, H., "An Empirical Study of Software Reuse vs. Defect-Density and Stability. Proc. The 26th Int'l Conference on Software Engineering (ICSE'04), May 23-28, 2004, Edinburgh, Scotland, *IEEE CS Order No. P2163*, pp. 282-292.

[Mohagheghi04b] Mohagheghi, P., Conradi, R., "An Empirical Study of Software Change: Origin, Impact, and Functional vs. Non-Functional Requirements". *Proc. the ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*, 19-20 August 2004, Redondo Beach CA, USA, *IEEE CS Order No. P2165*, 10 p.

[Moløkken04] Moløkken, K., Lien, A.C., Jørgensen, M., Tanilkan, S.S., Gallis, H., Hove, S.E.," Does Use of Development Model Affect Estimation Accuracy and Bias?". Proc. the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004), *Springer LNCS 3009*, April 5-8, 2004, Japan, pp. 17-29.

[Ribu01] Ribu, K. Estimating Object-Oriented Software Projects with Use Cases. Masters Thesis, University of Oslo, November 2001.

[Russell04] Rusell, R. Visited on July 2004, http://www.processwave.net/index.htm

[Schneider98] Schneider, G. & Winters, J.P., "*Applying Use Cases, a Practical Guide*". Addison-Wesley, 1998.

[Smith91] Smith, J., "The Estimation of Effort Based on Use Cases". *Rational Software*, White paper, 1999.

[Symons91] Symons, P.R., *Software Sizing and Estimating MK II FPA (Function Point Analysis)*, John Wiley & Sons, 1991.

[Uemura99] Uemura, T., Kusumoto, S., Inoue, K., "Function Point Measurement Tool for UML Design Specification". *Proc. Sixth Int'l Software Metrics Symposium*, 1999.