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

Parastoo Mohagheghi
Department of Computer and Information Science
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway
and
Faculty of Engineering
Agder University College
NO-4876 Grimstad, Norway
+47 37 25 34 22
parastoo@idi.ntnu.no

Bente Anda
Simula Research Laboratory
P.O.Box 134
NO-1325 Lysaker, Norway
+47 67 82 83 06
bentea@simula.no

Reidar Conradi
Department of Computer and Information Science
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway
and
Simula Research Laboratory
P.O.Box 134
NO-1325 Lysaker, Norway
+47 73 59 34 44
conradi@idi.ntnu.no

ABSTRACT
This paper describes an industrial study of an effort estimation method based on use cases, the Use Case Points method. The original method was adapted to incremental development and evaluated on a large industrial system with modification of software from the previous release. We modified the following elements of the original method: a) complexity assessment of actors and use cases, and b) the handling of non-functional requirements and team factors that may affect effort. For incremental development, we added two elements to the method: c) counting both all and the modified actors and transactions of use cases, and d) effort estimation for secondary changes of software not reflected in use cases. We finally extended the method to: e) cover all development effort in a very large project. The method was calibrated using data from one release and it produced an estimate for the successive release that was only 17% lower than the actual effort. The study identified factors affecting effort on large projects with incremental development. It also showed how these factors can be calibrated for a specific context and produce relatively accurate estimates.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *cost estimation, life cycle.*

General Terms
Management, Experimentation.

Keywords
Estimation, use cases, incremental development.

ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.

1. INTRODUCTION
Effort estimation is a challenge in every software project. The estimates will impact costs and expectations on schedule, functionality and quality. While expert estimates are widely used, they are difficult to analyze and the estimation quality depends on the experience of experts from similar projects. Alternatively, more formal estimation models can be used. Traditionally, software size estimated in the number of Source Lines of Code (SLOC), Function Points (FP) and Object Points (OP) are used as input to these models, e.g. COCOMO and COCOMO II [5].

Because of difficulties in estimating SLOC, FP or OP, and because modern systems are often developed using the Unified Modeling Language (UML), UML-based software sizing approaches are proposed. Examples are effort estimation methods based on use cases [11] [20], and software size estimation in terms of FP from various UML diagrams [22] [7].

The *Use Case Points* (UCP) estimation method introduced in 1993 by Karner estimates effort in person-hours based on use cases that mainly specify functional requirements of a system [11] [12]. Use cases are assumed to be developed from scratch, be sufficiently detailed and typically have less than 10-12 transactions. 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 and the method was more accurate than expert estimates in industrial trials [2] [3].

Recently, incremental or evolutionary development approaches have become dominant: requirements are covered (or discovered) in successive releases, and changing requirements (and software) is accepted as a core factor in software development. The Rational Unified Process (RUP) and agile methods like eXtreme Programming (XP) are examples. Each release of a software system may in turn be developed in iterations of fixed or variable duration, and may be maintained for a while, before being replaced with a new release. Project management 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.

This paper presents results of an empirical study on effort estimation of use cases in a large industrial system that is developed incrementally. We adapted the UCP estimation method for complex use cases with modified transactions. The effort needed to modify a previous release was estimated by applying a formula from COCOMO II for modification of reused software and in the so-called Equivalent Modification Factor (EMF).

We concluded that 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 software is incrementally updated. The accuracy of the estimates depends on many factors that are empirically set. Nevertheless, the method is rule-based, can be improved and be used as a supplement to expert estimates to produce early top-down estimates.

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 those factors that impact estimation accuracy.

This paper is organized as follows. Section 2 presents the UCP method and the elements of the COCOMO method that are used in this study. Section 3 introduces the context. The research questions are formulated in Section 4. Section 5 presents the adapted UCP estimation method and Section 6 gives the estimation results. The results are further discussed in Section 7 and the research questions are answered in Section 8. Section 9 discusses relation to other work. Finally, the paper is concluded in Section 10.

2. THE UNDERLYING ESTIMATION METHODS

2.1 The Use Case Points 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 showed that SLOC was moderately well correlated with the number of use cases, and that transactions in use cases can be used as measures of their complexity [8].

The Use Case Points (UCP) estimation method is an extension of the *Function Points Analysis* and *MK II Function Points Analysis* [21]. Table 1 gives a brief introduction of the six-step UCP method.

In step 4, there are 13 *technical factors* (related to how difficult it is to build the system, e.g. distributed system, reusable code and changeability) and eight *environmental factors* (related to the efficiency of the project e.g. object-oriented experience and stable requirements). The weights and the formula for technical factors is borrowed form the Function Points method proposed by Albrecht [1]. Karner himself interviewed experienced personnel and proposed the weights for environmental factors. The formula for environmental factors is based on some estimation results.

In step 6, the adjusted Use Case Points (UCP) is multiplied by person-hours needed to implement each use case point (PHperUCP). The literature on the UCP method proposes from 20 to 36 PHperUCP [11] [18] [19].

Table 1. The UCP estimation method

Step	Rule	Output
1	Classify actors: a) Simple, WF (Weight Factor) = 1 b) Average, WF = 2 c) Complex, WF = 3	Unadjusted Actor Weights (UAW) = ∑(#Actors * 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	Unadjusted Use Case Weights (UUCW) = ∑ (#Use Cases * WF)
3	Calculate the Unadjusted Use Case Point (UUCP).	UUCP = UAW + UUCW
4	Assign values to the technical and environmental factors [0..5], multiply by their weights [-1..2], and calculate the weighted sums (TFactor and EFactor). Calculate TCF and EF as shown.	Technical Complexity Factor (TCF) = 0.6 + (0.01 * TFactor) Environmental Factor (EF) = 1.4 + (-0.03 * EFactor)
5	Calculate the adjusted Use Case Points (UCP).	UCP = UUCP * TCF * EF
6	Estimate effort (E) in person-hours.	E = UCP * PHperUCP

Table 2 shows examples where the UCP method is applied to three projects in a company in Norway with 9-16 use cases each for new development. The application domain was banking. The system in this study is 20 times larger than the examples in Table 2 measured in use case points and 50 times larger in effort, something which motivates a modification of the method for larger systems.

Table 2. Some examples from [2]

Project	UCP	Estimated effort	Actual effort	Actual PHperUCP
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)

The Constructive Cost Model (COCOMO) is a well-known estimation method developed originally by Barry Boehm in 1970s [5]. COCOMO takes software size and a set of factors as input and estimates effort in person-months. The basic equation in COCOMO is:

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

In *EQ.1*, *E* is the estimated effort in person-months, *A* is a calibration coefficient, *Size* may be in OP, FP or SLOC, and *B* counts for economy or diseconomy of scale. Economy of scale is observed if effort does not increase as fast as the size, because of e.g. using CASE 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. COCOMO assumes an incremental model of development, but the effort is estimated for the whole project.

One challenge in estimation of incrementally developed projects is to estimate the degree of change between releases of a software system. This is earlier studied in the context of maintenance. Eick et al. reported that during maintenance of a 100 Million LOC system, 20-30% of source code is added or deleted every year, with a slightly decreasing change rate over time [10]. 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 [13]. It is interesting to evaluate these results when software is both maintained and is significantly evolved between releases.

Another challenge is to consider modification of software delivered in a previous release to integrate new functionality, improve quality, restructure, and correct bugs and other "secondary" changes. Boehm et al. write that there are non-linear effects involved in module interface checking (when *k* out of *m* software modules are modified), which occurs during design, coding, integration and testing of modified code [5]. The same conditions apply in incremental or evolutionary development. Only effort for software understanding may be less, since development is done by the same organization and with relatively stable staff.

COCOMO II includes a model to estimate *Equivalent new kilo Source Lines of Code* (ESLOC) from *Adapted kilo Source Lines of Code* (ASLOC, size of the reused product) for software reuse [6]. The model is shown in Figure 1.

If software is reused without modification (black-box reuse), the only cost of reuse is related to assessment and assimilation. The cost will increase with the modification degree. Note that the adaptation cost 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 [6], the above model is used to develop a cost estimation model for product line development.

COCOMO has also an extension for software maintenance, predicting maintenance effort as a function of initial development effort, the annual change traffic and an adjustment factor that reflects the difference between development and maintenance. This model may be used for maintenance of each release in incremental development (with mainly corrective changes). New requirements are usually forwarded to future releases.

ESLOC = ASLOC * AAM,

where

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

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

and

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

The abbreviations stand for:

AAM = Adaptation Adjustment Modifier

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

AAF = Adaptation Adjustment Factor

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

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

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

Figure 1. COCOMO II model for estimating cost of software reuse

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 (almost 60% of software components are shared with another product) and several programming languages (C, Erlang, Perl and Java). The software size of the latest release in this study is more than 450 KSLOC (Kilo SLOC), or about 1000 KSLOC measured in equivalent C code.

The software process is an adaptation of RUP. Each release typically has 5-7 iterations, and the duration of iterations is 2-3 months. Until now, five main releases of the system are delivered to the market. The architecture is component-based with most components built in-house. 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 contains use case diagrams modeled in Rational Rose, showing actors and relations between use cases, while flows are described in separate documents called *use case specifications*. Each use case specification includes:

- One or several main flows that are complex and each have several transactions.
- One or several alternative flows.

- A list of parameters and constraints, such as counters and alarms.
- Possible exceptional flows. These describe events that could happen at 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.
- Use cases may *extend* or *include* other use cases.

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

4. MOTIVATION OF THE STUDY AND RESEARCH QUESTIONS

In this system, expert estimates 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 the above estimated effort by an *Overhead Factor* (OF). OF varies between 1.0 and 2.5 in different estimates based on the project and the time of estimation. The reasons behind this factor are that activities like system test tend to fill whatever time available, and that the size of some activities such as project management is proportional to the size of the system.

Expert estimates 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 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 compared to the previous release.

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 apply the UCP method to incremental changes in use cases?

RQ3. How can effort to modify software from a previous release be estimated?

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

The UCP method has earlier been tested only on small projects and with development from scratch.

5. THE ADAPTED UCP ESTIMATION METHOD

5.1 Overview of the Adapted Method

This section describes how the UCP method has been modified. The new rules are summarized in Table 3, with the same abbreviations as in Table 1 and as described below.

Step 1. Actors. An actor may be a human, another system or a protocol. Since the classification has little impact on the final estimation result, all actors are assumed to be average. Modified actors are also counted as the Modified Unadjusted Actor Weights (MUAW).

Step 2. Counting the UUCW and MUUCW. We started to count the Unadjusted Use Case Weights (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 use case points would be still very low for all the 23 use cases (23 * 15 = 345 UUCW). But these use cases are much larger and more complex than in previous studies. Therefore we decided to break use cases down into smaller ones (not the use case specifications, but only in counting) as described in Rules 2.1 and 2.2. Rewriting use cases is too time-consuming while counting flows and transactions is an easy task.

Use cases should then be classified. Applying step 2 in Table 1 led to most new use cases being classified as simple (66%), and very few as complex. However, the complexity of transactions does not justify such distribution.

An example of a use case called *Connect* is given in Figure 2. It has one main flow with three transactions (*M1*, *M2* and *M3*) and one alternative flow with one transaction (*A1*). Note that each transactions may in turn include several (sub)transactions. Here, *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 classified the use cases according to Rule 2.5. So *M1*, *M2* and *A1* would be classified as simple use cases, while *M3* would be an average one.

In Rule 2.6, modified transactions or parameters are counted. In Figure 2, two transactions in *M3* are modified or are new (in bold text) and are classified as a new simple use case. Thus this use case is modified by 5/27 = 0.19.

Karner proposed not counting so-called included and extended use cases, but the reason is unclear. We have applied the same rules to all the use cases. However, these are counted only once (like other use cases).

Step 3. Counting UUCP. The Unadjusted Use Case Points (UUCP) are calculated; once for all use cases (in Rule 3.1) and once for modifications (in Rule 3.2).

Step 4. TCF and EF. Assigning values to technical and environmental factors are usually done by experts or project leaders, based on their judgment [2]. The impact of the Technical Complexity Factor (TCF) is small and it does not cover all the non-functional requirements either. We have therefore handled this otherwise, as described in step 6 below. The Environmental Factor (EF) is not relevant in this project as there are few changes to this factor from one release to another. However, this factor does have a large impact on the estimate and we have accounted

for omitting it by using a high PHperUCP. Dropping these factors is also suggested in other cost models [13].

Step 5. The adjusted UCP and MUCP will be equal to the unadjusted ones since TCF and EF are set to 1.

Step 6. We assume that there are two mechanisms that consume effort in our model: *E_primary* estimates effort for realizing new and modified use cases, while *E_secondary* estimates effort for *secondary changes* as described below. The total estimated effort is the sum of these.

5.2 Effort Estimation for Secondary Changes of Software

In addition to changes related to functionality estimated in *E_primary*, there are several other reasons for why software is modified:

1. **Perfective functional changes not specified in use cases**: Functionality is also enhanced and improved in each release by initiating so-called change requests after requirement freeze. There may also be ripple effects of changes in use cases.
2. **Perfective non-functional changes**: Quality attributes are improved between releases (performance, security, reliability etc.), but these changes are not reflected in use cases. A study of change requests for four releases of the same system showed that 44% of change requests were issued to improve quality attributes [16]. Improving quality attributes is usually achieved by modifying software that is already implemented.
3. **Corrective changes**: Some effort is also spent on modifying software to correct detected defects.
4. **Preventive changes** to improve design and reduce software decay also consume effort. Also note that preventive changes to improve file structure or to reduce dependencies between software modules may later impact quality attributes such as maintainability.

We decided to use the model described in Figure 1 as a first trial to estimate effort for secondary modifications by a so-called *Equivalent Modification Factor (EMF)*. For our model, in the simplest form we propose:

- AA (Assessment and Assimilation) = 2%, we assume low search, test and evaluation effort for software developed in-house.
- SU (Software Understanding) = 30% for moderate understandable software.
- In [16], we reported that source code is approximately modified by 55% between releases, and this can be used as mean value for CM (Code Modification percentage).
- DM (Design Modification percentage) is usually less than CM and is set to 30% here, which is close to the mean value of changes in use cases (23-31% counted in use case points).
- IM (Integration effort percentage) is set to be 65%, i.e. slightly over CM [6].
- UNFM (Unfamiliarity with software) = 20% for mostly familiar with code.

Table 3. The adapted 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.	Modified UAW (MUAW) = #New or modified actors * 2
2	2.1. Since each transaction in the main flow contains one or several transactions, count each transaction 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	Unadjusted Use Case Weights (UUCW) = ∑ (#Use Cases * WF) + ∑(#Use Case Points for exceptional flows and parameters from 2.3)
	2.6. Count points for modifications in use cases according to rules 2.1-2.5 to calculate the Modified Unadjusted Use Case Weights (MUUCW).	MUUCW = ∑ (#New/modified Use Cases * WF) + ∑(Points for new/modified exceptional flows and parameters)
3	3.1. Calculate UUCP for all software.	UUCP = UAW + UUCW
	3.2. Calculate Modified UUCP (MUUCP).	MUUCP = MUAW + MUUCW
4	Assume average project.	TCF = EF = 1
5	5.1. Calculate adjusted Use Case Points (UCP).	UCP = UUCP
	5.2. Calculate adjusted Modified UCP (MUCP).	MUCP = MUUCP
6	6.1. Estimate effort for new/modified use cases.	*E_primary*= MUCP * PHperUCP
	6.2. Estimate effort for secondary changes of software.	*E_secondary* = (UCP - MUCP) * EMF * PHperUCP
	6.3. Estimate total effort.	E = *E_primary* + *E_secondary*

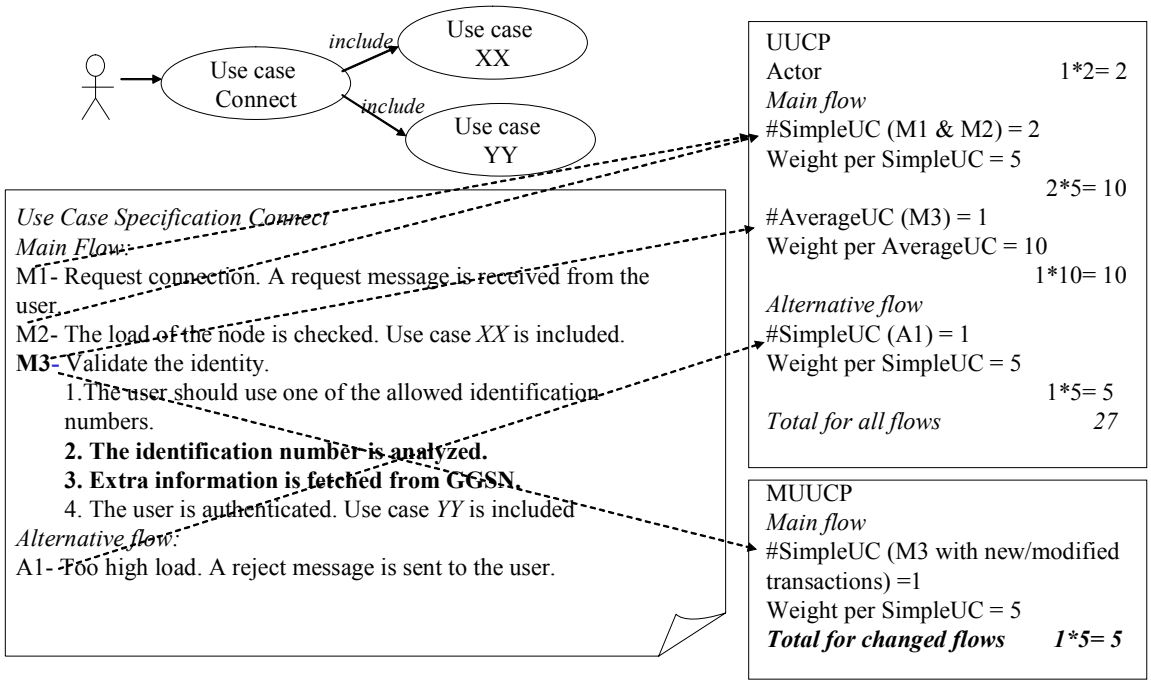

Figure 2. Example of counting UUCP and MUUCP for a use case called *Connect*

The above values give AAF (Adaptation Adjustment Factor) = 48%. Thus, EMF (AAM in Figure 1) will be equal to 56%. The reason for renaming the factor is to distinguish between modification of code from a previous release and systematic software reuse in COCOMO II. We have not found any empirical studies that evaluated such a factor in incremental development. This factor gives equivalent UCP for secondary modifications.

In Rule 6.2, we multiply EMF with the size of use case points that are not modified or new. 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 evaluate the UCP method, we used use case points also in this step.

6. ESTIMATION RESULTS

The estimation method was modified based on the use cases and effort of one release (called Release 1 here building on Release 0) and was later tested on the successive release (called Release 2). Release 2 was the latest release of the system at the time of this study.

Of the 23 use cases in Release 1, seven use cases were not modified, one use case was new, while 15 use cases were modified. We broke down the use cases, inserted the number of use cases, actors, exceptions and parameters in spreadsheets in Microsoft Excel, counted the use case points and estimated the effort following the rules in Table 3. Table 4 shows that after break-down of the use cases, we ended up with 288 use cases in Release 1.

Table 4 also shows data for Release 2, which had 21 use cases: two use cases were not modified, one use case was new, while 18 were modified. Note that three use cases are missing in Release 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.

Table 4. No. of use cases in each class

	Release 1	**Release 2**
Simple use case	170	95
Average use case	83	100
Complex use case	35	59
Sum use cases	*288*	*254*
Modified simple use case	57	81
Modified average use case	18	16
Modified complex use case	2	11
Sum modified use cases	*77*	*108*

We decided to compensate for not counting the environmental factors and for the large number of complex use cases, by using the maximum used number of PHperUCP that is 36. Nevertheless, the estimates were almost half the actual effort spent in Release 1 for all activities. Therefore we compared our estimates with the projects in Table 2 with respect to what the estimate should cover.

For 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 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

configuration management, software process adaptation, documentation and travel. Effort distribution profiles will of course vary depending on environment and technologies. In our case, Development before System Test (use case modeling, analysis and design, coding, module testing and use case testing added by Ericsson) counts for half the total 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 in Table 2

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

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

Release	Development before System Test	System Test	Project Management	Other
1	50%	25%	10%	15%
2	55%	19%	11%	15%

For confidentiality reasons, we cannot give the exact figures for the estimates and the actual effort. However, our UCP estimates were 21% lower for Release 1 and 17% lower for Release 2 than the actual effort, the latter being around 100 person-years; i.e. fairly accurate for large projects (all use cases used in the estimation are actually implemented).

For comparison, the effort to develop the first release was 2-3 times this number. The expert estimates for Release 2 were 35% lower than the actual effort, and thus the adapted UCP method seems to have lower relative error than expert estimates.

The effort for estimating each release was approximately two person-days, but we spent also a few days initially on modifying the method. The method was easy to learn, but assigning values to factors and deciding classification rules need access to experts, empirical data and guidelines for writing use cases.

7. DISCUSSION OF THE RESULTS

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 of this study and the second author had previous experience with the UCP method. Although the method was presented to a few project leaders, we could not involve them in the adaptation work due to internal reorganizations.

The adapted UCP method produced reasonable estimates with the following changes divided in three groups:

1) Generic changes that should be considered in every project (modified the UCP method):

a. **Size and the level of detail in use cases**. There is no standard way of writing use cases. We broke each use case down to several smaller ones to compensate for the size and we modified the classification rules, justified by the complexity of transactions.

b. **Technical and environmental factors**. These factors are incorporated in PHperUCP, OF and EMF. Since the method is adjusted to the context and is used on successive releases of the same system, technical and environmental factors are redundant.

2) Changes specific to incremental development (added to the UCP method):

c. **Incremental modification of use cases**. We estimated effort for new and modified transactions, actors or parameters to estimate effort for primary changes.

d. **The Equivalent Modification Factor (EMF)**. This factor is used to estimate equivalent use case points for modification of software from a previous release or secondary changes, including perfection of quality attributes. It is set to 56% in this study.

3) Changes specific to the development organization (extended the UCP method):

e. **The Overhead Factor (OF)**. We used the largest value of PHperUCP proposed by other studies, i.e. 36. It covered effort for use case modeling, analysis and design, coding, module testing and use case testing; i.e. Development before System Test. OF is used to estimate the total effort based on the effort for Development before System Test. This empirically derived factor is set to 2.

An alternative to OF would be to use 72 PHperUCP. We chose to add this factor to the method to highlight the diseconomy of scale in large projects and the impact of effort distribution profile on the estimation method. EMF, PHperUCP and OF rely on empirical observations and our judgments, and can be subject of further adjustments.

All estimation methods are imprecise, because the assumptions are imprecise. The method was adapted using data from one release, but it gave even better results for the successive release. Each estimate should also come with a range, starting with a wider range for early estimates. Use cases are updated in the early requirement specification stage, which gives a range of 0.67 - 1.5 estimated effort according to COCOMO [5].

The impact of *E_secondary* is large on the total estimated effort (65% in Release 1 and 55% in Release 2). In addition to portions of the software affected by changed use cases, the rest of the software is also modified to improve quality, and there are in general change requests and defect reports that were not pre-planned. Parts of the software may become more stable after a few releases or be used in a black-box style without modification, so that *E_secondary* decreases. But software also decays and there is cost related to re-factoring and redesign.

When releases are normally planned in relatively constant intervals, a planned release will take approximately the same effort as the previous one. The question is then whether all the

assigned use cases can be implemented by available resources and whether the project has taken into account the impact of secondary changes. Therefore, the method is still useful.

8. ANSWERS TO RESEARCH QUESTIONS

Four research questions were presented in Section 4. We answer these 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. One alternative is to include examples of typical use cases in the method, such as those defined in [9].

RQ2. Is it possible to apply the UCP method to incremental changes in use cases? Yes. Rules 1.2, 2.6, 3.2, and 6.1 in Table 3 show how to estimate effort for changes in use cases.

RQ3. How can effort to modify software from a previous release be estimated? We used the COCOMO II formula for adapted software, calculated EMF and applied it on use case points for the rest of the software. The EMF may be adjusted to the context.

RQ4. Does the method produce reasonable results in this industrial setting? The adapted UCP method fitted well into the adapted RUP process and produced reasonable results. The method is cheap and understandable.

9. RELATED WORK

Work in effort estimation for *ab initio* software development has been reported since the mid-sixties. Work has also been done to predict effort needed for software maintenance, see [17] for an overview. Different models use the history or a prediction of change impact (as number of modules, FP, modification requests, SLOC or change traffic) as input. Maintenance or evolution in these models is unplanned changes to software after *ab initio* development, applicable e.g. to each release of a system in incremental development.

Carbone et al. propose combining data from use case diagrams, class diagrams and state diagrams in an automated tool for effort estimation [7]. The method has a *Fast* estimation when there are few details in these diagrams and a *Serious* estimation later. There are many factors that should be set empirically, it depends on an OO paradigm for modeling and design, and incremental development is not discussed. The authors also present earlier work, basically based on class diagrams.

Ashman suggests predicting effort for implementing each use case in person-days and summing up these to estimate effort for an iteration [4]. The predictions are expert opinions rather than rule-based. He makes two observations that are applicable also to our model. Firstly, a use case encompasses a discrete and significant proportion of a system's functionality. Therefore it is easier to estimate effort using these large functional chunks. Secondly, it is possible to compare the estimated and actual effort at the end of each iteration (or each release in this case) and tune the model to fit the project.

An important question is whether multiple estimation techniques should be applied to achieve a better result. MacDonell et al. have compared three estimation methods on effort data from a medical records information system [15]. They observed that the best method varies from one study to another. Since they could not define *a priori* which technique is best for each case, a combination of techniques is recommended.

10. CONCLUSION

An effort estimation method based on use cases has been adapted and tested on a large industrial system with incremental changes in use cases. One main assumption is that use cases may be used as a measure of the size of a system, and that changes in these may be used as a measure of changes in functionality between releases. Generally, predicting the size of a software system in SLOC or FP in early phases is as difficult as predicting the needed effort. For changes not reflected in use cases, an additional model is used.

The method does not depend on any tools (although there are tools for the original UCP method), paradigms or programming languages, and can promote high quality use cases. The method is cheap, transparent and easy to understand. The method is also suitable when development is being outsourced.

The main contributions of the study are:

- **Discussing how to adapt the UCP method for a large industrial project and to a specific context.** The proposed changes for breaking down use cases and new classification rules were necessary since use cases are written with different level of details in different projects. Future work may provide some example use cases that may be used for calibrating classification rules to a context. For a model to scale up, additional factors such as a higher value of PHperUCP and the Overhead Factor (OF) may also be necessary.
- **Adapting the UCP method for incremental development.** Two mechanisms of effort consumption are identified: a) primary changes reflected in changes in use cases, and b) secondary changes or modification of software from a previous release as a ripple effect of primary changes, unplanned changes and improvements in quality attributes.
- **Identifying the impact of effort distribution profile on estimation results**, reflected in the Overhead Factor (OF).

Use case diagrams are usually available before other UML diagrams, but have variable level of details. A challenge is to define some standards for these. 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 as the estimated effort for secondary changes.

There are few empirical studies on estimation of incrementally developed projects. The UCP method showed flexibility in adapting to the context, but there are many project-specific factors in the original method and in our extensions of it. Future studies can help to understand the degree of modification in incremental development of a system (use cases, code and integration costs), and how the method works on other types of systems.

11. ACKNOWLEDGEMENTS

The study was performed in the INCO project (INcremental and COmponent-based Software Development), a Norwegian R&D

project in 2001-2004 [http://www.ifi.uio.no/~isu/INCO/]. It was part of the first author's PhD study, which was performed in Ericsson, Grimstad- Norway. We thank Ericsson for the support.

12. REFERENCES

[1] Albrecht, A.J. Measuring application development productivity. In *Proceedings of the IBM Applic. Dev. Joint SHARE/GUIDE Symposium,* Monterey, CA, 1979, 83-92.

[2] Anda, B., Dreiem, D., Sjøberg, D.I.K., 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 Int'l Conference.* Springer-Verlag LNCS 2185, 487-502.

[3] Anda, B. Comparing effort estimates based on use cases with expert estimates. In *Proceedings of Empirical Assessment in Software Engineering (EASE 2002)* (Keele, UK, April 8-10, 2002), 13 p.

[4] Ashman, R. Project estimation: a simple use-case-based model". *IT Pro*, 6, 4 (July/August 2004), 40-44.

[5] 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

[6] Boehm, B., Brown, W., Madachy, R., Yang, Y. Software product line cycle cost estimation model. In *Proceedings of the ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE 2004)* (Redondo Beach CA, USA, 19-20 August 2004), IEEE-CS Order No. P2165, 156-164.

[7] Carbone, M., Santucci, G. Fast&&Serious: a UML based metric for effort estimation. In *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02)* (Spain, June 11, 2002), 12 p.

[8] Chen, Y., Boehm, B.W,, Madachy, R., Valerdi, R.. An empirical study of eServices product UML sizing metrics. In *Proceedings of the ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE 2004)* (Redondo Beach CA, USA, 19-20 August 2004), IEEE-CS Order No. P2165*,* 199-206.

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

[10] 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. Software Engineering*, 27, 1 (January 2001), 1-12.

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

[12] Karner, G. Resource estimation for Objectory projects. Objective Systems SF AB (copyright owned by *Rational Software)*, 1993.

[13] Kemerer, C.F. An empirical validation of software cost estimation models. *CACM*, 30, 5 (May 1987), 416- 429.

[14] Lehman, M.M., Perry, D.E. Ramil, J.F. Implications of evolution metrics on software maintenance. In *Proceedings of the Int'l Conference on Software Maintenance (ICSM 1998)*, IEEE-CS Press, 208-219.

[15] MacDonell, S.G., Shepperd, M.J. Combining techniques to optimize effort predictions in software project management. *Journal of Systems and Software*, 66, 2 (May 2003), 91-98.

[16] Mohagheghi, P., Conradi, R. An empirical study of software change: origin, impact, and functional vs. non-functional requirements. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004)* (Redondo Beach CA, USA, 19-20 August 2004), IEEE-CS Order No. P2165*,* 7-16.

[17] Ramil, J.F., Lehman, M. M. Metrics of software evolution as effort predictors- a case study. In *Proceedings of the Int'l Conference on Software Maintenance (ICSM 2000)*, IEEE-CS Press, 163-172.

[18] http://www.processwave.net/, July 2004.

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

[20] Smith, J. The estimation of effort based on use cases. *Rational Software*, White paper, 1999.

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

[22] Uemura, T., Kusumoto, S., Inoue, K. Function point measurement tool for UML design specification. In *Proceedings of the 6th Int'l IEEE Software Metrics Symposium*, IEEE-CS Press, 1999, 62-69.