Change Effort Estimation based on UML Diagrams Application in UCP and COCOMOII

Dhikra Kchaou[1], Nadia Bouassida[1] and Hanene Ben-Abdallah[1,2]
[1]*University of Sfax, Sfax, Tunisia*
[2]*King Abdulaziz University, Jeddah, K.S.A.*

Keywords: Effort Estimation, Change, COCOMO, UCP.

Abstract: Change impact must be accounted for during effort estimation to provide for adequate decision making at the appropriate moment in the software lifecycle. Existing effort estimation approaches, like the Use Case Point method and the COnstructive COst MOdeL, estimate the effort only if the change occurs at one level, for example when a new functionality is added (at functional level). However, they do not treat elementary changes at the design level such as the addition of a class or a sequence diagram; because they incur several modifications at different modelling levels, such changes are important to account for in effort estimation during the software development. In this paper, we take advantage of intra and inter UML diagrams dependencies, first, to assist developers in identifying the necessary changes that UML diagrams must undergo after a potential change, and secondly to estimate the necessary effort to handle any elementary change e.g. adding a class, an attribute, etc. We use our traceability technique in order to adapt the UCP and COCOMO methods to estimate the effort whenever a change occurs during the requirement or design phases.

1 INTRODUCTION

Software systems are inevitably subject to continuous changes during their development as well as their exploitation. A change during any phase of a software lifecycle often impacts several artifacts/models of the software (requirements, design, code) and incurs additional effort (i.e., cost) to handle it. Every change must be therefore analyzed to account for its impacts and incurred effort in order to provide for an adequate decision making in the software project management. A change impact analysis and management technique should provide for both the identification of the effects of every change type on *all* software artifacts and activities, and the estimation of the costs incurred to handle these effects. The effort/cost estimation ensures the success of the software development/maintenance project and can be used to decide whether to undertake the change or to cancel it. Several effort estimation methods have been adapted to estimate changes in order to identify the factors that should be included in a modified software—e.g. the Use Case Point method (UCP) (Karner, 1993), the COnstructive COst MOdeL (COCOMO II) (Boehm, 2000). However, current effort estimation methods face two essential limits:

1. when they rely on the code which is produced relatively late, their results must be revised each time a change in the requirements or design happens. In addition, the projection of code changes onto design/requirement changes is not straightforward because it is programming language and style dependent. This limit makes code-based estimation methods unable to offer on-time and precise support for adequate project management; and
2. because they rely on one level of software modelling (functional level for UCP and code level for COCOMO), current estimation effort methods do not account for changes introduced in or incurred on other software artefacts during all the development phases—for instance, adding a class to the design or an interaction in a sequence diagram.

Our objective is to provide for a means to estimate the effort required to deal with a change at the requirements and/or the design level, knowing that a change may be elementary (affecting a class or an attribute) yet it can affect the overall development effort. To attain our objective, first, we make use of a graph-based approach that ensures traceability among the different software artifacts (e.g., the class, sequence and use

Kchaou D., Bouassida N. and Ben-Abdallah H..
Change Effort Estimation based on UML Diagrams Application in UCP and COCOMOII.
DOI: 10.5220/0005510503010308
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 301-308
ISBN: 978-989-758-114-4
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

case diagrams) in order to identify all impacted elements. Secondly, we show how this information can be used with UCP and COCOMO II to estimate the effort in terms of work hours. (Note that our traceability approach applies to any method of effort estimation.)

More specifically, our approach exploits intra and inter UML diagrams dependencies to identify the affected elements. It explicitly encodes the intra and inter diagram semantic and syntactic dependencies and integrates them into a model dependency graph (MDG) to identify the affected elements at any modelling level. Thanks to the model dependency graph, for instance, when a change occurs in a design diagram (class or sequence diagrams), affected elements in a use case diagram can be determined in order to be able to apply the effort estimation methods.

The remainder of this paper is organized as follows: Section 2 first overviews the steps in the UCP and COCOMOII methods. Section 3 shows how the model dependency graph integrating various UML diagrams can be used to identify all elements affected by a change at any modelling level. Section 4 illustrates the application of COCOMOII and UCP through an example. Section 5 concludes with a summary of the presented work and a highlight of its extensions.

2 EFFORT ESTIMATE METHODS: AN OVERVIEW

In this section, we first overview the most known effort estimation methods which are adapted to changes: the Use Case Point (UCP) method (Karner, 1993) and the COnstructive Cost MOdel II (COCOMOII) (Boehm, 2000). Secondly, we discuss those change impact analysis approaches that calculate the effort needed to handle a change.

2.1 Change Effort Estimation using COCOMO II

COCOMO (Boehm, 2000) was first introduced by Barry Boehm in 1981. It takes software size and a set of factors as input and it estimates effort in *person-months* according to equation **(1)**:

$$PM = A^* \, size^{E} * \Pi \, EM_{i} \tag{1}$$

where:

- A = 2.94 for COCOMO II.2000;
- the size is in Kilo Source Lines Of Code (KSLOC). It is derived from estimating the size of software modules that will constitute the application program. It can also be estimated from unadjusted function points (UFP) converted to SLOC (Boehm, 2000) (Kama et al., 2013);
- the exponent E is an aggregation of five scale factors (SF) that account for the relative economies or diseconomies of scale encountered for software projects of different sizes:
$$E=B+0.01 * \sum_{j=1}^{5} SFj$$
with B = 0.91 for COCOMO II.2000; and
- the effort multipliers (EM_i) are used to adjust the nominal effort, *person-months*, to reflect the software product under development. For instance, the Required Software Reliability (RELY) is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only a slight inconvenience, then RELY is very low.

All the input parameters of the software cost estimate model need to be consistent and available in the early stages of a software project. However, very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project, or the detailed specificities of the process to be used.

Kama et al., (Kama et al., 2013) developed a new change effort estimation model based on COCOMOII. This contribution integrates the change impact analysis technique SPD-CIF (Kama, 2011) and COCOMO II effort estimation technique. Based on the traceability between requirement, design artefacts and classes, the impact analysis is performed. To estimate the change effort for a given change, the modified, added, and canceled KSLOCs are determined. Weights are assigned to change types (addition, modification, cancellation). To calculate the change effort for every change type, the constant *A* (in equation **(1)**) is replaced by the weight of the change type and the *size* is replaced by the amount of changed (added, modified or cancelled) KSLOCs. Finally, the effort in person/months of every change type is summed to obtain the overall change effort. The change type's weights are not justified. In addition, the effort of the deletion is subtracted from the total effort (weight equals -1) which is not true in terms of development since a deleted element incurs changes in the models and hence effort to update them.

2.2 Change Effort Estimation using UCP

Introduced in 1993 by Karner (Karner, 1993), the Use

Case Points (UCP) method estimates effort in *person-hours* based on use cases specifying the functional requirements of a system. UCP is an adaptation of Function Point (Albrecht, 1979) for measuring the size of projects whose functional requirements specifications are described by a use case model. The design of UCP takes into account three aspects of a software project: Use cases, Technical qualities, and

Table 1: Adapted UCP estimation method.(Mohagheghi, 2005).

	Steps	Output
1	1.1. Classify all actors as average, WF = 2.	Unadjusted Actor Weight UAW = #Actors * 2
	1.2. Count the number of new/modified actors.	Modified Unadjusted Actor Weight(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)	Modified UUCW (MUUCW) = ∑(#new or modified Use Cases * WF) + ∑(# new or modified exceptional flows and parameters)
3	3.1. Calculate UUCP for all software.	UUCP = UAW + UUCW
	3.2. Calculate Modified UUCP (MUUCP)	MUUC = 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	Estimate effort for new/modified use cases	Effort = MUCP * PHperUCP

Development resources. Each actor and each use case is classified at a complexity level (simple, average, or complex) and assigned a corresponding weight (from one to three points for the actors, and from 5 to 15 for the use cases). The technical qualities of UCP are represented by a Technical Complexity Factor (TCF), which consists of 13 technical qualities, each with a specific weight (zero is "not applicable" and five is "essential"), combined into a single factor. The development resources for UCP are represented by the Environment Factors (EF). The UCP model identifies eight such factors contributing to the effectiveness of the development team. To calculate the EF, an expert must assess the importance of each factor and classify it on a scale from zero to five (zero meaning "very weak" five meaning "very strong").

The UCP method was adapted by Mohagheghi et al. (Mohagheghi, 2005) to incremental development. Table 1 summarizes the steps of the adapted UCP method as proposed in (Mohagheghi, 2005). This UCP method has the advantage of measuring easily the software functional size as early as possible in the development cycle. Nevertheless, this method has many problems, for instance, the same category labels are used for use cases (simple, average, and complex). However, it cannot be assumed that the categories and the categorization process are similar, since different entity types are involved (Abran, 2010). In addition, both the technical and resource factors are evaluated through a categorization process with integers from 0 to 5. Note that these numbers do not represent numerical values on a ratio scale, but merely a category on an ordinal scale type; that is, they are merely ordered labels and not numbers. For instance, a programming language assigned a difficulty level of 1 is considered to be less difficult for the development team than a programming language of difficulty level 2, but it cannot be considered to be exactly one unit less difficult than a programming language categorized as having a difficulty level of 2, since these levels are measured on an ordinal scale (Abran, 2010).

2.3 Effort Estimate in Change Impact Management Approaches

Several works tried to calculate the cost of repairing inconsistencies caused by changes. For instance, Dam et al. (Dam, 2010) proposes an approach to support change propagation within UML design models. This approach first generates repair plans for each detected consistency constraint. It then calculates the cost of each generated repair plan instance to propose the cheapest repair to the designer. Finally, the selected repair plan instance is executed to update the design model. This approach assumes that repair plans that lead to fewer changes in the model, and thus have lower costs, are preferable.

Briand et al. (Briand, 2006) propose another way of calculating the cost of a repair. They define a distance between the changed elements and potentially impacted elements that represents the number of impact analysis rules invoked to identify the impacted elements. Based on this distance, they assumption that the larger the distance is, the less likely the model element is to be impacted.

Sharif et al. (Sharif, 2012) compute the effort in terms of the total working hours needed to implement a requirement change. Based on an empirical investigation, a regression equation is derived by performing correlation and regression analysis on the change request data.

Overall, the above examined approaches estimate the effort of changes in software artefacts in a particular development phase and do not account for changes incurred in other related software artefacts. To account for change propagation among artefacts, we propose an adaptation of COCOMO and UCP estimate models to estimate the effort needed when changes occur in the design and/or the requirements.

3 CHANGE IMPACT ANALYSIS ACROSS ARTEFACTS

Our approach to effort estimate identifies and measures the potential side effects of changes across different UML diagrams; the focus in this paper is on the use case, class and sequence diagrams which cover the requirements and design phases where changes are frequent. To identify the dependencies among these UML diagrams, our method adopts a graph-based technique to construct automatically a model dependency graph (MDG). Based on the change type and the MDG produced for the UML diagrams, our method determines the impacted elements for every change type. These elements are then used to estimate the effort need to handle the overall changes using any effort estimate method; we illustrate in this paper the use of COCOMOII and UCP.

3.1 Traceability among the Use Case, Class and Sequence Diagrams

Based on the fact that UML diagrams can be assimilated to graphs, the dependencies among UML diagram elements could therefore be determined using a graph reachability analysis technique. Indeed, inspired from the work of Lallchandani et al., (Lallchandani, 2009) for static slicing of UML models, we defined a method to construct the model dependency graph (MDG), which is used to ensure traceability between the class, sequence and use case diagrams.

The UML class diagram is transformed into a Class Dependency Graph (CDG) and every UML sequence diagram is transformed into a Sequence Dependency Graph (SDG). In addition, every UML use case diagram is transformed into a Use Case Dependency Graph (UCDG) based on a structured use case description (Ali, 2005). To get all dependencies among the various diagrams, the UCDG, CDGs and SDGs are merged into one Model Dependency Graph (MDG).

Our CDG and SDG construction and integration method adapts the transformations initially proposed by (Lallchandani, 2009). To trace the change impact from the use case diagram across the class and sequence diagrams, the UCDG, CDG and SDGs are integrated into a single graph called Model Dependency Graph (MDG) using an information retrieval technique. In fact, we use the cosine similarity measure to identify the correspondence among the ordered actions and data objects (specifying the use case scenarios) and the information present in the sequence diagrams.

3.2 Example: The ATM System

To illustrate the effort estimation and the traceability through the MDG, let us consider the automatic teller machine (ATM) system example (Russel, 2004). The use case diagram comprises, essentially, four use cases (Figure1): “System startup”, “system Shutdown”, “Session” and “Transaction”. The “Session” UC and the “Transaction” UC textual descriptions are presented respectively in Table 2 and Table 3.

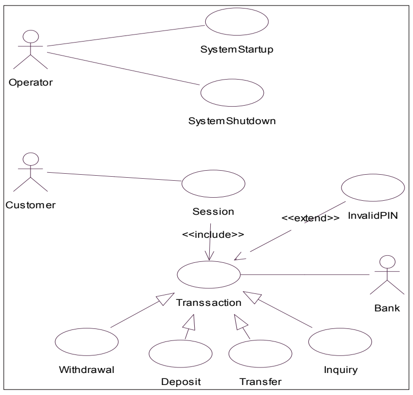

Figure 1: Main use-case diagram of the ATM system (Russel, 2004).

The documentation of the use cases can be formalized through the sequence diagrams shown in Figure 3 and 4 where the objects are instances of the classes shown in the class diagram of Figure 2.

Table 2: UC1: Use case "Session" description.

+	**Session**
Actor	Customer
Precondition	ATM card is inserted into the card reader slot of the machine
Postcondition	Nothing
Extension Point	[**Pin valid**], use case «Transaction »
Normal Scenario	NSa1.<The customer> < enters the card into the machine> NSa2. <The ATM> <reads the card > **NSa3.< The customer > < enter his/her PIN>** **NSa4.< The customer > <chooses from a menu the type of transaction>** **NSa5. < The customer > < performs a transaction>** **NSa6.<The ATM> <ejects the card and ends the session>**
Alternatives Scenario	<the reader cannot read the card, **restart from NSa2**> <AS1a1> <the ATM> <reject the card and displays an error screen>
Error Scenario	< too many invalid PIN entries> <ES1a1> <the ATM> < abort the session with the card being retained in the machine >

Table 3: UC2: Use case "Transaction" description.

Use case	**Transaction**
Actor	Bank
Precondition	the customer chooses a transaction type from a menu of options
Postcondition	Nothing
Extension Point	[PIN is invalid], use case « invalid PIN »
Normal Scenario	NSa1.<The customer> < furnish appropriate details (e.g. account(s) involved, amount)> NSa2. <The ATM> < send to the bank the information from the customer's card and the PIN the customer entered> NSa3. < The ATM > < perform the transaction> NSa4. < The ATM > <print a receipt> NSa5. < The ATM > < display a menu of possible types of transaction >
Alternatives Scenario	<the reader cannot read the card, **restart from** "session" use case> AS1a1. <the ATM> <reject the card and displays an error screen>
Error Scenario	< too many invalid PIN entries> ES1a1. <the ATM> < abort the session with the card being retained in the machine >

Let us suppose that the designer wants to make the following change to the class diagram presented in Figure 2 delete the "session" class. Based on the MDG of Figure 5, the impacted elements in the use case diagram can be determined: In the use case "session", the scenarios NSa3, NSa4, NSa5, NSa6 are affected. The messages/actions I(k) from the deleted class "session" are impacted by the change.

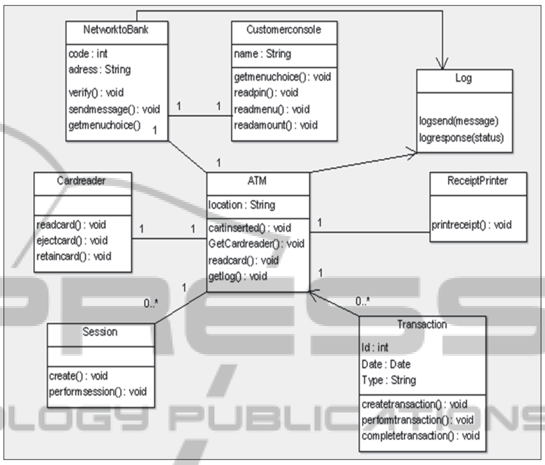

Figure 2: ATM system class diagram (CD) (Russel, 2004).

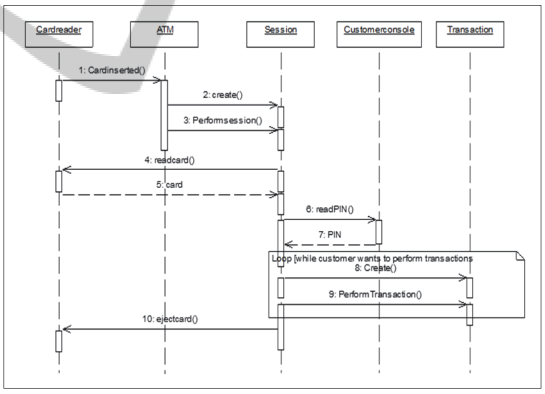

Figure 3: The sequence diagram "Session" (Russel, 2004).

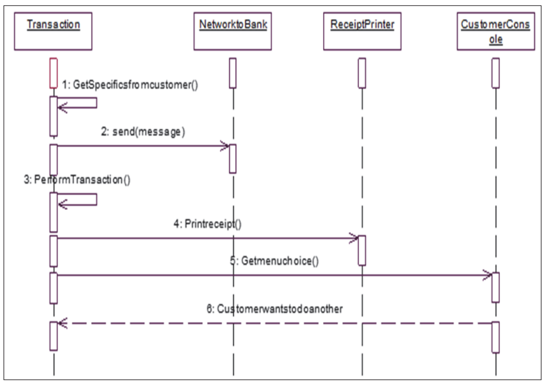

Figure 4: The sequence diagram "Transaction" (Russel, 2004).

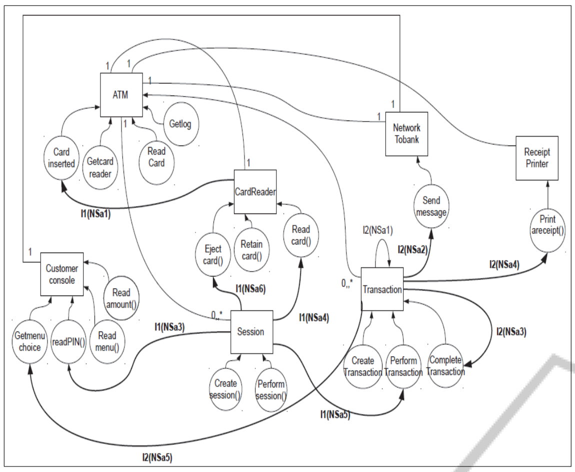

Figure 5: The model dependency graph integrating the CD, UC1 and SD1.

3.3 UCP Application

Based on the impacted elements detected in the use case diagram through the MDG, we can apply the adapted UCP estimation method of Mohagheghi et al. (Mohagheghi, 2005). Recall that the change (deleting a class) occurs in the class diagram whereas the UCP method depends on use cases. Our proposition has the merit of determining the affected use cases (via the MDG) in order to apply UCP. The application of the adapted UCP is presented in Table 4.

Table 4: Example of counting UUCP and MUUCP for the use case “session”.

Steps	UUCP & MUUCP values
1	UAW= 3*2= 6
	MUAW = 0
2	*Normal scenario/Main flow* #simple use cases =6 #Average use cases=0 #complex use cases = 0 Weight per simple use case=5 6*5=30 *Alternative flow* #simple use cases =1 Weight per simple use case=5 1*5=5 *Error flow* #simple use cases =1 Weight per error use case=5 1*5=5 **Total =40**
	Normal scenario(Main flow) #Average use cases=1 (the use case session with 4 new or modified action) Weight per Average UC = 10 1*10=10 **Total for changed flow = 10**
3	UUCP = 6+40 = 46
	MUUC = 0+10=10
4	TCF = EF = 1.
5	UCP = UUCP = 46
	MUCP = MUUCP =10
6	Effort = 10 *36 =360

To derive an appropriate duration for the project, we need to know the team’s rate of progress through the use cases. The literature on the UCP method proposes from 20 to 36 PHperUCP. Mohagheghi et al. (Mohagheghi, 2005) decided to ignore counting the environmental factors and decided for the large number of complex use cases to choose the maximum used number of PHperUCP that is 36. This means that our example of 10 use case points corresponds to 360 hours of work.

Now let us suppose that a developer spend about 40 hours per week on project tasks. As a consequence, he will spend about two month (360/40= 9 weeks) to manage this change (the deletion of the “session” class).

3.4 COCOMO II Application

To apply the COCOMOII model, we need the size parameter expressed in SLOC. Since the change occurs in a design diagram, the size of changed elements is calculated first in function points and then it is converted to SLOC.

3.4.1 Function Points Calculation

The calculation of function points is based on the principle of Albrecht’s function point analysis (FPA) (Albrecht, 1979) where a system is decomposed into functional units. The inputs represent information entering the system, and the outputs represent information leaving the system. The enquiries represent requests for instant access to information. The internal logical files represent information held within the system. Finally, the external interface files represent information held by other systems used by the system being analyzed.

Information domain of the use case “session”:

The information domain treats only the part of the use case affected by the change. Based on the MDG in the Figure 5, the deletion of the use case “session” impacts the actions NSa3, NSa4, NSa5, NSa6. The information domain of these actions is extracted from the lines in bold in Table 2.

External Inputs: the PIN, the selected transaction.
External Outputs: the types of transactions.
External inquiries: 0.
Internal logical files: 0.

External interface files: error screen, type of transaction interface.

Table 5: Functional units with weighting factors.

Information Domain	Weights		
	Simple	Average	Complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

***Function point calculation*:**

Function points (FP) are calculated based on the functional units extracted from the actions affected by the change multiplied by their weights as defined in Table 5:

FP =2*3 (simple External Inputs) +
1*4 (simple external outputs) +
2*5 (simple external interface files)
= **20 *function-points*.**

For instance, in the case of the C++ programming language, 1 FP = 29 LOC (Boehm, 2000), which gives the following estimated size (ES):

ES =20* 29 = 580= **0.56 KLOC**

Note that function points describe what the application interacts with; but there is other implicit information in addition to the interactions. We have to scale the function value according to expectations. These expectations are presented by scale factors and cost drivers. The values of scale factors and cost drivers are usually affected by experts based on their judgment (Mohagheghi, 2005). In our work, we suppose that the impact of the scale factors and cost drivers is insignificant and it does not have a large impact on the estimate in this particular example. Dropping these factors is also suggested in other cost models (Kemerer, 1987) (Mohagheghi, 2005).

3.4.2 Scale Factors Calculation

We suppose that all scale factors are nominal. The nominal scale factors values presented in COCOMOII 2000 (Boehm, 2000) are summed up to calculate the exponent E.

$$E=B+0.01 * \sum_{j=1}^{5} SFj$$
$$E=0.91+0.01*18.67= 1.0997$$

3.4.3 Effort Multiplier Calculation

Similarly to the scale factors, we suppose that cost drivers are nominal. So, based on the values presented by COCOMOII (Boehm, 2000), the effort multiplier product is taken as 1.00.
Finally, the effort in terms of *person-per-months* can be calculated:

$$\text{Effort} = 2.94 * (0.56)^{1.0997} * 1 = 1.55$$

3 EMPIRICAL INVESTIGATION

We evaluated our method through an empirical evaluation based on a comparison between results (effort needed to correct a change) built by applying our method and results constructed by experts. Experts are UML professionals and have years of experience studying and developing projects designed with UML.

More specifically, we choose six designers who are responsible in updating projects (versioning, design improvement, error handling) and have already made and managed changes. Our evaluation consists in collecting information (actual effort, weighting factors, programming language, etc.) for two change types in order to calculate the effort based on our adapted UCP and COCOMOII approach.

Table 6 presents a comparison between the efforts (The number of weeks needed to manage a change) estimated by experts (E1, E2…, E6) and effort estimated by our adapted COCOMO II and UCP estimation techniques.

Table 6: A comparative study.

	Experts' estimate		Adapted COCOMOII		Adapted UCP	
	C1	C2	C1	C2	C1	C2
E1	4,5	2,5	6	2,5	9	3,5
E2	2	3	2,5	3	5	5
E3	10	8	8	8	15	10
E4	6	2	5.5	3	8	5
E5	14	6	13	3	9	10
E6	2	7	3	6,5	5	10

Our preliminary evaluation affirmed that compared to experts' estimate, COCOMOII estimate is closer to the reality than UCP.

4 CONCLUSIONS

This paper illustrated the feasibility of integrating an effort estimate method with a change impact analysis technique early in the software development life cycle. Effort estimate during the accommodation of a change helps the designer and/or project manager to decide whether to undertake a change or to cancel it. Such a decision is needed during the requirements and/or design phases since most often changes occur during these phases and impact the overall software project management.

Our integration approach explored the semantic relations among the UML diagrams to ensure the traceability of a change in one diagram in other diagrams. This paper illustrated through an example how COCOMOII and UCP estimate models can be adapted to estimate the effort needed when changes occur in the design (class diagram) and/or the requirements (use case diagram).

Besides conducting a large-scale experimental evaluation of our propositions, we are in the process of extending the adaptation of COCOMO and UCP to cover changes in the remaining UML diagrams.

REFERENCES

Abran, A., 2010. *Software Metrics and SoftwareMetrology*. Wiley-IEEE Computer Society Press.

Albrecht, A.J., 1979. Measuring Application Development Productivity. *In: Proceedings of Joint Share, Guide and IBM Application Development Symposium*.

Ali M., Ben-Abdallah H., Gargouri F., 2005. Towards a Validation Approach of UP Conceptual Models. *In: Proceeding of Consistency in Model Driven Engineering in European Conference on Model Driven Architecture - Foundations and Applications Nuremberg.*

Boehm, B., Brown, W., Madachy, R., Yang, Y., 2000. COCOMO II Model Definition Manual, Center for Software Engineering, Version 2.1. *http://csse.usc.edu/csse/research/COCOMOII/cocomo 2000.0/CII_modelman2000.0.pdf.*

Briand, L. C., Labiche, Y., O'Sullivan, L., & Sówka, M. M. 2006. Automated impact analysis of UML models. J. System Software,

Dam H. K., Winikof M.: Supporting change propagation in UML models, 2010. IEEE International Conference on Software Maintenance (ICSM).

Karner, G., 1993. *Resource Estimation for Objectory Projects*. Objective Systems SF AB.

Kama, N., Halimi, M., 2013. Extending Change Impact Analysis Approach for Change Effort Estimation in the Software Development Phase. *In 13th WSEAS International Conference on APPLIED COMPUTER SCIENCE*.

Kama, N.M., 2011. A change impact analysis framework for the software development phase. *Thesis, University of Western Australia*.

Kemerer, C.F., 1987. An empirical validation of software cost estimation models. *In: Communications of the ACM magazine*. Volume 30, Issue 5.

Lallchandani, J.T., Mall, R., 2009. Static Slicing of UML Architectural Models. *Journal of object technology*, Vol. 8, No. 1.

Mohagheghi, P., Anda, B., Conradi, R., 2005. Effort estimation of use cases for incremental large-scale software development, *In: 27th International Conference on Software Engineering*, IEEE.

Russell C., 2004. Bjork, Gordon College, Copyright©2004, http://www.math-cs.gordon.edu/courses/cs211/AT-MExample/

Sharif B., Khan S. A., Bhatti M.W., 2012. Measuring the Impact of Changing Requirements on Software Project Cost: An Empirical Investigation, IJCSI International Journal of Computer Science Issues.